

Sensur av hovedoppgaver

Universitetet i Sørøst-Norge

Fakultet for teknologi og maritime fag



Prosjektnummer: **2019-03**

For studieåret: **2018/2019**

Emnekode: **SFHO3201-1 18H Bacheloroppgave**

Prosjektnavn

Kontaktløs Posisjonssensor

Contactless Position Sensor

Utført i samarbeid med: Kongsberg Defence & Aerospace

Ekstern veileder: Stian Laugerud

Sammendrag: Denne rapporten omhandler en sensorsystem løsning med tilhørende teststasjon for en kontaktløs posisjonssensor. Sensoren skal detektere en endring i vinkelposisjon på en antenne eller andre pekemekanismer på en satellitt. Undersøkelsene er gjennomført for å potensielt erstatte det nåværende systemet som er i bruk, et potensiometer design. Utviklingen er gjort gjennom analyser, tester og inspeksjon av design. Hovedområdene som er undersøkt omhandler deteksjon av posisjon gjennom en endring av frekvens.

Stikkord:

- Posisjonssensor
- Oscillatorer
- Teststasjon

Tilgjengelig: JA

Prosjekt deltagere og karakter:

Navn	Karakter
Anders Rønning	
Hans Fredrik Jamtveit	
Henrik Sæter	
Jarand Solberg Strømmen	
Magnus Berntsen Caro	

Dato: 14. juni 2019

Sigmund Gudvangen
Intern Veileder

Karoline Moholth
Intern Sensor

Aage V. Sørensen
Ekstern Sensor

Contactless Position Sensor

BACHELOR THESIS 2019

GROUP 3



KONGSBERG



Anders Rønning – Hans Fredrik Jamtveit – Henrik Sæter

Jarand Strømmen – Magnus Berntsen Caro

Published 23.05.2019

i Abstract

This report is concerned with discussing a sensor system solution with a coherent test station for a contactless position sensor. The sensor shall detect change in angular position of an antenna or other pointing mechanisms on a satellite. The study is conducted to potentially replace the current system in use, a potentiometer design. The development was done through analysis, tests and design review. Main areas researched revolves around detecting position through change of frequency, knowing the position of the sensor by the use of a precise stepper motor to count steps.

This report gives a suggestion of a theoretical solution. The sensor can achieve a theoretical precision of 0.01° by the use of a Clapp oscillator. The coils on the stator shows a change in inductance when affected by the rotor, giving the possibility to detect a change in position. This report shows the possibility of using a contactless position sensor for the detection of change in position. Further research is required regarding changes in temperature and how it affects the electrical components in the sensor.

ii Content

1	Introduction	1
2	Project description	2
2.1	Document history	2
2.2	Introduction	3
2.3	Project background	3
2.4	Kongsberg Defence & Aerospace	3
2.5	Project tasks and goals	3
2.5.1	Primary goals:	3
2.5.2	Secondary goals	3
2.6	Project Stakeholders	4
2.6.1	Stakeholder	4
2.6.2	Primary stakeholders	4
2.6.3	Secondary stakeholders	4
2.6.4	Stakeholder requirements	4
2.7	Project hierarchy	6
2.7.1	Contactless Position Sensor project team	7
2.7.2	University of Southeast-Norway	8
2.7.3	Kongsberg Defence & Aerospace	8
2.8	Project timeline	9
2.9	Updated project timeline	11
2.10	Final project timeline	13
3	Scrum	15
3.1	Document history	15
3.2	Introduction	16
3.3	Scrum values	16
3.3.1	Courage	16
3.3.2	Focus	16
3.3.3	Commitment	16
3.3.4	Respect	17
3.3.5	Openness	17
3.4	Scrum roles	17
3.4.1	Scrum Master	17
3.4.2	Product Owner	17
3.4.3	Developer Team	17
3.5	Scrum Artefacts	18
3.5.1	Product Backlog	18
3.5.2	Sprint Backlog	18
3.5.3	Increment	18
3.6	Scrum Events	18
3.6.1	Sprint	19
3.6.2	Sprint Planning	19
3.6.3	Daily Scrum	19

3.6.4	Sprint Review	20
3.6.5	Sprint Retrospective	20
3.7	Definition of Done	21
3.8	CPS approach to Scrum	21
3.8.1	Sprint	21
3.8.2	Sprint Planning	21
3.8.3	Daily Scrum	21
3.8.4	Sprint Retrospective	22
3.8.5	Definition of Done	22
3.8.6	Online tool	22
3.8.7	Roles	22
4	Work routines	24
4.1	Document history	24
4.2	Introduction	25
4.3	Asana	25
4.3.1	Asana routines	25
4.4	General work	25
4.4.1	Work hours and breaks	25
4.5	Meetings and presentation	25
4.5.1	Meetings	26
4.5.2	Presentations	26
5	Risk management	27
5.1	Document history	27
5.2	Introduction	28
5.3	Risk matrix	29
5.4	Changes in risk analysis	30
5.5	Comparison of iterations	30
5.6	Risk analysis iteration 4	31
5.6.1	Hardware risks:	31
5.6.2	Human risks:	33
5.6.3	Management risks:	35
6	Stakeholder requirements	36
6.1	Document history	36
6.2	Introduction	37
6.3	Stakeholder requirement specification	38
7	Requirements	40
7.1	Document history	40
7.2	Introduction	41
7.3	System requirements	42
7.4	Requirement specification	43
7.4.1	Functional requirements	43
7.4.2	Non-functional requirements	44

8	User Story	47
8.1	Document history	47
8.2	Introduction	48
8.3	The Three C's	48
8.4	Implementation	48
8.5	CPS User Stories	50
9	Testing	51
9.1	Document history	51
9.2	Introduction	52
9.3	Test plan	52
9.3.1	Example test ID	52
9.3.2	Test template	53
9.3.3	Testing of subsystems	53
9.3.4	Example subsystem test ID	54
9.3.5	Subsystem test template	54
10	Sensor design	55
10.1	Document history	55
10.2	Introduction	56
10.3	Oscillator	56
10.4	Oscillator tank	56
10.5	Bias network	56
10.6	LC oscillator	57
10.6.1	Colpitts	57
10.6.2	Clapp	58
10.6.3	Hartley	59
10.6.4	Amplifier	59
10.7	Design tools	60
10.7.1	OrCAD Capture CIS and OrCAD PCB editor	60
10.7.2	Electronics Explorer Board	60
10.8	CPS circuit design	60
10.8.1	First iteration oscillator	61
10.8.2	Second iteration Colpitts oscillator	64
10.8.3	Third iteration Colpitts oscillator	66
10.8.4	Fourth iteration Colpitts oscillator	68
10.8.5	Fifth iteration oscillators	70
10.8.6	Sixth iteration oscillator	72
10.9	Final oscillator iteration	74
10.9.1	Common emitter amplifier	74
10.9.2	Clapp oscillator circuit	80
10.10	CPS PCB design	87
10.10.1	Material selection in PCB	88
10.10.2	Pins and vias	88
10.10.3	Stator	90
10.10.4	Coil design	93

10.10.5	Final coil design	96
10.10.6	Rotor	99
10.10.7	PCB ordering	101
10.11	Stator and rotor PCB	101
11	Mechanical concepts	103
11.1	Document history	103
11.2	Introduction	104
11.3	Mechanical concept 1, KDA prototype	104
11.3.1	Implementation of mechanical parts	104
11.3.2	List of parts	104
11.3.3	Bracket, P.1.0.1	106
11.3.4	Cap, P.1.0.2	107
11.3.5	Assembly of KDA prototype, A.3.0.0	107
11.3.6	Step by step assembly, A.1.0.0	108
11.3.7	Design review, DR-A.1.0.0	108
11.4	Mechanical concept 2	109
11.4.1	Rotor and stator	109
11.4.2	Housing	111
11.4.3	Shaft	112
11.4.4	Rotor fixture	113
11.4.5	Assembly, A.2.0.0	114
11.4.6	Cross section, A.2.0.0	115
11.4.7	List of parts	116
11.4.8	Design review, DR-A.2.0.0	116
11.5	Mechanical concept 3	116
11.5.1	Assembly concept 3, A.3.0.0	117
11.5.2	List of parts	120
11.5.3	Housing, P-C3-3.0.1	121
11.5.4	Rotor fixture	122
11.5.5	Shaft, P-C3-3.0.4	123
11.5.6	Fixture for electrical motor, P-C3-3.0.3	124
11.5.7	Design review, DR-A.3.0.0	124
11.6	Mechanical concept 4	125
11.6.1	Assembly concept 4, A.4.0.0	125
11.6.2	List of parts	128
11.6.3	Housing, P-C4-4.0.1	129
11.6.4	Design review, DR-A.4.0.0	130
11.7	Choice of concept	131
11.7.1	2D	131
11.7.2	Tolerances	131
11.7.3	2D-drawings of final concept	133
11.8	Edits during manufacturing	140

12 Bearings	141
12.1 Document history	141
12.2 Introduction	142
12.3 General	142
12.3.1 Loads	142
12.4 Rolling bearing types	145
12.4.1 Tapered roller bearing	145
12.4.2 Cylindrical bearing	145
12.5 Ball bearing types	146
12.5.1 Single row ball bearing	146
12.6 Bearing selection	147
12.6.1 SKF 6000-2RSH	147
12.7 Bearing lubrication	148
12.7.1 Dry Moly	148
12.8 Bearing preload	149
12.8.1 Springs	150
12.8.2 Disc spring, 4293	150
12.8.3 Disc spring, 5028	151
13 Choice of materials	152
13.1 Document history	152
13.2 Introduction	153
13.3 Important elements	153
13.4 Austenitic stainless steel	154
13.4.1 Stainless steel 304, 316	154
13.4.2 Elongation of housing and shaft	154
13.5 Passivation	156
13.5.1 Nitric acid passivation	156
14 KDA motor and SMC133-2 driver	157
14.1 Document history	157
14.2 Introduction	158
14.3 Stepper motor	158
14.4 Stepper driver	158
14.5 NanoPro software - first code version	159
14.5.1 Arduino code - first code version	160
14.5.2 Connection diagram	162
14.6 SMC133-2 driver problems and possible solutions	163
14.6.1 NanoJEasy	163
14.6.2 Software development kit	164
14.7 SMC133-2 driver after researching	164
14.8 Important about the SMC133-2 driver	164
14.9 Parameters unit - SMC133-2	165
14.10 Driver nonfunctional	166
14.11 Other drivers	167
14.11.1 The EM402 - Leadshine	167

14.12SMCI33-2 driver conclusion	168
15 Final driver - DM420A	169
15.1 Document history	169
15.2 Introduction	170
15.3 Backup driver	170
15.4 Driver control	170
15.4.1 Stepper motor driver connection diagram	171
15.5 Parameters unit - DM420A	171
15.5.1 Current	171
15.5.2 Stepmode	172
15.5.3 Speed	172
15.5.4 Distance	173
15.5.5 Direction	173
15.5.6 Acceleration	173
15.6 Conversions	174
15.6.1 Distance	174
15.6.2 Speed	174
15.7 Communication protocol	175
15.7.1 Parameter types	175
15.7.2 Example parameter string	176
16 Unified Modelling Language	177
16.1 Document history	177
16.2 Introduction	178
16.3 What is Unified Modelling Language	178
16.4 Use cases	178
16.5 Sequence diagrams	179
16.6 Further reading	180
17 CPS software planning	181
17.1 Document history	181
17.2 Introduction	182
17.3 Initial challenge	182
17.4 First software concept	182
17.5 Second software concept	182
17.6 Third software concept	183
17.6.1 Use Cases	184
17.7 Third software concept architecture	188
17.7.1 Why CPS test station utilises software architecture	188
17.7.2 Choosing the right architecture (Software architecture)	188
17.7.3 Objects/classes interaction (Sequence diagram)	189

18 Detecting frequency	193
18.1 Document history	193
18.2 Introduction	194
18.3 Frequency detection initial thoughts	194
18.3.1 Frequency detection concept - DAQ	194
18.3.2 Frequency detection concept - USB oscilloscope	195
18.3.3 Frequency detection concept - FPGA	196
18.3.4 Frequency detection concept - Counter Integrated Circuit	197
19 Frequency calculation	201
19.1 Document history	201
19.2 Introduction	202
19.3 Explanation of frequency span	202
20 Qt Framework	207
20.1 Document history	207
20.2 Introduction	208
20.3 Qt framework	208
20.4 Qt Integrated Development Environment	208
20.5 Qt Designer	208
20.6 Not using Qt Designer	209
21 Log files and directory structure	211
21.1 Document history	211
21.2 Introduction	212
21.3 Stakeholder requirement	212
21.4 The directory structure	212
21.4.1 Directory structure diagram	212
21.5 Log and report file	213
21.5.1 XML log file	213
21.5.2 Example XML file output	215
21.5.3 PDF report	215
21.5.4 Log and report file name format	216
21.6 Last step	217
21.7 Calibration	217
21.8 Doxygen documentation and code	217
22 CPS graphical user interface	218
22.1 Document history	218
22.2 Introduction	219
22.3 GUI Concepts	219
22.3.1 GUI concept 1	219
22.3.2 GUI concept 2	220
22.3.3 GUI concept 3	221
22.3.4 GUI concept 4	222
22.3.5 Choice of GUI concept	222

22.4	Final graphical user interface	222
22.4.1	File directory	223
22.4.2	Control section	223
22.4.3	Display CPS data section	226
23	Source code	228
23.1	Document history	228
23.2	Introduction	229
23.3	Hardware configuration for developing and testing	229
23.4	Class diagram	230
23.5	Include graphs	230
23.6	Arduino source code flowchart	231
23.7	Signals and slots in Qt	233
24	Conducted tests	235
24.1	Introduction	235
24.2	Test, KDA sensor T-C0-1.0.0	236
24.3	Subtest, inductance final concept Sub-T-2.0.0	240
24.4	Subtest, counter IC concept Sub-T-3.0.0	244
24.5	Subtest, height of rotor impact on inductance Sub-T-4.0.0	248
25	Improvements	253
25.1	Document history	253
25.2	Introduction	254
25.3	General system improvements	254
25.4	Counter circuit improvements	254
25.5	Software improvements	254
25.6	Stepper motor improvement	255
25.7	Oscillator improvements	255
26	Conclusion	256
27	References	257
28	Appendix A - All iterations of risk analysis	262
28.1	Introduction	262
28.2	Risk analysis - 25.01.2019	263
28.2.1	Hardware risks:	263
28.2.2	Human risks:	265
28.2.3	Management risks:	267
28.3	Risk analysis - 27.02.2019	268
28.3.1	Hardware risks:	268
28.3.2	Human risks:	270
28.3.3	Management risks:	272
28.4	Risk analysis - 20.03.2019	273
28.4.1	Hardware risks:	273

28.4.2	Human risks:	275
28.4.3	Management risks:	277
28.5	Risk analysis - 1.05.2019	278
28.5.1	Hardware risks:	278
28.5.2	Human risks:	280
28.5.3	Management risks:	282
29	Appendix B - Working with Qt	283
29.1	Document history	283
29.2	Introduction	284
29.3	Potentiometer values on a graphical user interface	284
29.4	Arduino code to read and send potentiometer value	284
29.5	Potentiometer reading GUI	285
29.6	Working with Qt conclusion	286
30	Appendix C - Directory structure doxygen documentation	287
30.1	Introduction	287
31	Appendix D - Potentiometer reader GUI doxygen documentation	296
31.1	Introduction	296
32	Appendix E - Final software doxygen documentation	321
32.1	Introduction	321

iii List of Figures

1	Subsystem overview	1
2	Project organisation map	6
3	Initial project Gantt chart part 1	9
4	Initial project Gantt chart part 2	10
5	Updated project Gantt chart part 1	11
6	Updated project Gantt chart part 2	12
7	Final project Gantt chart part 1	13
8	final project Gantt chart part 2	14
9	Scrum values [47]	16
10	Colpitts oscillator	58
11	Design process from circuit schematic(left), simulation(middle) to PCB layout(right)	60
12	First Colpitts oscillator	62
13	Output signal of first Colpitts oscillator	62
14	PSpice FFT simulation of first Colpitts oscillator output	63
15	Output signal form breadboard circuit of first Colpitts oscillator	63
16	FFT oscilloscope first Colpitts oscillator	63
17	Common emitter Colpitts oscillator	64
18	FFT simulation common emitter Colpitts oscillator	65
19	3 channel Colpitts oscillator	66
20	PSpice FFT simulation 3 channel oscillator	67
21	Oscilloscope FFT 3 channel Colpitts oscillator	67
22	Oscilloscope FFT 3 channel Colpitts oscillator	68
23	Oscilloscope FFT 3 channel Colpitts oscillator	69
24	Oscilloscope FFT 3 channel Colpitts oscillator	70
25	Oscilloscope FFT 3 channel Colpitts oscillator	70
26	Fifth iteration BJT Clapp oscillator	71
27	Fifth iteration JFET Clapp oscillator variation 1	71
28	Fifth iteration JFET Clapp oscillator 2	71
29	Sixth iteration BJT Clapp oscillator	72
30	Q-point of amplifier in figure 29	73
31	Sixth oscillator iteration input signal on Schmitt trigger	73
32	Common emitter amplifier	75
33	Q-point of amplifier in figure 32, $(V_{CEQ}, I_{CQ}) = (1.9357V, 1.1076mA)$	79
34	Final iteration, 3 channel Clapp oscillator	82
35	PSpice FFT final oscillator	83
36	Oscilloscope FFT of final oscillator output signal.	83
37	Spectrum analysis of signal from final oscillator	84
38	PSpice simulation of output signal of final oscillator	84
39	Plot from oscilloscope of output signal of final oscillator	85
40	Schmitt trigger and voltage divider	86
41	Complete CPS circuit	87
42	Trough hole pin diameter 1.2 mm (left) and Via diameter 0.4 mm (right)	89

43	Coil connection through hole pin	90
44	3D model of stator PCB top side	90
45	3D model of stator PCB top side. Position of the coils marked with white silkscreen line	91
46	Angle of coil placement and spacing between coils. Coil part centre(black cross). Coil connection pin(red square)	92
47	3D coil placement in stator	93
48	Coil design in Orcad PCB Editor	94
49	Coil 1: Single layer spiral coil	94
50	Coil 2: Three layer spiral coil	95
51	Coil 3: Three Layer curved rectangular coil	95
52	Coil 4: 7 Layer curved rectangular coil	96
53	Coil 4 excel input table	96
54	Coil 4 coordinate excel sheet	96
55	Final coil design, layers connected in series with vias	98
56	Angle of copper layer in rotor	99
57	Side view of rotor,	100
58	Top view of rotor	100
59	Copper layers in rotor	100
60	Vias surrounding mounting hole to ensure grounding	101
61	Finished Stator PCB	101
62	Finished rotor PCB	102
63	Bracket, KDA prototype, P.1.0.1.1	106
64	Cap, KDA prototype, P.1.0.2	107
65	Assembly of KDA prototype, A.1.0.0	107
66	3D model of stator, concept 2	109
67	3D model of rotor, concept 2	110
68	Housing top view	111
69	Housing bottom view	112
70	Shaft, concept 2	113
71	Rotor fixture, concept 2	113
72	Assembly of concept 2, A.2.0.0	114
73	Cross section, A.2.0.0	115
74	Assembly of concept 3, A.3.0.0	117
75	Cross section, A.3.0.0	118
76	Top view, A.3.0.0	119
77	Bottom view, A.3.0.0	119
78	Housing concept 3, P-C3-3.0.1	121
79	Rotor fixture concept 3, P-C3-3.0.3	122
80	Shaft concept 3, P-C3-3.0.4	123
81	Shaft top section side view, P-C3-3.0.4	123
82	Fixture for electrical motor, P-C3-3.0.4	124
83	Assembly of concept 4, A.4.0.0	125
84	Cross section, A.4.0.0	126
85	Top view, A.4.0.0	127

86	Bottom view, A.4.0.0	127
87	Housing concept 4, P-C4-4.0.1	129
88	Housing top view, P-C4-4.0.1	130
89	Rotor Mount measures and tolerances, 2D	133
90	Shaft measures and tolerances, 2D	134
91	Housing measures and tolerances, sheet 1, 2D	135
92	Housing measures and tolerances, sheet 2, 2D	136
93	Housing measures and tolerances, sheet 3, 2D	137
94	Housing measures and tolerances, sheet 4, 2D	138
95	Cap measures and tolerances, 2D	139
96	Example, ball bearing [52]	142
97	Radial load, [52]	143
98	Thrust load, [52]	144
99	Angular load, [52]	144
100	Preloaded bearings, forces on balls	149
101	Nanotech SMCI33-2 driver	158
102	NanoPro motor settings tab	159
103	NanoPro input settings tab	160
104	Code version 1 connection diagram KDA driver and motor	162
105	Code version 2 connection diagram KDA driver and arduino	163
106	EM402 stepper driver by Leadshine	167
107	Stepper motor driver connection diagram	171
108	Use case example	178
109	Sequence diagram example	179
110	First software concept	182
111	second software concept	183
112	Third software concept	184
113	CPS software base Use Case	185
114	View CPS data Use Case	186
115	Set parameters Use Case	186
116	Control stepper motor Use Case	187
117	Log CPS data Use Case	187
118	software concept architecture	188
119	Display CPS data sequence diagram	189
120	Write log sequence diagram	190
121	Execute with parameters sequence diagram	190
122	Calibrate sequence diagram	191
123	Frequency detection DAQ concept	195
124	Frequency detection USB oscilloscope concept	195
125	Frequency detection FPGA concept	196
126	Counter integrated circuit concept	197
127	Interface diagram	198
128	Counter integrated circuit [54]	199
129	Circuit diagram integrated circuit diagram	200
130	Stator on rotor	205

131	Frequency depending on rotor position in steps	206
132	Qt Designer example	209
133	Diagram of the directory structure	212
134	GUI concept 1	219
135	GUI Concept 2	220
136	GUI Concept 3	221
137	GUI Concept 4	222
138	Directory view from final GUI	223
139	Control section part 1	224
140	Control section part 2	225
141	Control section part 3	225
142	GUI realtime CPS data	226
143	GUI current position	226
144	Final graphical user interface	227
145	Class diagram final software	230
146	Include graph final software part 1	230
147	Arduino flowchart general	231
148	Arduino flowchart clear counter	232
149	Arduino flowchart snapshot register	232
150	Arduino flowchart print bytes	232
151	Arduino flowchart move steps	233
152	Qt signal and slots part 1	234
153	Qt signal and slots part 2	234
154	Prototype from KDA	239
155	Circuit for setup	242
156	Stator in 3D printed test station	243
157	Rotor in 3D printed test station	243
158	Frequency at 0% covered, 0.6 mm distance	250
159	Inductance with 0% covered, 0.6 mm distance between rotor and stator	250
160	Frequency at 0% covered, 1.4 mm distance	251
161	Inductance with 0% covered, 1.4 mm distance between rotor and stator	251
162	Frequency at 100% covered, 0.6 mm distance	251
163	Inductance with 100% covered, 0.6 mm distance between rotor and stator	252
164	Frequency at 100% covered, 1.4 mm distance	252
165	Inductance with 100% covered, 1.4 mm distance between rotor and stator	252
166	Reading potentiometer on GUI system	284
167	Qt potentiometer reading GUI	286

iv List of Tables

1	Project description document history	2
2	Scrum document history	15
3	Work routines document history	24
4	Risk management document history	27
5	Risk matrix	29
6	Probability and severity levels	29
7	Color code description	29
8	Changes done in risk analysis	30
9	Management risks	30
10	Hardware risks page 1	31
11	Hardware risks page 2	32
12	Human risks page 1	33
13	Human risks page 2	34
14	Management risks	35
15	Stakeholder requirements document history	36
16	Stakeholder requirement template	37
17	Stakeholder requirement specification 1/2	38
18	Stakeholder requirement specification 2/2	39
19	Requirements document history	40
20	Requirement priority	41
21	Verification criteria	41
22	System requirement template	42
23	Functional requirements	43
24	Non-functional requirements	44
25	Non-functional requirements	45
26	Non-functional requirements	46
27	User Story document history	47
28	Example of User Story setup	49
29	Overview of all User Stories and their respective requirements	50
30	Testing document history	51
31	Example test template	53
32	Example subsystem test template	54
33	Sensor design document history	55
34	BJT amplifier components	76
35	Capacitor tolerance amplifier	76
36	Clapp oscillator tank components	80
37	Pin and via dimensions	88
38	Specifications final coil design	98
39	Mechanical concepts document history	103
40	Tolerance symbols and descriptions, [21]	132
41	Specific tolerances and values	132
42	Bearings document history	141
43	k-value and preload forces	150

44	Choice of materials document history	152
45	KDA motor and driver document history	157
46	Overview of the stepmodes available on the SMCI33-2 driver	165
47	SMCI33-2 driver max speed per step resolution	165
48	SMCI33-2 driver max speed in °/s per step resolution	166
49	Other available SMCI33-2 driver parameter range with units	166
50	DM420A driver document history	169
51	Overview of current settings on the DM420A driver	172
52	Overview of stepmodes available on the DM420A driver	172
53	Overview of possible speeds with the DM420A driver	173
54	Distance Overview of possible distances in each stepmode on the DM420A driver	173
55	Parameters to the Arduino Mega 2560	175
56	An example parameter string message to the Arduino Mega 2560	176
57	Explanation of the parameter string given in 56	176
58	Unified Modelling Language document history	177
59	CPS Software planning document history	181
60	Detecting frequency document history	193
61	Frequency calculation document history	201
62	Speed table	203
63	Qt Framework document history	207
64	Log files and directory structure document history	211
65	Explanation of each node in an XML log file	214
66	Overview of libraries used to set up PDF report generation	215
67	CPS graphical user interface document history	218
68	Source code document history	228
69	Hardware configuration	229
70	Test, KDA sensor T-C0-1.0.0	236
71	Test, KDA sensor T-C0-1.0.0	237
72	Test, KDA sensor T-C0-1.0.0	238
73	Subtest, inductance final concept Sub-T-2.0.0	240
74	Subtest, inductance final concept Sub-T-2.0.0	241
75	inductance 2V	242
76	inductance 5V	242
77	Subtest, counter IC concept Sub-T-3.0.0	244
78	Subtest, counter IC concept Sub-T-3.0.0	245
79	Subtest, counter IC concept Sub-T-3.0.0	246
80	Subtest, counter IC concept Sub-T-3.0.0	247
81	Height of rotor impact on inductance Sub-T-4.0.0	248
82	Height of rotor impact on inductance Sub-T-4.0.0	249
83	Frequency depending on height of rotor	250
84	Improvements document history	253
85	Hardware risks page 1 - iteration 1	263
86	Hardware risks page 2 - iteration 1	264
87	Human risks page 1 - iteration 1	265

88	Human risks page 2 - iteration 1	266
89	Management risks - iteration 1	267
90	Hardware risks page 1 - iteration 2	268
91	Hardware risks page 2 - iteration 2	269
92	Human risks page 1 - iteration 2	270
93	Human risks page 2 - iteration 2	271
94	Management risks - iteration 2	272
95	Hardware risks page 1 - iteration 3	273
96	Hardware risks page 2 - iteration 3	274
97	Human risks page 1 - iteration 3	275
98	Human risks page 2 - iteration 3	276
99	Management risks - iteration 3	277
100	Hardware risks page 1 - iteration 4	278
101	Hardware risks page 2 - iteration 4	279
102	Human risks page 1 - iteration 4	280
103	Human risks page 2 - iteration 4	281
104	Management risks - iteration 4	282
105	Learning Qt document history	283

v Abbreviations

Phrase	Description
AC	Alternating current
AR	Anders Rønning
BJT	Bipolar Junction Transistor
CPS	Contactless Positioning Sensor
DAQ	Data acquisition
DC	Direct current
EE	Electronics Explorer
FFT	Fast Fourier Transform
FPGA	Field-programmable gate array
GUI	Graphical User Interface
HFJ	Hans Fredrik Jamtveit
HS	Henrik Sæter
IC	Integrated circuit
IDE	Integrated Development Environment
JFET	Junction gate Field-Effect Transistor
JSS	Jarand Solberg Strømmen
KDA	Kongsberg Defence and Aerospace
KSS	Kongsberg Kongsberg Space & Surveillance
MBC	Magnus Berntsen Caro
MoS ₂	Molybdenum Disulphide
MSVC	Microsoft Visual Studio
PBI	Product Backlog Items
PCB	Printed Circuit Board
SDK	Software Development Kit
STRQ	Stakeholder requirement
UI	User Interface
UML	Unified Modeling Language
USB	Universal Serial Bus
USN	University of South-Eastern
V _{cc}	Voltage common collector
XML	eXtensible markup Language

1 Introduction

This bachelor thesis is initiated by Kongsberg Defence and Aerospace (KDA), University of South-Eastern Norway (USN) and a group of five multidisciplinary students from USN.

A satellite is a moon, planet or machine that orbits a planet or star [34]. Humans use satellites for a vast selection of tasks. They range from providing information about our own planet, to research directed away from the earth. It is expensive to launch satellites into space, thus from the moment they have been launched, long operational time is expected. Space environment is also extremely harsh. There are extreme temperatures, and external environmental conditions such as radiation. Human intervention is something that is rarely done due to the nature of the environment. Therefore, as previously stated, once a satellite is in space it must continue to work for the desired lifespan.

Satellites often use pointing mechanisms to ensure that data is sent, or received in the correct direction, or to direct solar arrays towards the sun. Most of KDA's space applications are equipped with a position sensor to read actual pointing position, however the only solution KDA has integrated is a potentiometer design. This design reads actual position of e.g. solar array by having parts that are continuously in contact. KDA is looking for an alternative to the potentiometer design with extremely high accuracy, high repeatability, and high reliability. To verify this, a test station shall also be developed.

The sensor system shall detect a position on a rotational axis and consist of one stationary part and one rotating part. The sensor detects a change in frequency which correlates to a specific angle. The change in frequency occurs in a LC oscillator. A rotor affects the component value in the stationary LC oscillator by rotating over it without being in contact, thus making it a contactless position sensor.

The entire system consist of four main subsystems depicted in figure 1. Oscillator, test station, counter circuit, and coherent software.

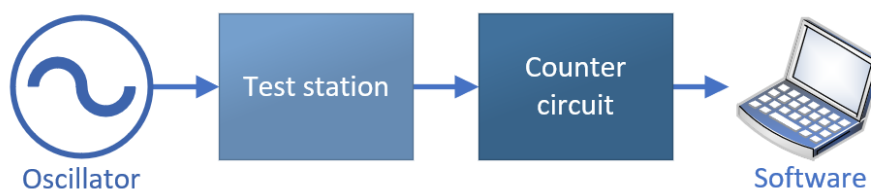


Figure 1: Subsystem overview

The frequency value is generated and changed in the oscillator. The counter circuit consists of a Schmitt trigger and a binary counter. The Schmitt trigger transforms the sinusoidal wave to a square wave which then triggers the counter. The counter deals with square waves better than with sinusoidal waves, and counts the frequency. The test station makes sure the sensor can be tested properly under various conditions. The software controls the stepper motor and makes sure the counted values are stored.

2 Project description

2.1 Document history

Table 1: Project description document history

Version	Date	Author	Description
1.0.0	18.01.2019	MBC	Document created, added abstract, added project background.
1.1.0	21.01.2019	MBC	Added Kongsberg Defence & Aerospace, stakeholders and group information.
1.2.0	22.01.2019	MBC	Added project hierarchy, stakeholder requirements and task summary.
1.2.1	24.01.2019	JSS	Document review.
1.3.0	25.01.2019	MBC, AR	Changed project description document history table, added project goal, corrected error in organisation map.
1.3.1	25.01.2019	AR	Changed abstract to introduction.
1.3.2	24.03.2019	JSS	Added paragraph in project background.
1.3.3	24.03.2019	AR	Fixed a spacing issue.
1.4.0	25.03.2019	MBC	Added updated Gantt chart.
1.5.0	12.05.2019	MBC	Added final Gantt chart.
1.5.1	17.05.2019	MBC	Proofread.
1.5.2	19.05.2019	AR	Changed Kongsberg Defence & Aerospace to KDA.

2.2 Introduction

This section will explain the reason for the task as well as information about the different stakeholders that are involved in the project. This is important to assure stakeholders that the task is understood, as well as assuring that the process is done correctly.

2.3 Project background

The project is based on creating a contactless positioning sensor that can replace today's potentiometer solution. A test station shall be developed to validate the crucial criteria that the CPS need to pass for it to be used in space.

2.4 Kongsberg Defence & Aerospace

Kongsberg Defence & Aerospace (KDA) is a large company with many subsidiaries. Their portfolio consists of surveillance and control of weapon-systems, missiles and communications systems, and systems for command and control. KDA is Norway's premier supplier of defence and aerospace-related systems [27], and in December 2018 KDA acquired a number of contracts worth 805 MNOK [31].

Kongsberg Space & Surveillance (KSS) is a subsidiary that delivers vast selection of equipment related to maritime and space surveillance. This includes components and equipment for the European heavy-lift launcher Ariane 5, scientific probes, surveillance satellites and observation-satellites. KSS is world leading in supplying ground stations for satellites, observation and scientific probes. KSS is also a provider of maritime domain awareness systems and control centers. KSS is Norway's largest supplier of equipment and services to the European Space Agency (ESA) [26].

2.5 Project tasks and goals

Based on already existing solutions that read actual pointing position of different equipment, KDA is interested in a system with an even lower possibility of malfunction. The CPS team has therefore specified primary, and secondary project goal with purpose to meet the requirements.

2.5.1 Primary goals:

- Trough analysis and review, make a functional contactless position sensor.
- Make a test station that can measure and log data about the angle of degree.

2.5.2 Secondary goals

- Test the sensor, and evaluate how it meets the different requirements given, such as operation in extreme thermal conditions, vibrations resistance, how external homogeneous magnetic fields affect the sensor.

2.6 Project Stakeholders

2.6.1 Stakeholder

A stakeholder is anyone with an interest or concern in the project. The stakeholders can affect the project, as well as be affected by it. One can often divide into several categories of stakeholders, and for this project scope there are two main categories. Primary and secondary stakeholders.

2.6.2 Primary stakeholders

The primary stakeholders for the CPS 2019 project are:

- University of Southeast-Norway (USN). USN have their own set of academic requirements for student thesis, such as documentation, literature with scientific standards, presentations and project planning. USN is therefore considered a primary stakeholder for this project.
- KDA. KDA is the main employer of this project and therefore a primary stakeholder
- Contactless Position Sensor team. The CPS team is the group to research and develop the project and therefore a primary stakeholder.
- External censor and advisor. The external censor and external advisor have influence over the product, the process and the grade setting and therefore a primary stakeholder.
- Internal censor and advisor. The internal censor and advisor both have influence over the grading and the process of the project and is therefore considered a primary stakeholder.

2.6.3 Secondary stakeholders

The secondary stakeholders for CPS 2019 project are:

- Vendors and suppliers. Different vendors and suppliers of equipment, material, components have an indirect relation to the project and are therefore considered a secondary stakeholder.
- Customers of KDA. The customers of KDA also have indirect relations to the project and is therefore considered a secondary stakeholder.

2.6.4 Stakeholder requirements

Each primary stakeholder has a set of goals they want to achieve from the CPS 2019 project. While USN focuses more on the process of the project, KDA focuses more on the technical solution.

University of Southeast-Norway:

- Project process

- Project model
- Project documentation
 - Research
 - Requirement Specification
 - Project plan
 - Risk
 - Concept study
 - Design description
 - Test plan
 - Test procedures
 - Test reports
 - Scientific standard
- Project presentations
 - Presentation 1 - introduction to the project
 - Presentation 2 - current progress and remaining tasks
 - Presentation 3 - sale pitch and technical presentation
- Project product

Kongsberg Defence & Aerospace:

- Establish system requirements
- Develop, manufacture and test a contactless position sensor with the necessary hardware, software and documentation
- Develop a test station with a convenient user interface to perform functional testing and calibration of the sensor
- The sensor shall have the same mechanical interface as the position sensor used by KONGSBERG today
- Verification of requirements shall happen through reviews, analysis and testing
- Documentation

2.7 Project hierarchy

The project is to be executed by 5 engineering students from University of South-Norway. They have been assigned one advisor and one censor from KDA. The project team has also been assigned one advisor and one censor from the University of Southeast-Norway.

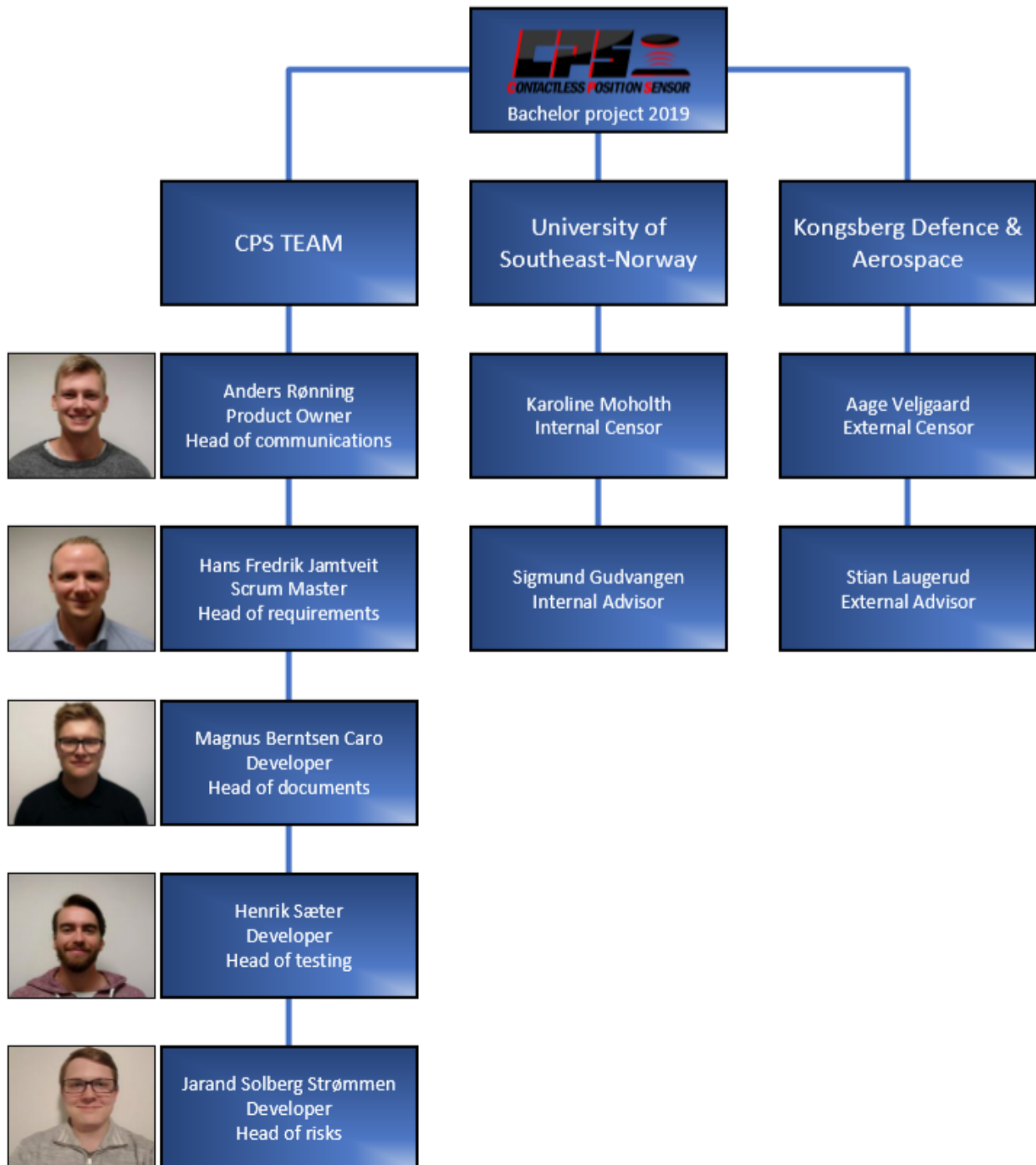


Figure 2: Project organisation map

2.7.1 Contactless Position Sensor project team

The engineering students come from three different disciplines. There are two electrical engineering students, two computer engineering students, and one mechanical engineering student. They are all in their last year of their bachelors degree and this project is their main thesis.

Anders Rønning

Electrical engineering
Product Owner

Phone: +47 452 52 178

Email: anders.ronning94@gmail.com

Hans Fredrik Jamtveit

Electrical engineering
Scrum Master

Phone: +47 924 30 598

Email: hans_fredrik_jam@hotmail.com

Magnus Berntsen Caro

Computer engineering
Developer

Phone: +47 480 23 353

Email: magnus.caro@outlook.com

Jarand Solberg Strømmen

Computer engineering
Developer

Phone: +47 934 62 615

Email: jarand_ss@hotmail.com

Henrik Sæter

Mechanical engineering
Developer

Phone: +47 954 69 340

Email: henrik.saeter@hotmail.com

2.7.2 University of Southeast-Norway

The University of Southeast-Norway have provided the project with a censor and an advisor.

The internal censor task is to censor all of the bachelor projects. The internal censor is also to take part in a panel of 3 individuals that decide the individual grade of the project team members. The internal censor must be present at all three presentations by the project group.

The internal advisor task is to guide and support the students during the execution of the project. The internal advisor is also a part of a panel of three persons who decides the individual grade of the project team members. The internal advisor must be present at all three presentations by the group.

Karoline Moholth

Internal censor

Phone: +47 310 08 898

Email: Karoline.Moholth@usn.no

Sigmund Gudvangen

Internal advisor

Phone: +47 310 08 905

Email: Sigmund.Gudvangen@usn.no

2.7.3 Kongsberg Defence & Aerospace

KDA shall provide the project team with an external censor and an external advisor.

The external censor task is to censor the bachelor project, and take part in a panel of three persons that decide the individual grade of the project team members. The external censor must be present at all three presentations by the project team.

The external advisor tasks are split into two parts. Firstly, the external advisor represent the client and is therefore a customer. Secondly the external advisor is responsible to make sure that the necessary resources from KDA are available for the project group. This includes equipment/software, and information/guidance.

Aage Veljgaard

External censor

Phone: +47 975 74 979

Email: aage.soerensen@kongsberg.com

Stian Laugerud

External advisor

Phone: +47 472 38 500

Email: stian.laugerud@kongsberg.com

2.8 Project timeline

The project timeline starts at the first of January 2019 and ends when the third presentation is finished tentatively mid June. The team has created an estimate of the sprints from start to finish. Sprints are explained in section 3.6.1. This estimate will most likely change as problems arises and experience is gained. The points at the bottom of the figure shows a preliminary estimate of important project aspects. This chart will change as the team focuses on having an agile project. The end of the project is when the grade is set on the June 14.

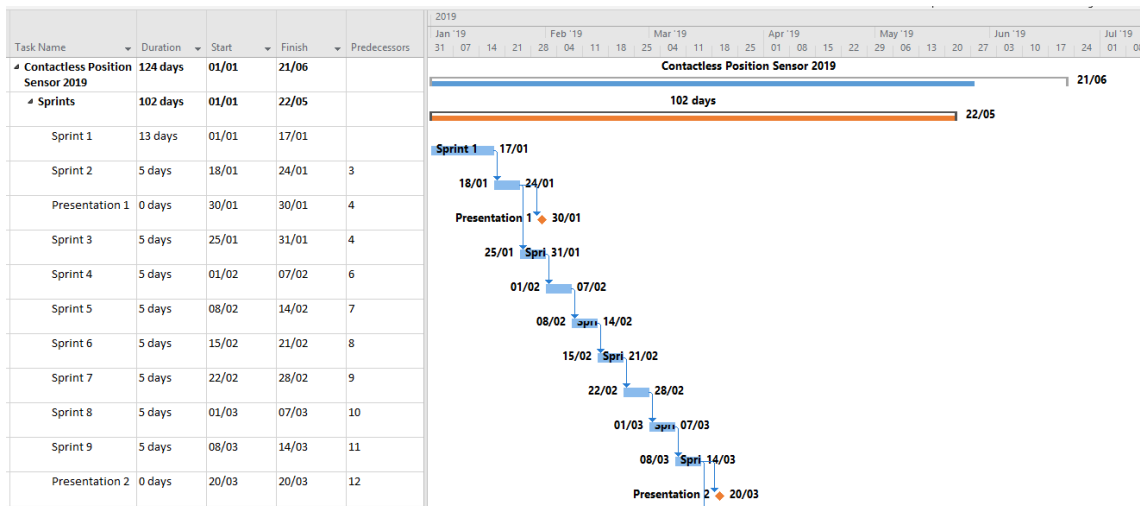


Figure 3: Initial project Gantt chart part 1

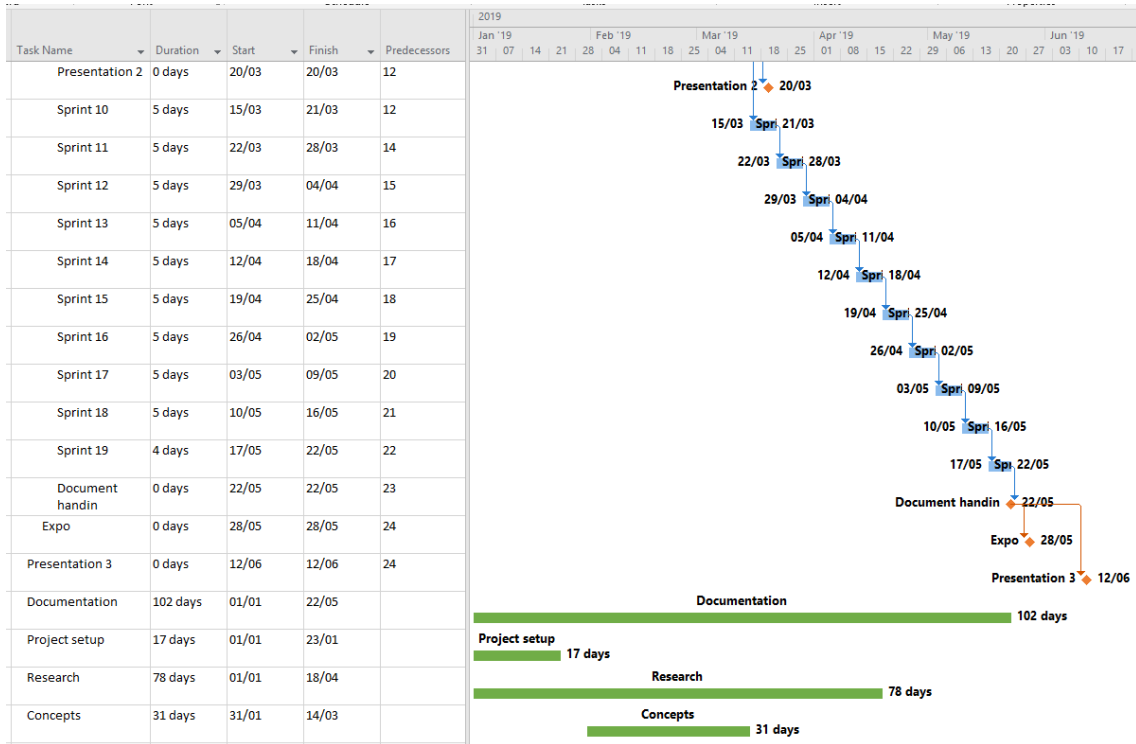


Figure 4: Initial project Gantt chart part 2

2.9 Updated project timeline

As mentioned in section 2.8 the initial Gantt chart was prone to changes. The Gantt chart has been updated and is illustrated in figure 5 and figure 6 During Sprint 6 the CPS team discovered that the Scrum Events Sprint Review and Sprint Retrospective was more time consuming than rewarding so the team decided to prolong the Sprints by one week. This meaning that the Sprints would be 2 weeks until the middle of Easter whereas the Sprints will go back to one-week Sprints. Other changes are minor ones such as the date for presentation 2 and the end of the project which is now set to the date of the last presentation. The date for the last presentation will also change one last time whereas that date is set by USN.

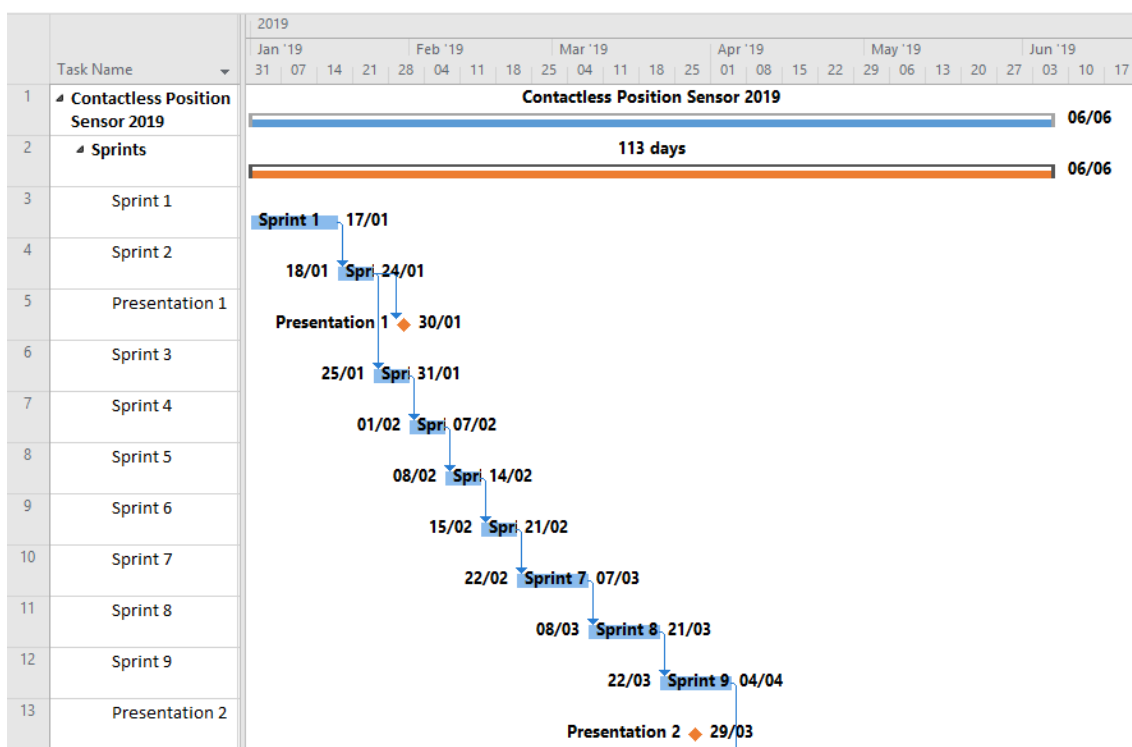


Figure 5: Updated project Gantt chart part 1

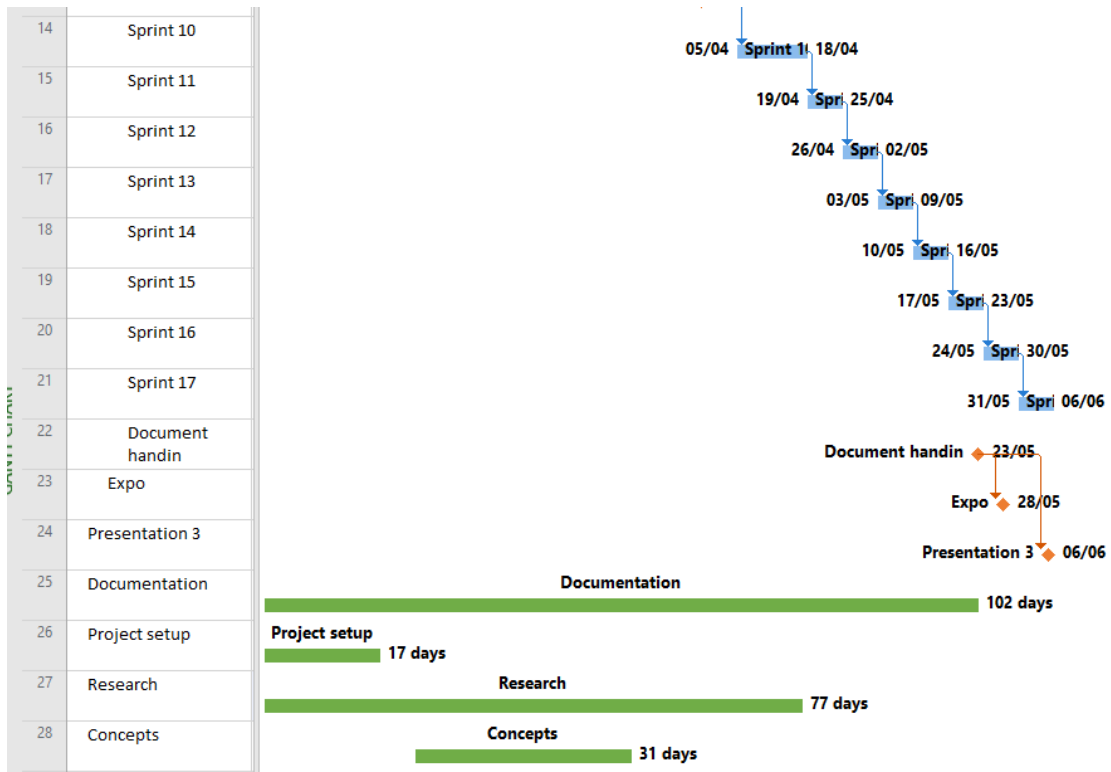


Figure 6: Updated project Gantt chart part 2

2.10 Final project timeline

The final project timeline is illustrated in figure 7 and figure 8. After 22.04.19, the sprints were to be shortened to down to one week each. This however was not done as the CPS team was comfortable with the achievements and the dynamic of having two week sprints.

New milestones have been added and they are; The date the CPS team received the printed circuits board ordered, and the mechanical test station. This allowed for testing and drove the project further towards its end goal.

Overall, the project plan was predicted fairly precise, except from minor change. These changes were presentation dates changing, and prolonging of sprints. Predicting this accurate was possible due to the agile approach, enabled by using Scrum as a project model. The CPS team was able to adapt the Scrum methodology in a way that benefited the team.

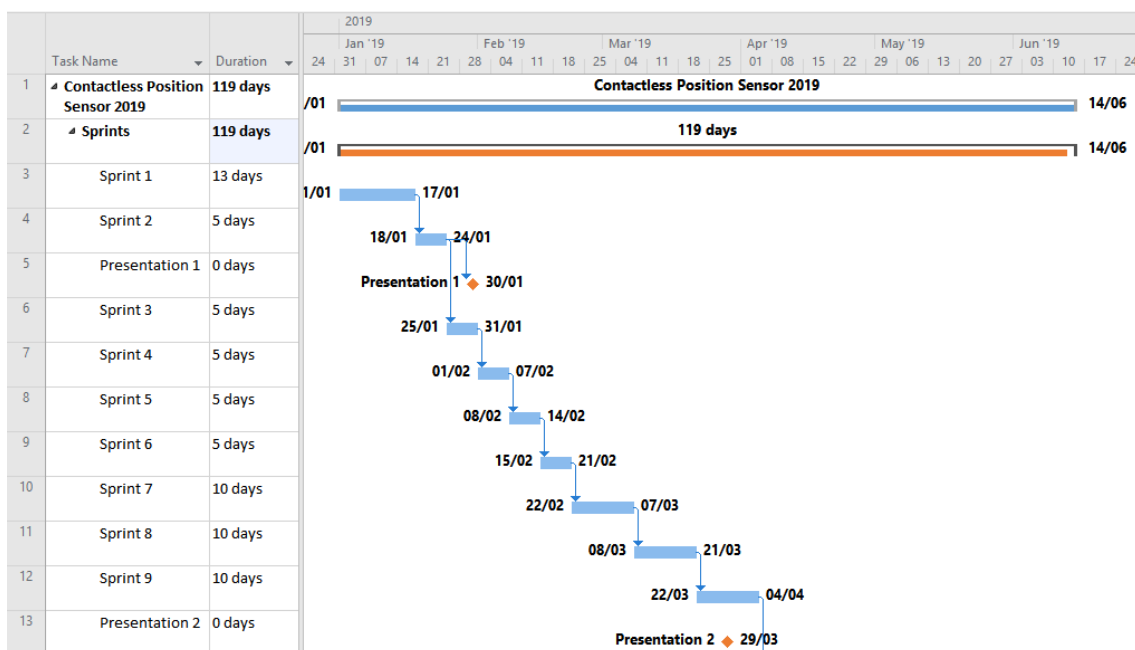


Figure 7: Final project Gantt chart part 1

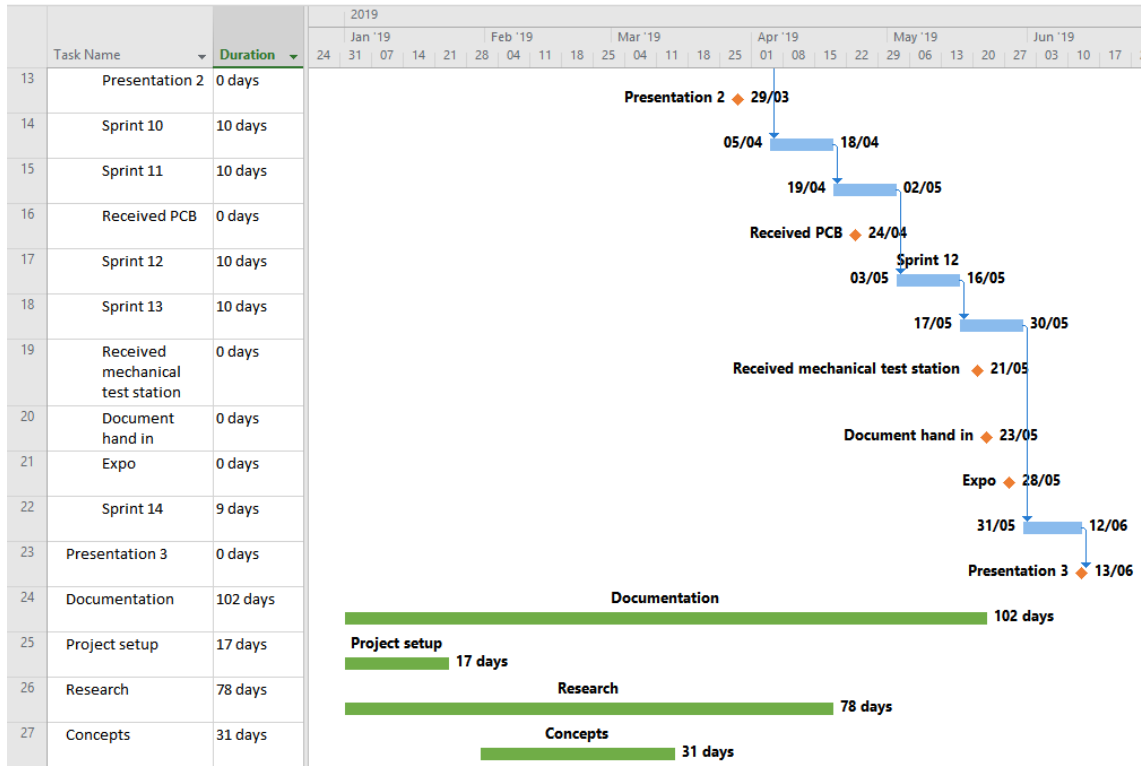


Figure 8: final project Gantt chart part 2

3 Scrum

3.1 Document history

Table 2: Scrum document history

Version	Date	Author	Description
1.0.1	18.01.2019	HFJ	Added Scrum.
1.1.1	23.01.2019	HS	Document reviewed.
1.1.2	25.01.2019	AR	Changed abstract to introduction.
1.1.3	17.05.2019	MBC	Proofread.

3.2 Introduction

This section will explain the different Scrum values, roles, artefacts, events, and the approach to Scrum from the CPS team.

3.3 Scrum values

The Scrum values for the CPS team will aid to collaborate efficiently and achieve a common goal. A proficient use of these values will help the team grow as engineers. [63]



Figure 9: Scrum values [47]

3.3.1 Courage

The CPS team members will do the right thing and try to accomplish tasks that are challenging, even if they are uncertain on how to proceed. Courage also involves asking for input or help from other members of the CPS team, as well as from the advisors.

3.3.2 Focus

The CPS team members will focus on the tasks at hand in each Sprint to not waste time on unnecessary discussions and events. By using strict timeboxes that Scrum is based on, this will be achieved.

3.3.3 Commitment

The CPS team members will commit to doing their best to be able to get the best possible outcome of the project. Commitment to the tasks set in the Sprint Backlog will be important to get a stable process in the project.

3.3.4 Respect

The CPS team members will respect each other as their own equal to avoid conflict within the team. Respecting the stakeholders is a given, but all the more important, for the CPS team.

3.3.5 Openness

The CPS team will be open on all documentation and actions along the project so that stakeholders will have a complete view of how the project evolves and also the challenges along the way. This will add to the transparency of the project as well as trust in the CPS team.

3.4 Scrum roles

Opposed to other project models Scrum only has a limited set of roles.

3.4.1 Scrum Master

The Scrum Master role in Scrum is to ensure and/or enforce a correct Scrum methodology. This role also makes sure that everyone in the team understand Scrum theory and practices it.

The Scrum Master takes part in the Sprint Retrospective Meetings to make sure everything is done in the predetermined timebox, as well as making sure that every member of the team is staying on topic. The Scrum Master is not supposed to take part in this event himself, but only make sure Scrum theory is followed. This is however something that every Scrum team needs to figure out what is best for themselves.

3.4.2 Product Owner

The Product Owner is responsible to make sure the Developer Team can maximise their efficiency so that work can move on. This is achieved through the Product Backlog. The Product Owner is responsible to make backlog items clear and described to the level needed for the team to understand it.

The Product Owner is also responsible for ordering all equipment so the team can continue to work and not be hindered by lack of equipment in the project.

3.4.3 Developer Team

The Developer Team consist of people who are working on the tasks on the Sprint board. It is their duty to always have an assigned task if there are incomplete tasks in the Sprint Backlog. This requires that the team is self-driven.

3.5 Scrum Artefacts

Scrum has something called artefacts and these are the Product Backlog, Sprint Backlog and Increment. They make the key information, regarding the value or work, transparent to the Scrum team. Having Scrum artefacts the team gains a common understanding of the work, values and goals.

3.5.1 Product Backlog

Product Backlog is a list that contain every requirement, function and ability the product shall have. Early in the project the Product Backlog might not contain more than the upper level requirements, but as the project progresses the Product Backlog increases in size as requirements are broken down to smaller requirements.

Items in the Product Backlog is called Product Backlog Items (PBI's). Requirements, scenarios regarding the product and the market is likely to change during the project. This will make changes in the Product Backlog necessary. PBIs often start of as high-level requirements displayed as User Stories. As the project progresses the Product Owner cooperate with the Development Team in refining the Backlog and breaking down the PBIs to more manageable size items. The Product Owner is the sole person responsible for keeping the Product Backlog in good condition. This is done by prioritising the items, refining existing items and adding new ones. [36]

3.5.2 Sprint Backlog

Sprint Backlog is an Artefact that is created at the start of every Sprint. The Scrum Team conducts the Sprint Planning, the product of this event is the Sprint Backlog. The Sprint Backlog consist of PBIs that are broken down to functional tasks by the Development Team. The Sprint Backlog is constructed to accomplish the Sprint Goals. Sprint Goals are defined by the top PBIs. As the Development Team works through the Sprint Backlog and completes task, they may discover that some task is unnecessary or that they have to add task to the Sprint Backlog to reach the Sprint Goals. Changes in the Sprint Backlog can only be done by the Development Team. [36]

3.5.3 Increment

An Increment is a product of the Scrum Team's work in the Sprint. The Increment must be something that is testable, usable, and meets the team's definition of Done. It is ready to be set into production, and is a small steps towards the vision and goal of Scrum Team.

3.6 Scrum Events

The Sprint is the main event in Scrum. All the other events are subevents of the Sprint. The purpose of the Scrum Events is to inspect and adapt the different aspects of the Scrum Team's working process. Scrum Events are designed to increase the value of meetings throughout the process. Time boxes are used on every event to ensure that the team uses

the events effectively. Inspection and transparency are fundamental in Scrum projects, the events make the Scrum team able to be transparent and inspect their work and process and make the necessary adaptations. Lack of implementation of any of the Scrum Events effect the transparency and the teams ability to inspect and adapt their process. [36]

3.6.1 Sprint

A Sprint is a timeboxed iterative event where the team have Scrum Events and do development work. Each Sprint contain the Sprint Planning, Daily Scrum, the Development work, Sprint Review and the Sprint Retrospective. The timebox for a sprint should not exceed one calendar month. The goal of a Sprint is to have a Done, potentially releasable product Increment coherent with the Sprint Goal set by the Product Owner and Development Team in the Sprint Planning Event. During the Sprint there should not be made any changes in the Sprint Backlog that would decrease the ability of the Development Team to reach the Sprint Goal. [36]

3.6.2 Sprint Planning

The Sprint Planning is a Scrum Event that that is led by the Product Owner and facilitated by the Scrum Master but the whole Scrum Team participate in the event. The Product Owner and the Development Team negotiate the Sprint Goals that should be the product Increment of the Sprint. The Sprint Planning can be divided into two parts. The first half of the event should be used by the Development Team to decide which Product Backlog Items that should be included in the Sprint Backlog to ensure that the Sprint Goals is accomplished. The PBIs are already prioritised during the Backlog Refinement. The PBIs are often formulated as user stories. Because of this the second half of the Sprint Planning should be used by the Development Team to break down the PBIs into tasks that have to be accomplished to meet the Sprint Goals. [36]

3.6.3 Daily Scrum

Daily Scrum is a stand-up meeting. This is a 15-minute timeboxed event to create a plan for the next 24-hours. This is done by inspecting the work done since the last Daily Scrum, preferably held in the same location and at the same time each day. Time of day is ideally in the morning as this helps setting an image of the day's coming work. Only the Development Team members should attend the Daily Scrum meeting, as they are the ones who own the Sprint Backlog and are responsible to organise the work independently. Each member of the team will explain these points:

- What did I do yesterday that helped the Development Team meet the Sprint Goal?
- What will I do today to help the Development Team meet the Sprint Goal?
- Do I see any impediment that prevent me or the development team from meeting the Sprint Goal?

When the Daily Scrum has ended, the Development Team gathers and creates a detailed plan for the rest of the day using the information shared in the Daily Scrum. The main

purpose of the Daily Scrum is to optimise the collaboration in the Development Team and increase the probability of meeting the Sprint Goal. [36]

3.6.4 Sprint Review

A Sprint Review is held at the end of each Sprint. The Scrum Master makes sure everyone included in the meeting understands the purpose and keeps within the time-box. The time varies based on the Sprint length. The Increment is reviewed, and the Product Backlog may be adapted. During the review the Scrum team and the key stakeholders collaborate on what was done in the Sprint. During the Sprint Review the Scrum team has the opportunity to get an honest feedback on the Increment of their product from the stakeholders. Transparency is key in this meeting. The stakeholders should get the truth of how the team is progressing and if the team is producing any business value. This is essential in terms of getting feedback and the Scrum team can use it in order to inspect and adapt their product Increment. This is done in order to meet the expectation of the stakeholder. This is not a formal meeting or a status meeting. [49] [20]

Elements included in the Sprint Review:

- Attendees include the Scrum Team and key stakeholders invited by the Product Owner;
- The Product Owner explains what Product Backlog items have been “Done” and what has not been “Done”;
- The Development Team discusses what went well during the Sprint, what problems it ran into, and how those problems were solved, or not solved;
- The Development Team demonstrates the work that it has “Done” and answers questions about the Increment;
- The Product Owner discusses the Product Backlog as it stands. He or she projects likely target and delivery dates based on progress to date (if needed);
- The entire group collaborates on what to do next, so that the Sprint Review provides valuable input to subsequent Sprint Planning;
- Review of how the marketplace or potential use of the product might have changed what is the most valuable thing to do next;
- Review of the time line, budget, potential capabilities, and marketplace for the next anticipated releases of functionality and capability of the product.

3.6.5 Sprint Retrospective

Sprint Retrospective is a Scrum Event where the Scrum Team inspects itself. Every member of the Scrum Team should participate in this event. The Scrum Master encourages the team to make improvements to the development process. Such as making the process more efficient and increase the quality by enhancing the work process. The team should

construct simple and specific procedures that should be adapted in order to achieve the desired work process. The Sprint Retrospective gives every member of the Development Team the possibility to give feedback on what prevents the team from working at its full potential, and how to continue doing the things that make the team prosper. The event takes place after the Sprint Review and prior to the next Sprint Planning.

A typical model for the Sprint Retrospective should include [48]:

- What went well in the Sprint
- What could be improved
- What will we commit to improve the next Sprint?

3.7 Definition of Done

In Scrum an important keyword is Definition of Done. The Definition of Done is a product that can potentially be sent to the customer. With “potentially product” it means something that can give value to the company, but not a 100% complete product. [35]

The Definition of Done is something every Scrum Team can in some degree change and fit to their team and their goal. [37]

3.8 CPS approach to Scrum

3.8.1 Sprint

For the CPS team the duration of the Sprint has been set to one to two weeks due to the time span of the entire project. One week in the early part of the project with the ability to extend the Sprint duration to two weeks when project becomes more technical. The CPS team starts off a Sprint on Fridays. In time for the Sprint Planning event the CPS team has Refined the Product Backlog. From Friday to Thursday the team do development work. The end of Thursday is used for Sprint Review and Sprint Retrospective.

3.8.2 Sprint Planning

Due to the setup of the CPS team both the Product manager and the Scrum Master will participate as a Development Team member during the Sprint Planning.

3.8.3 Daily Scrum

During the CPS team’s Daily Scrum meeting the Scrum Master and Product Owner will also participate because they also hold the roles as Development Team members. The Scrum Master will ensure the time-box is kept and that the structure of the event is according to the Scrum template.

3.8.4 Sprint Retrospective

The CPS team conduct the Sprint Retrospective as the last event of the Sprint. The Scrum Master organizes the event. The centre of the event is a whiteboard that is divided into 3 sections, +, -, and ?. The team writes comments of sticky notes and places the notes in one of the 3 sections. Plus section is for comments on what the team is good at. Minus section is for comments on issues the team have and that they should improve. Question mark section is for solutions to how the team can improve.

3.8.5 Definition of Done

In the CPS team the Definition of Done is not a potentially shippable product, but rather an Increment towards the end goal and vision of the project. This is something that gives value to the team's end result.

Before something is consider Done it needs to be proofread by another team member. This is an important step to minimize the risk for needing to redo a task. This also helps the team to make a unified end document.

3.8.6 Online tool

The CPS team decided to use an online tool to help with organising Scrum. The decision of an online tool was done early in Sprint 1. The process of picking the right online tool for the team was to look into different sites that all advertised as a "Agile and Scrum tool".

The decision ended on Asana. The reason behind the choice was positive reviews and that it seemed intuitive to use. Asana is helping the CPS team to make an online backlog and a place to leave daily reminders and notes for the team members.

If done correctly Asana is also making a time line over all task that is marked as Done and sets a time stamp on it. Asana will automatically log every action done with every task so that the team knows who created the task and when it was last edited. This can be used by the team to make an Increment of all progression after every Sprint.

3.8.7 Roles

Since this is a bachelor thesis and we cannot have daily meetings with the KDA, one in the team is needed to take the role as Product Owner. This is not standard procedure. It is more common to have one person from the company to be the Product Owner.

The team only consists of 5 people and one aspect of the grading is how the team works during the project process. The team has therefore decided to make some small adjustments to the Scrum approach. The Scrum Master and Product Owner will be a part of the Scrum Team. Instead of the Scrum Master making sure things were done correctly on Retrospective and Daily Scrum meeting. The Scrum Master also needs to take part as a Developer under those meetings.

This was done so everyone on the CPS team had a voice on how they feel things are. This is important for the group so everyone can express their opinion. This ties in with the Scrum values explained earlier.

4 Work routines

4.1 Document history

Table 3: Work routines document history

Version	Date	Author	Description
1.0.0	15.01.2019	MBC	Document created.
1.1.0	16.01.2019	MBC	Document fix.
1.1.1	24.01.2019	JSS	Document review.
1.1.2	25.01.2019	AR	Changed abstract to introduction.
1.1.3	17.05.2019	MBC	Proofread.

4.2 Introduction

This chapter will explain the work routines agreed upon by the team. The document will work as a guideline for each team member and a quality assurance for the team.

4.3 Asana

The team will use the project management tool Asana. This is an Agile project management tool that will aid the team in maintaining a correct framework for this project.

4.3.1 Asana routines

1. Asana will be the main source for the project management.
2. During the sprint planning the team shall enter/edit the product backlog in Asana.
3. All tasks shall be entered in Asana and given a weight and importance.
4. Once a task has been taken by a team member, the start and due date for this task shall be entered.
5. Once a task is finished it shall be moved to the review section.
6. A team member can only work on a maximum of 2 tasks at the time.
7. A team member cannot review their own work.

4.4 General work

These are the guidelines for taking breaks and working during the project.

4.4.1 Work hours and breaks

1. The work hours starts at an agreed upon time. This time is 09:00.
2. The team member administrates their own breaks. These breaks shall be fair and if they exceed 15 minutes for regular breaks, or 30 minutes for lunch breaks, they shall be deducted from the hour tracking.
3. Each team member shall track his work hours at the end of each day. This tracking shall be fair, and the team member shall not lie about the hours worked.
4. Work breaks and lunch is included as work hours. As long as they do not exceed the agreed upon amount of time, if they do they shall be deducted from the hour tracking.

4.5 Meetings and presentation

There are guidelines for meeting with external client and internal advisor.

4.5.1 Meetings

1. Each team member shall be dressed appropriately for meetings with both the external client and the internal advisor.
2. Before each meeting a formal invitation shall be sent from the team to the participants at least 2 workdays ahead of the meeting.
3. For each meeting, a team member shall be a notary.
4. After each meeting a formal meeting abstract shall be sent to all participants within 24 hours of the meeting.

4.5.2 Presentations

1. Each team member shall be dressed appropriately for presentations.
2. Each team member shall be prepared and show up at an appropriate time before presentations.
3. The current documentation shall be sent to internal sensor, internal advisor, external sensor and external advisor at least 2 workdays before the presentation.

5 Risk management

5.1 Document history

Table 4: Risk management document history

Version	Date	Author	Description
1.0.0	18.01.2019	JSS	Document created.
1.1.0	18.01.2019	JSS	First review.
1.2.0	20.01.2019	JSS	Formatting, wrote introduction and abstract.
1.3.0	21.01.2019	JSS	Second review, fixed risk analysis points, wrote risk matrix
1.3.2	25.01.2019	AR	Changed abstract to introduction.
2.0.0	20.03.2019	JSS	Changed to risk analysis iteration 3.
3.0.0	09.05.2019	JSS	Changed to risk analysis iteration 4.
3.0.1	17.05.2019	MBC	Proofread.

5.2 Introduction

This chapter contains the iterations of risk analysis for the CPS team and our mitigation plan. The risks have been re-evaluated throughout the project.

Risk management is important to foresee possible complications to the project. By setting up a detailed risk analysis the CPS team will be able to respond quickly and reduce the risk of impeding the progress of the project.

The CPS team chose to include risk management as a measurement to increase the possibility to accomplish the project within the time span. Scrum as an agile management model does not have a set meaning on the use of risk analysis. Effectively under Sprint planning and Sprint review the CPS team will assess the backlog items and give them a weight where the risk of not accomplishing the task is considered. This will help the CPS team mitigate the risks as the project progresses. The risk analysis will be re-evaluated throughout the project to track change in risks, as well as to add in additional ones that are discovered.

5.3 Risk matrix

The risk matrix is used to calculate the level of risks. Probability is the chance of occurrence, and severance is the impact it has on the project. The risk level is the multiplication of the probability and the severity, and it signifies the total impact of a risk on the project.

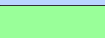
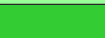




Table 5: Risk matrix

		Severity				
		1	2	3	4	5
P r o b a b i l i t y	1	1	2	3	4	5
	2	2	4	6	8	10
	3	3	6	9	12	15
	4	4	8	12	16	20
	5	5	10	15	20	25

Table 6: Probability and severity levels

Level	Probability	Chance of occurrence	Level	Severity
5	Almost certain	80 - 100%	5	Critical
4	Very likely	60 - 80%	4	Severe
3	Likely	40 - 60%	3	Moderate
2	Highly unlikely	20 - 40%	2	Minor
1	Not likely	0 - 20%	1	Minimal

Table 7: Color code description

Color	Description
	Minimal risk
	Low risk
	Moderate risk
	High risk
	Very high risk
	Critical risk

5.4 Changes in risk analysis

This section will list the changes done to the risk analysis between the period of the third iteration(20.03.2019) and the fourth iteration(1.05.2019). All iterations can be found in section 28

Table 8: Changes done in risk analysis

Risk ID	Column changed	Type of change
R1.1.2	Mitigating action done	Added
R1.2.1	Severity	Updated
R1.2.3	Probability, Mitigating action done	Updated, Added
R1.3.1.2	Mitigating action done	Added
R1.3.1.4	Mitigating action done	Added
R1.3.2	Mitigating action done	Added
R2.1.1	Probability, Severity	Updated
R2.3.1	Mitigating action done	Added
R2.3.2	Mitigating action done	Added
R2.4	Probability, Severity	Updated
R2.5	Mitigating action done	Added
R3.1.2	All	New risk added
R3.1.3	ID, Risk, Description	Changed, fixed typo
R3.2	Severity	Updated
R3.3	Probability, Severity	Updated
R3.4	Probability, Severity	Updated

5.5 Comparison of iterations

Here is the changes in risk level within the three different categories of risks:

Table 9: Management risks

ID	Risk	Iteration 1	Iteration 2	Iteration 3	Iteration 4
R1	Hardware faults	6	5,4	5,7	5,0
R2	Human risks	8,1	7,7	7,6	7,1
R3	Management risks	10,5	11,3	10,3	13,1

5.6 Risk analysis iteration 4

5.6.1 Hardware risks:

Table 10: Hardware risks page 1

ID	Risk	Description	Root cause	Mitigating actions done	Mitigating action	Probability	Severity	Risk level
R1	Hardware faults					2,4	2,0	5,0
R1.1	Sensor failure					3,0	1,5	4,5
R1.1.1	External magnetic field	If the sensor starts to show incorrect data due to external magnetic field	Other electronic components near the sensor could affect the readings of the sensor	Taken action to properly ground the PCB	Take EMC into consideration under the construction phase	5,0	2,0	10,0
R1.1.2	Cracked or bent PCB	If the PCB becomes cracked or bent leaving it unusable	Structural damage due to the intensive testing as well as mishandling it	Multiple PCBs has been ordered and delivered	Have several backup PCBs. Correctly assemble the PCB to the test station	1,0	1,0	1,0
R1.2	Sensor limitation					1,7	2,0	3,3
R1.2.1	Weight capacity	If the weight of the sensor becomes problematic	Under testing the weight of the components for the sensor can be damaging if it is too heavy. The more weight added, the more stress for the connections		Research on material to use material with low density	2,0	1,0	2,0
R1.2.2	Continuous rotation	Problems that could arise from the required continuous rotation of the sensor	The wires to and from the sensor could tangle if handled improperly as well as the overall precision of the sensor could degrade	This will not happen due to choices in design	Run tests on the prototype borrowed from KDA	1,0	3,0	3,0
R1.2.3	Scale sensor to mechanical interface	Problems that could arise when scaling the sensor to fit the standard mechanical interface KDA use today	The sensor may become more unstable due to noise	Proper research, testing has been done. The distance between coils on stator helps to eliminate noise	Analyse the sensor after scaling to make sure it still is viable. Choose frequency range less affected by external noise	2,0	2,0	4,0

Table 11: Hardware risks page 2

R1.3	Test environment					2,6	2,6	6,9
R1.3.1	Extreme temperature					3,3	3,3	10,6
R1.3.1.1	Low temperatures - Sensor data	Problems that could arise from testing the sensor under cold temperatures	Under very low temperatures the electronics may become unreliable		Research into available material that can withstand low temperatures. Research effect of low temperatures on coils	4,0	3,0	12,0
R1.3.1.2	Low temperatures - Fractured PCB	Fracured PCB under low temperature due to shrinking	Materials shrinks when decreasing the temperature	Verification from external advisor that current PCB material is suitable for low temperature	Research into available material that can withstand low temperatures. Research effect of low temperatures on PCB	2,0	4,0	8,0
R1.3.1.3	High temperatures - Sensor data	Problems that could arise from testing the sensor under high temperatures	Under very high temperatures the electronics may become unreliable		Research into available material that can withstand high temperatures	3,0	3,0	9,0
R1.3.1.4	High temperatures - Defect components	Components failing due to high temperatures	Wires and components could become defective under high temperatures	Choice in material in the PCB is suitable for high temperatures	Shielding components and material choice	2,0	3,0	6,0
R1.3.2	Vibration	Problems that could arise from testing the sensor under severe vibrations	Under vibration testing the different layers in the sensor could start hitting each other	The stiffness of the PCB and how it is fixed helps to avoid this	Fasten components firmly and research into material that can withstand the vibration	2,0	2,0	4,0

5.6.2 Human risks:

Table 12: Human risks page 1

R2	Human risks					1,5	4,6	7,1
R2.1	Absence					1,7	3,0	5,0
R2.1.1	Absence within the CPS team	If a member of the CPS team is absent	The members can fall ill or have other events happening in their life that require them to be elsewhere		Early notification to the rest of the team so precautions can be made for their absence	3,0	5,0	15,0
R2.1.2	Absent internal censor	If the internal censor becomes unavailable to attend an activity	The internal censor has other responsibilities that may take priority		Mutiple possible dates for the activity to better fit their schedule	1,0	1,0	1,0
R2.1.3	Absent external censor	If the external censor becomes unavailable to attend an activity	The external censor has other responsibilities that may take priority	Early invitaions have been sent to presentation dates and multiple possible dates available	Mutiple possible dates for the activity to better fit their schedule	1,0	3,0	3,0
R2.2	Data unavailable	If some data is not available to the whole team at crucial points	Some may have data local on their own computer when it needs to be integrated to the main document	The CPS team has frequently uploaded data to a cloud server. Backup downloaded weekly.	All documentation is to be available online even when it's being written	1,0	5,0	5,0
R2.3	Damaged equipment					1,0	5,0	5,0
R2.3.1	Borrowed equipment	If accidents happens where some of the borrowed equipment is damaged	Accidents can happend where the borrowed equipment is damaged due to improper handling or carelessness	Read data sheet and taken precaution to avoid possible damaging scenarios for components	Do not test anything before the proper procedures are known and can be used	1,0	5,0	5,0
R2.3.2	CPS teams equipment	If accidents happens where some of the CPS teams equipment is damaged	Accidents can happend where the personal equipment is damaged due to improper handling or carelessness	Read data sheet and taken precaution to avoid possible damaging scenarios for components	Do not test anything before the proper procedures are known and can be used	1,0	5,0	5,0

Table 13: Human risks page 2

R2.4	New risks	If new risks are discovered that previously was unplanned for	There are many points of views to consider when discovering risks and it is as good as impossible to count for all aspects of a project		Ask the team about it. Make it a point in the next sprint or add to current one if it is vital	3,0	5,0	15,0
R2.5	Plagiarism	If a member of the CPS team copies others work and claims it as their own	If the CPS team don't use references in a proper way so some of the documentation becomes plagiarism	Proper use of references from sources when gathering information	Ask internal adviser if the references is proper and all CPS team member can question content if it seems off	1,0	5,0	5,0

5.6.3 Management risks:

Table 14: Management risks

R3	Management risks					2,8	4,8	13,1
R3.1	Time management					3,0	4,0	12,0
R3.1.1	Documents ready for turn in	If the CPS team does not meet time limits to turn in documentations	Events can happend that takes time to solve properly		Follow scrum and do the highest priority task first	1,0	5,0	5,0
R3.1.2	Document completed	Everything important added and ready for turn in	Not enough time	Internal deadline one week before the proper deadline		3,0	4,0	12,0
R3.1.3	Not meeting all requirements	If CPS team is not capable to fulfill the stakeholder requirements before the project end	Challenges appear along the way and other events can happend to delay the project to the point where meeting all the requirements is not possible		Strict structural approach throughout the project	5,0	3,0	15,0
R3.2	Cost	If unexpected costs appear	If the CPS team needs to buy in additional equipment or components to be able to continue the project		Overestimate what equipment cost. Consult the group before items are bought	2,0	5,0	10,0
R3.3	Delivery	If deliveries of components take longer than anticipated	Deliveries that requires shipping can be uncertain if it will arrive in time		Order as soon as possible and look for backup parts	5,0	5,0	25,0
R3.4	New requirements	If the stakeholder arrives with new requirements	New requirements can be introduced all along the project timeline and require rework of the design	Consulted with employer that new requirements are unlikely to happen	Make it a point in the next sprint or add to current one if vital	1,0	5,0	5,0

6 Stakeholder requirements

6.1 Document history

Table 15: Stakeholder requirements document history

Version	Date	Author	Description
1.0.0	22.01.2019	HS	Document created, added Stakeholder Requirements template 6.3 and requirements 6.4.
1.0.1	18.01.2019	AR	Small format change.
1.1.0	25.01.2019	MBC	Changed project description document history table.
1.1.1	25.01.2019	AR	Changed Abstract to introduction.
1.2.1	22.03.2019	JS	updated requirements.
1.2.2	17.05.2019	MBC	Proofread.

6.2 Introduction

This chapter will introduce the requirements for the project given by KDA.

The CPS team decided to name the requirements given from KDA stakeholder requirements. This makes it easier to keep track of the main requirements and have traceability back to them as the project proceeds.

The template for the Stakeholder requirements will include two columns.

- STRQ. ID, the Stakeholder Requirement ID.
- Requirement, a description of the requirement.

Table 16: Stakeholder requirement template

STRQ. ID	Requirement
X.X.	The system shall ...

6.3 Stakeholder requirement specification

Table 17: Stakeholder requirement specification 1/2

STRQ. ID	Requirement
1.1	The new sensor shall be a contactless position sensor
1.2	The system shall have continuous position adjustability
1.3	The system shall have a linearity of $\pm 0.05\%$ today
1.4	The system shall have a repeatability on 0.001 degrees
1.5	The system shall weigh less than 100 grams
1.6	The sensor shall endure vibrations: sine vibration 30g @ 5-100 Hz, random vibration 25 GRMS @ 20-2000 Hz
1.7	The system shall handle operational temperatures between -80°C and $+120^{\circ}\text{C}$
1.8	The sensor shall have a redundant design
1.9	The sensor shall show no degradation in performance due to variation in homogenic magnetic fields
1.10	The position sensor shall based on change in frequency provide an accurate and analogue position value
1.11	The system shall use LC oscillators to generate frequency
1.12	The CPS shall have the same mechanical interface as the potentiometer currently in use
1.13	Spacing between stator and rotor vs pointing error shall be characterised from 0.5mm to 3mm in order to evaluate the limitations of the sensor

Table 18: Stakeholder requirement specification 2/2

STRQ. ID	Requirement
1.14	Copper thickness of PCB shall be 17, 35 or 70 μm
1.15	Number of copper layers in PCB shall be ≤ 20
1.16	Minimum insulation distance on all layers in PCB shall be $> 120 \mu\text{m}$
1.17	Distance between track and PCB edge $> 0.7\text{mm}$
1.18	Distance between screw hole and PCB edge $> 1.0\text{mm}$
1.19	The test station shall trough a graphical user interface be able to show change in angle[°] and speed [°/s]
1.20	The test station shall log the results of a test in a format which enables for extraction at a later time (all parameters and settings shall be included in the log file)
1.21	The following parameters shall be adjustable; step mode(full, half, 32, 64 and 128), current, distance, distance, speed and acceleration
1.22	The test station shall include a <<Motor enable>> feature for activating the motor bridge (when this mode is activated, the motor shall be in hold mode with the specified current)
1.23	The test station shall include a <<Run>>feature for allowing the motor to start rotating.
1.24	The test station shall have an accuracy of 0.01 degree.

7 Requirements

7.1 Document history

Table 19: Requirements document history

Version	Date	Author	Description
1.0.0	17.01.2019	HS	Document created, added descriptions, priority- and verification tables, 7.3.
1.0.1	18.01.2019	AR	Small format change.
1.1.0	24.01.2019	HFJ	Added section 4 & 5.
1.2.1	25.01.2019	MBC	Changed project Requirements document history table.
1.2.2	25.01.2019	AR	Changed Abstract to introduction.
1.3.0	22.03.2019	JSS	Added new nonfunctional requirements.
1.3.1	26.03.2019	MBC & HS	Proofread and grammar correction.
1.3.2	17.05.2019	MBC	Proofread.

7.2 Introduction

In this chapter the CPS team, in collaboration with KDA, has specified the requirements needed in this project. Requirements are formulated to give a better understanding of what the system should include, and help achieve a better end result. The list of requirements will be updated through the course of the project. To clarify the importance of each requirement, the CPS team has decided to give each requirement a priority shown in the table below.

Table 20: Requirement priority

A	The requirement must be fulfilled to achieve an operational system
B	The requirement should be fulfilled
C	The requirement can be fulfilled

The requirements will undergo a verification process. Verification criterion needs to be fulfilled for a requirement to be met. Methods of verification is shown in the table below.

Table 21: Verification criteria

T	Verification by test
A	Verification by analysis
R	Verification by review of design

Each requirement will have a compliance status. If a requirement has not yet been evaluated the status is set to "Pending" If a method of verification is failed during evaluation the status is set to "Failed". A requirement with the verification ID "T.A.R." only need to be verified through one of the verification methods to be set as "Verified". This means the given requirement has been met.

The requirements will have an individual requirement ID. The Req. ID will have a link back to the wants and need of KDA. The CPS team has decided to include a table for each requirement divided by seven columns. STRQ, the ID of the stakeholder requirement given by KDA. Req. ID, including a requirement number with traceability to the Want/need, and an abbreviation for the verification method. Requirement, describing the requirement. "Pri.", a letter given the priority of the requirement. Origin, an indication of who identified the requirement. "Evaluation", referring to a test ID, analysis document or a design review document. Status, a column for compliance status, "Pending", "Verified" or "Not verified"

Table 22: System requirement template

STRQ	Req. ID	Requirement	Pri.	Origin	Evaluation	Status
1.1	1.1.1 T.A.R.	The system shall...	X	XXX	T. 1.1.1	Pending

7.3 System requirements

Correct and accurate system requirements is crucial to the success of any project. Having a structured and systematic method of collecting, analysing and verifying stakeholder requirements is necessary to develop the product the costumer wants. System requirements is divided into functional and non-functional requirements. Functional requirements define tasks a system should perform. Non-functional requirements define how the task shall be performed.

When using Scrum as a project model the system requirement may be depicted as user stories. User stories shall describe every function and ability that the system is to have. The user stories by itself does not define the requirement in detail, but it contain enough information to derive the requirements. In addition to describing the functions of the system the user stories should have an acceptance criterion to help the development team know when the user story is completed. Requirements may be delivered in the form of use cases from the stakeholder. This is to prevent misunderstanding of the requirements. [22]

7.4 Requirement specification

7.4.1 Functional requirements

Table 23: Functional requirements

STRQ	Req. ID	Requirement	Pri.	Origin	Evaluation	Status
1.1	1.1.1 T.R.	The new sensor shall be a contactless position sensor	A	KDA	Design review	Verified
1.2	1.2.1 T.R.	The CPS shall detect position on continuously rotating mechanism	A	KDA	Design review	Verified
1.10	1.10.1 T.	The CPS shall based on change in frequency provide an accurate and analogue position value	A	KDA	Sub-T-2.0.0, Sub-T-4.0.0	Verified
1.11	1.11.1 T.R.	LC oscillators shall be used to generate the frequency in the CPS	A	KDA	Design review	Verified

7.4.2 Non-functional requirements

Table 24: Non-functional requirements

0 STRQ	Req. ID	Requirement	Pri.	Origin	Evaluation	Status
1.3	1.3.1 T.R.	The linearity of the CPS shall be $\pm 0.05\%$	A	KDA	Not specified	Pending
1.4	1.4.1 T.	The CPS shall have a repeatability of 0.01°	A	KDA	Not specified	Pending
1.5	1.5.1 T.A.	Weight of the CPS shall not exceed 100 grams	A	KDA	Not specified	Pending
1.6	1.6.1 A.	The sensor shall endure vibrations: sine vibration 30g @ 5-100 Hz, random vibration 25 GRMS @ 20-2000 Hz (To be considered. Testing not required)	C	KDA	Not specified	Pending
1.7	1.7.1 T.	The system shall handle operational temperatures between -80°C and $+120^\circ\text{C}$	B	KDA	Not specified	Pending
1.8	1.8.1 A.R.	The sensor shall have a redundant design	A	KDA	Design review	Verified
1.9	1.9.1 T.A.	The sensor shall show no degradation in performance due to variation in homogenic magnetic fields	A	KDA	Not specified	Pending
1.12	1.12.1 R.	The CPS shall have the same mechanical interface as the potentiometer currently in use	A	KDA	Design review	Verified
1.13	1.13.1 T.A.R.	Spacing between stator and rotor vs pointing error shall be characterised from 0.5mm to 3mm in order to evaluate the limitations of the sensor	A	KDA	Design review	Verified

Table 25: Non-functional requirements

STRQ	Req. ID	Requirement	Pri.	Origin	Evaluation	Status
1.14	1.14.1 R	Copper thickness of PCB shall be 17, 35 or 70 μm	A	KDA	Design review	Verified
1.15	1.15.1 R	Number of copper layers in PCB shall be ≤ 20	A	KDA	Design review	Verified
1.16	1.16.1 R	Minimum insulation distance on all layers in PCB shall be $> 120 \mu\text{m}$	A	KDA	Design review	Verified
1.17	1.17.1 R	Distance between track and PCB edge $> 0.7\text{mm}$	A	KDA	Design review	Verified
1.18	1.18.1 R	Distance between screw hole and PCB edge $> 1.0\text{mm}$	A	KDA	Design review	Verified
1.19	1.19.1 T.R.	The test station shall through a graphical user interface be able to show change in angle[°] and speed [°/s]	A	KDA	Design review	Verified
1.20	1.20.1 T.R.	The test station shall log the results of a test in a format which enables for extraction at a later time (all parameters and settings shall be included in the log file)	B	KDA	Design review	Verified
1.21	1.21.1 T.R.	The following parameters shall be adjustable; step mode(full, half, 32, 64 and 128), current, distance, distance, speed and acceleration	B	KDA	Design review	Verified

Table 26: Non-functional requirements

STRQ	Req. ID	Requirement	Pri.	Origin	Evaluation	Status
1.22	1.22.1 T.R.	The test station shall include a <<Motor enable>> feature for activating the motor bridge (when this mode is activated, the motor shall be in hold mode with the specified current)	C	KDA	Design review	Verified
1.23	1.23.1 T.R.	The test station shall include a <<Run>>feature for allowing the motor to start rotating.	A	KDA	Design review	Verified
1.24	1.24.1 T.R.	The test station shall have an accuracy of 0.01 degree.	A	KDA	Initial test	Failed

8 User Story

8.1 Document history

Table 27: User Story document history

Version	Date	Author	Description
1.0.0	23.01.2019	HS	Document created, added User Story template.
1.1.0	24.01.2019	HS	Added the Three C's and INVEST.
1.1.1	25.01.2019	AR	Changed abstract to introduction.
1.2.0	23.03.2019	JSS	Added CPS User Story table.
1.2.1	26.03.2019	MBC	Proofreading and corrections.
1.2.2	17.05.2019	MBC	Proofreading and corrections.

8.2 Introduction

This section includes User Stories, their definition and how the CPS team has implemented them. A User Story is a tool used to get a better understanding of tasks that needs to be done in a project. User Stories are defined in the Product Backlog from the system requirements and are written from a user/consumer perspective where every member of the Scrum Team can write them.

8.3 The Three C's

User Stories have three critical aspects called Card, Conversation and Confirmation.

Card, is a reminder to keep the User Story short. Card can also represent an actual card, the User Story is written on to limit its size. The User Story should have just enough text to identify the requirement, but not make up the entirety of the requirement [24].

- Who are we building it for? Who is the user? - "As a <type of user>
- What are we building? What is the intention? - I want <some goal or objective>
- Why are we building it? What value does it bring for the user? - So that <benefit, value>" [10]

Conversation means the requirement is conversed between developers and customers through thoughts, opinions and feelings. The conversation takes place through the project and is mostly verbal, however it can be supplemented with documentation as well. This is an important process as requirements often tend to change during the course of a project. Which also will lead to new or changed User Stories [24].

The final "C" is confirmation. This is to make sure the User Story is correctly implemented and successfully delivered. This is also called the Acceptance Criteria. At the beginning of an iteration the customer tells the developer what he/she wants. This person also decides the acceptance test of the User Story so that it shows its correctly implemented [24] .

8.4 Implementation

When it comes to writing a good User Story, the team has decided to follow the INVEST model. Invest is an acronym for:

I - Independent
N - Negotiable
V - Valuable
E - Estimable
S - Small
T - Testable
[64]

The CPS has decided to implement User Stories as a high-level definition of a system requirement. This means that a System Requirement will be divided into smaller tasks defined as User Stories. One requirement may have more than one User Story.

Table 28: Example of User Story setup

Req. ID	US. ID	User Story
X.X.X.	X.X.X.X.	As a xxx, xxx want ... so that ...
X.X.X.	X.X.X.X	As a xxx, xxx want ... so that ...

8.5 CPS User Stories

Table 29: Overview of all User Stories and their respective requirements

Story ID	Req ID	User stories
US 1	1.9.1, 1.12.1	As a developer i want a stable frame for the test station so that the frame does not degrade the accuracy of the CPS testing
US 2	1.2.1, 1.21.1, 1.19.1	As a developer I want to track the position of the test motor continuously so that I can know its position at any time
US 3	1.1.1	As a developer I want to make the sensor contactless so that it does not degenerate due to friction
US 4	1.10.1, 1.11.1, 1.13.1	As a developer I want to read the change in magnetic field so that I can get an accurate position
US 5	1.3.1, 1.4.1	As a developer I want a drivetrain with high accuracy so that I can get accurate reading from the CPS
US 6	1.11.1	As a developer I want to construct an oscillator so that I can generate a magnetic field in the coils of the stator
US 7	1.3.1, 1.4.1	As a developer I want to read changes in Hz so I can use this information to calculate the accuracy of the sensor
US 8	1.10.1	As a developer I want to use the change in Hz to calculate the angle in degrees so that I can calculate the position
US 9	1.12.1	As a developer I want the CPS to have the same mechanical interface as the potentiometer so that I can use it in the satellite
US 10	1.5.1	As a developer I want the CPS to weigh less than 100 grams so that I can use it in a satellite
US 11	1.6.1	As a developer I want the CPS to endure vibrations so that i can know its limits
US 12	1.7.1	As a developer I want the CPS to endure high and low temperatures so that I can use it in a satellite
US 13	1.8.1	As a developer I want the CPS to have a redundant design so that I can use a backup if something goes wrong
US 14	1.14.1, 1.15.1, 1.16.1, 1.17.1	As a developer I want to construct a PCB with strict width and height so that I can use it in space
US 15	1.19.1, 1.20.1, 1.21.1	As a developer I want to construct a PCB with strict width and height so that i can use it in space
US 16	1.20.1, 1.22.1, 1.23	As a developer I want to make functions to turn on and run the stepper motor so that I can log and move the stepper motor between tests

9 Testing

9.1 Document history

Table 30: Testing document history

Version	Date	Author	Description
1.0.0	22.01.2019	JSS	Document created.
1.1.0	23.01.2019	JSS	Added abstract, introduction, requirement test, example ID and example test block.
1.2.0	24.01.2019	HS	Added analysis and review.
1.2.2	25.01.2019	AR	Changed abstract to introduction.
1.3.0	20.02.2019	HS	Added test plan and edited test template.
1.3.1	26.03.2019	MBC	Proofreading and corrections.
1.4.0	14.05.2019	HS	Added section.
1.4.1	17.05.2019	MBC	Proofreading and corrections.

9.2 Introduction

Verification of requirements is important to make sure that the system is achievable and ensure progress. This will be done through testing, analysis and review. In this chapter the CPS team will outline a thorough test plan and a test template.

9.3 Test plan

A test plan is important to have before conducting tests. Because the team is using an incremental and iterative project model, multiple stages of testing will be conducted. The tests will either be quantitative or qualitative. Quantitative tests are numeric and measurable. Qualitative properties are properties that are observed and can generally not be measured with a numeric result. To ensure high quality, the standard for the documentation of the tests will include these points:

- Test ID; a unique ID for the test document.
- Requirement ID; to be able to trace it to what requirement the test relates to.
- Name; name/names of the person/persons conducting the test.
- Date and location; the actual date when the test takes place.
- Goal; what is the intention of the test.
- Hypothesis; an idea of the test outcome.
- Pass criteria; what result is needed to pass the test.
- Equipment; a list of equipment used in the test.
- Safety precautions; explanation of safety precautions
- Execution; explain the test procedure in detail step by step and sampling procedure.
- Result; a detailed description of the test result.
- Observations; external factors that may affect the test, temperature, noise etc.
- Analysis; interpretation of data analysis and other observations.
- Conclusion; interpretation of data analysis and other observations and compare it to the hypothesis.

The unique test code will consist of a letter signifying that it is a test and a set of numbers to have a unique id for each test. The test ID will also refer to which concept is being tested, as shown in the example below.

9.3.1 Example test ID

T-C1-1.0.0

9.3.2 Test template

Table 31: Example test template

Test ID	
Requirement ID	
Name	
Date and location	
Goal	
Hypothesis	
Pass criteria	
Equipment	
Safety precautions	
Execution	
Result	
Observations	
Analysis	
Conclusion	

9.3.3 Testing of subsystems

As the team progresses and develops systems and subsystems, testing during the development phase is important and valuable to give an indication of functionality. However, some of the subsystems may not directly be tested up against a requirement. A new Test ID is needed for these subsystems. The subsystems shall almost use the same template as the final tests except from referring to a Requirement ID, the test refers to the subsystem being tested instead. As the subsystems may be used in multiple concepts, a concept will not be referred to either in the Test ID.

9.3.4 Example subsystem test ID

Sub-T-1.0.0

9.3.5 Subsystem test template

Table 32: Example subsystem test template

Test ID	
Subsystem	
Name	
Date and location	
Goal	
Hypothesis	
Pass criteria	
Equipment	
Safety precautions	
Execution	
Result	
Observations	
Analysis	
Conclusion	

10 Sensor design

10.1 Document history

Table 33: Sensor design document history

Version	Date	Author	Description
1.0.0	01.03.2019	AR	Document created.
1.1.0	20.03.2019	AR	Removed phase noise and JFET.
1.2.0	21.03.19	HFJ	Sensor design.
1.3.0	22.03.2019	HFJ	Added sections.
1.3.1	23.03.2019	JSS	Proofread.
1.3.2	26.03.2019	HS & MBC	Proofreading and corrections.
1.3.3	17.05.2019	MBC	Proofreading and corrections.
1.3.0	18.05.2019	HFJ	Added sections.
1.3.0	20.05.2019	AR	Added section and combined two sections.
1.3.1	22.05.2019	HS	Proffread and corrections

10.2 Introduction

To get a deeper understanding of how the sensor system works it is required to understand the technical principle and its purpose. The oscillator circuit is the source for the base frequency that is measured in the system. The frequency is going to be affected by the rotor and it is important to know how this affects the system. This section will also explain the electrical circuits and PCB design of the CPS.

10.3 Oscillator

An oscillator is a circuit that takes direct current (DC) and transforms it into alternating current (AC). The AC can vary in form depending on the configuration of the circuit. Sawtooth, square, triangle and sinusoidal waveform are different periodic signals that can be produced from oscillator circuits [19] [18]. An oscillator can have many configurations depending on the desired output.

Stakeholder requirement 1.11 from table 18 states that the CPS shall use an LC oscillator (inductor (L) and capacitor (C)) to control the frequency. There are 3 possible candidates for this; Hartley oscillator, Colpitts oscillators and Clapp oscillators.

10.4 Oscillator tank

To make a choice for the optimal oscillator for this project it is important to know how the circuits work.

The main part of an oscillator is the tank. This is where the component's values are chosen to achieve the desired frequency. The tank consists of a capacitor and an inductive coil. When the capacitor is charged from direct current it stores this energy and produces a potential. The inductive coil stores the energy in form of an electromagnetic field [9].

When the capacitors have reached their maximum stored potential they start to discharge. Then this electrostatic energy gets transferred to inductor that creates a magnetic field. When capacitors are completely discharged and the inductor reaches its peak value, there is no potential in the tank. The electromagnetic field is induced back to the coil. This charges the capacitor. In an ideal circuit this is going to continue forever.

10.5 Bias network

For the oscillator to work as intended the circuit needs to meet some requirements.

- The phase shift in the feedback loop must be approximately 0° .
- The voltage gain A_{cl} in the closed loop must equal [23].

$$A_{cl} = A_v B \quad (1)$$

Where A_v is the amplifier gain and B is the attenuation of the feedback circuit [23].

$$B = \frac{V_f}{V_{out}} \cong \frac{iX_{C1}}{iX_{C2}} = \frac{X_{C1}}{X_{C2}} = \frac{\left(\frac{1}{2\pi f_r C_1}\right)}{\left(\frac{1}{2\pi f_r C_2}\right)} \quad (2)$$

Cancelling the $2\pi f_r$

$$B = \frac{C_2}{C_1} \quad (3)$$

Since

$$A_v B = 1 \quad (4)$$

Then the equation can be solved for A_v

$$A_v = \frac{1}{B} \quad (5)$$

Replacing B gives

$$A_v = \frac{C_1}{C_2} \quad (6)$$

If the oscillator shall be self-starting, $A_v B$ needs to be greater than 1.

$$A_v > \frac{C_1}{C_2} \quad (7)$$

This is because the calculation is based on an ideal circuit with zero loss of energy. However, a real circuit is going to have some loss of energy in from of heat. Hence the A_v needs to be slightly more than 1 [23].

The phase shift requirements is met by a total shift of 360° because of a 180° in the transistor and another 180° in the feedback circuit.

In figure 10, there is a voltage divider to set the transistor in forward bias. This also sets the Q-point so that the circuit does not go into saturation or cutoff. R17 limits the collector current of the transistor. The bypass capacitor C19 is there to prevent any loss in AC power in the amplified AC signal.

10.6 LC oscillator

10.6.1 Colpitts

The Colpitts (See figure 10) has two capacitors and an inductive coil in parallel in the tank of the circuit. The tank is located on the far-left side and has component C18, C20 and L4. In this configuration the equation for frequency is given as:

$$f_r \cong \frac{1}{2\pi \sqrt{LC_t}} \quad (8)$$

Where C_t

$$C_t = \frac{C_{18}C_{20}}{C_{18} + C_{20}} \quad (9)$$

The advantage of this configuration is the simplicity of the circuit and the stable frequency.

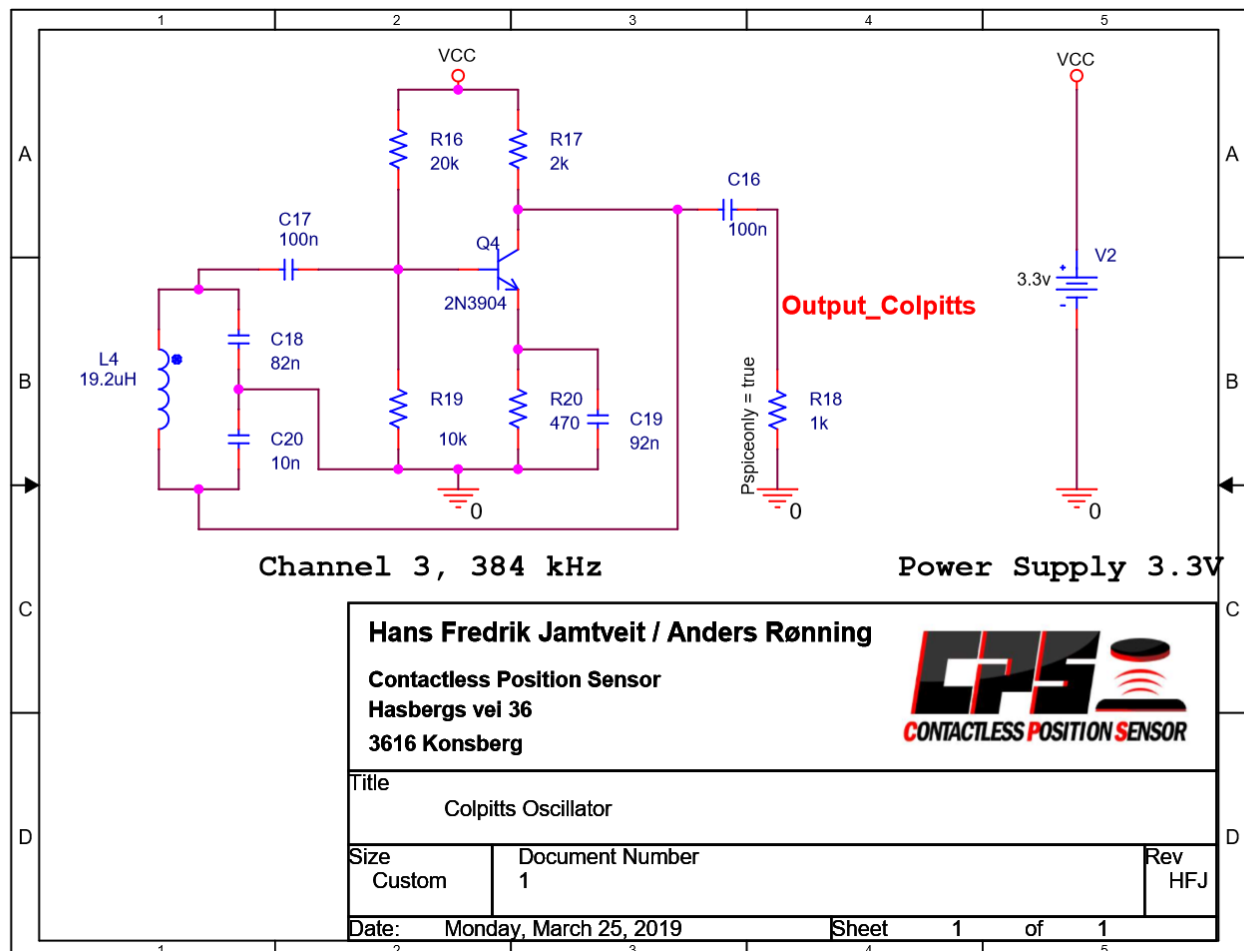


Figure 10: Colpitts oscillator

10.6.2 Clapp

A Clapp oscillator is sometimes viewed as a modifier of Colpitts. The only difference in the circuit layout is an extra capacitor. The new capacitor is placed in series with the coil, and this change results in the adding of a component and change the value of C_T , in the equation.

$$C_t = \frac{1}{\frac{1}{C_1} + \frac{1}{C_2} + \frac{1}{C_3}} \quad (10)$$

$$f_r \cong \frac{1}{2\pi \sqrt{LC_t}} \quad (11)$$

With the new capacitor it is possible to reduce the stray capacitance if the new capacitor is much smaller than the two others.

10.6.3 Hartley

A Hartly configuration looks like a Colpitts oscillator. The difference is that the capacitors and inductive coil have changed place. Consequently, the inductive coils are deciding the attenuation.

$$A_v = \frac{L_1}{L_2} \quad (12)$$

Since the sensor is based on change in the inductive coils, this alternative is not practical to use. It would also make the sensor need twice the number of coils as a Colpitts oscillator or a Clap oscillator.

10.6.4 Amplifier

In the circuit there is a component called amplifier. The purpose of this is as the name suggest, to amplify the signal. This is needed to sustain the desired signal. There are several ways to achieve this. The most common way to do this in an oscillator circuit is by implementing an operational amplifier; Bipolar junction transistor(BJT), or to use a junction gate field-effect transistor(JFET).

10.7 Design tools

10.7.1 OrCAD Capture CIS and OrCAD PCB editor

OrCAD Capture CIS with PSpice and OrCAD PCB editor are the CPS team's tools of choice when developing oscillator circuits designs and PCB design. Schematics of the LC oscillator circuits are drawn in Capture. When the components are correctly placed and connected, the circuit is annotated using the correct standard. Simulating the design with PSpice verifies that the components chosen result in the desired output signal.

Components values in Capture schematics are easily changed, giving the CPS team the ability to test multiple values of components to tune the oscillator frequency to the desired magnitude. PCB design of the CPS is conducted in OrCAD PCB editor. Components used in the schematic in Capture CIS are transferred to OrCAD PCB editor. The design process starts with design of the physical outline of the PCB. Then the components are placed on the board and connected according to the rat's nests that display which pins that shall be connected. OrCAD PCB editor has the possibility to display the PCB in 3D and can be used to review the component placement and the physical outline of the PCB.

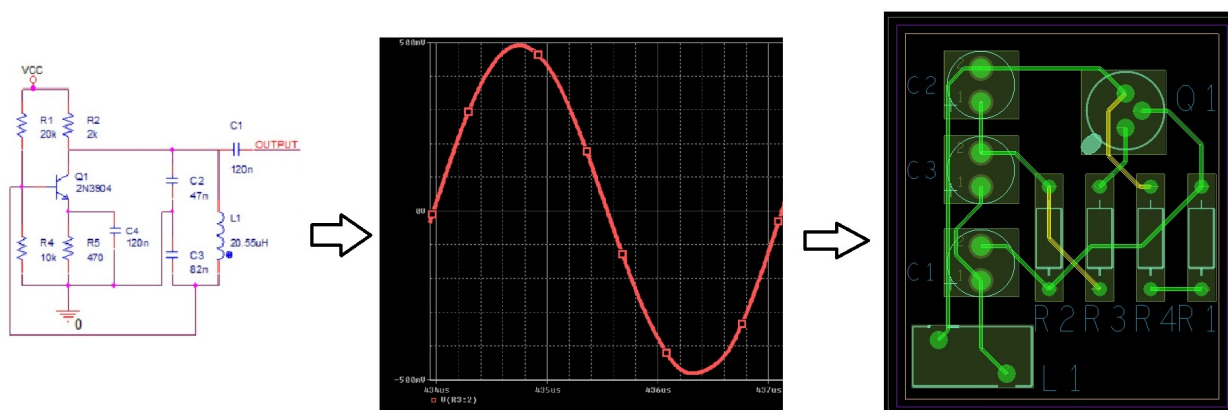


Figure 11: Design process from circuit schematic(left), simulation(middle) to PCB layout(right)

10.7.2 Electronics Explorer Board

Assembly of the oscillator circuits was done on an Electronics Explorer board (EE board) from DIGILENT. The EE board is a breadboard containing a 4-channel oscilloscope, DC/AC power supply and digital ports. The board is connected to a laptop via USB and oscilloscope data can be displayed in the Waveforms software. This software can display the FFT (Fast Fourier Transform) of the input signals.

10.8 CPS circuit design

The Colpitts and the Hartley oscillator are two types of LC oscillators. After reviewing the two designs, the Colpitts oscillator was chosen as the team's base design. Hartley oscillator

circuits contain a LC tank circuit with two inductors in series, with a capacitor in parallel, to generate the oscillation. Colpitts oscillator circuits contain an LC tank circuit with two capacitors in series with an inductor in parallel [19]. By reviewing the two circuits the decision to use the Colpitts oscillator was made. In the CPS, the coil of the oscillator is located on the stator circuit board and this makes the use of the Colpitts oscillator more suitable in the team's design.

10.8.1 First iteration oscillator

The CPS team's first oscillator design used the coils in the prototype from KDA as the inductive element in the oscillator. Knowing the inductance of the coils from testing (72), a simple circuit was designed. The CPS team's approach began by designing a functional circuit in Capture CIS. The oscillation frequency of a Colpitts oscillator is given by (8). Knowing the inductance of the coil and the desired output frequency, rearranging the formula with respect to C gives

$$C = \frac{1}{(2\pi)^2 f^2 L} = \frac{1}{(2\pi)^2 (300 \text{ kHz})^2 (19.2 \text{ }\mu\text{H})} = 14.66 \text{ nF}. \quad (13)$$

The total capacitance of C1 in series with C2 equals 14.66 nF. Finding the individual component values C1 and C2 is done by rearranging the formula

$$C = \frac{1}{\frac{1}{C_1} + \frac{1}{C_2}}. \quad (14)$$

The component values were chosen based on which component the CPS team had access to in early moments of the project. Calculating C1 with C2 as a 47 nF capacitor gives

$$\frac{1}{C_1} = \frac{1}{C} - \frac{1}{C_2} = \frac{1}{14.6 \text{ nF}} - \frac{1}{47 \text{ nF}} = \frac{1}{472.8 \text{ nF}} = 21.8 \text{ nF}. \quad (15)$$

The CPS team used a 18 nF capacitor as it was the closest to 21.8 nF. This gives the values C1 = 18 nF and C2 = 47 nF giving the oscillator the output frequency of

$$f = \frac{1}{2\pi \sqrt{L \cdot \left(\frac{18 \text{ nF} \cdot 47 \text{ nF}}{18 \text{ nF} + 47 \text{ nF}} \right)}} = 318.4 \text{ kHz} \quad (16)$$

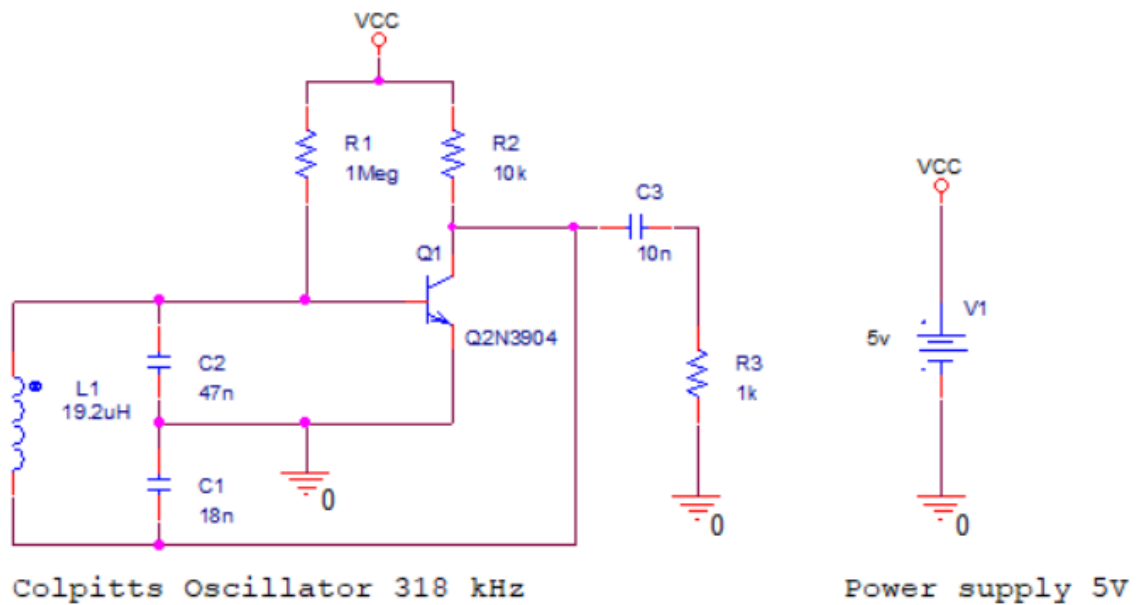


Figure 12: First Colpitts oscillator

Simulating the design before assembling is used as a method of verifying the shape of the sine wave. This is done to adjust the bias network if necessary. Simulating is also used to check if the calculations of the oscillator frequency output is correct. The simulations are compared with the oscilloscope data from the assembled oscillator circuit.

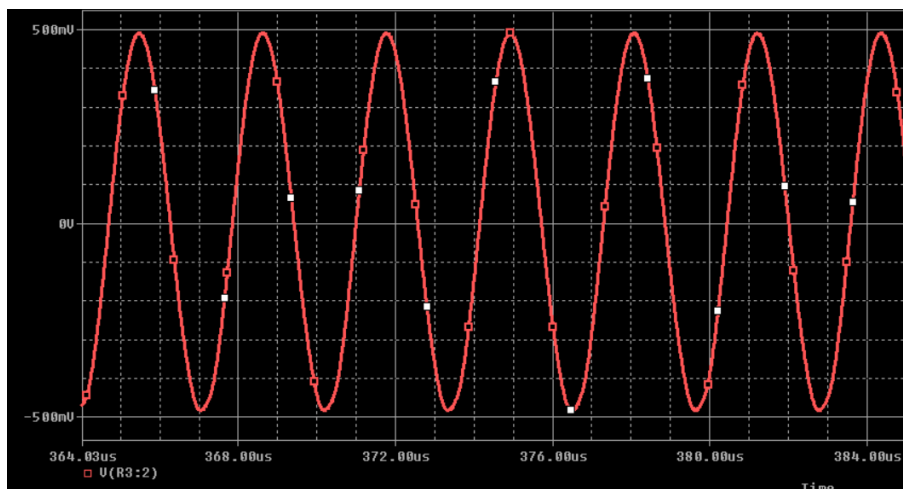


Figure 13: Output signal of first Colpitts oscillator

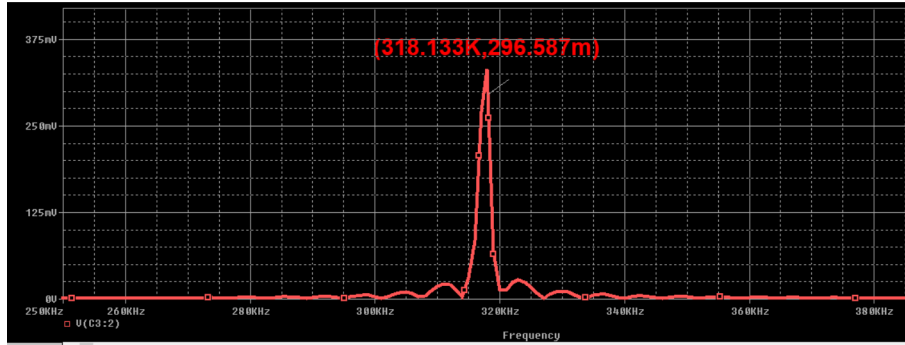


Figure 14: PSpice FFT simulation of first Colpitts oscillator output

Assembly of the oscillator circuit was done on the EE board. The output signal (Figure 15) and the FFT data (Figure 16) is exported from the oscilloscope on the EE board.

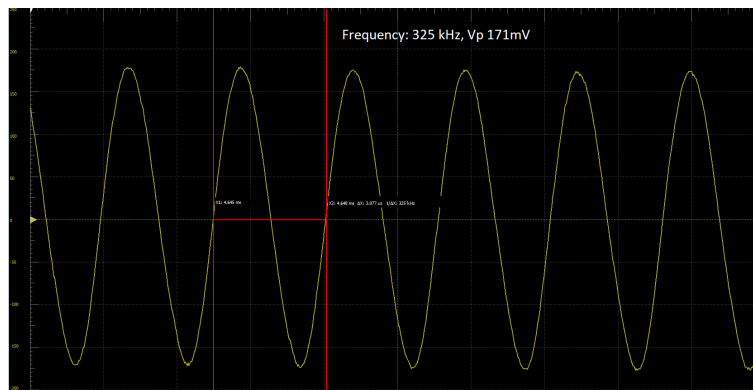


Figure 15: Output signal form breadboard circuit of first Colpitts oscillator

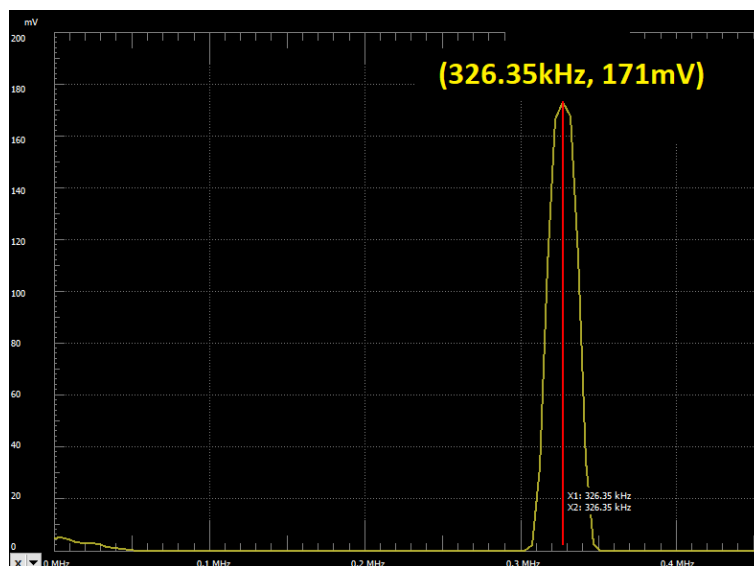


Figure 16: FFT oscilloscope first Colpitts oscillator

Analysis of the simulation and the oscilloscope data shows that the simulated oscillator circuit has an oscillation frequency of 318 kHz and a peak voltage of 296.5 mV, while the breadboard circuit has a oscillation frequency of 326.35 kHz and a peak voltage of 171 mV. This is an increase in frequency of 8.35 kHz and a decrease in peak voltage of 125.5 mV from the simulation to the breadboard circuit.

10.8.2 Second iteration Colpitts oscillator

The next oscillator improvements were done to the bias network of the oscillator, to improve the stability of the oscillator and reduce the phase noise. Improvements are shown in figure 17. The improvements in phase noise is illustrated in figure 18.

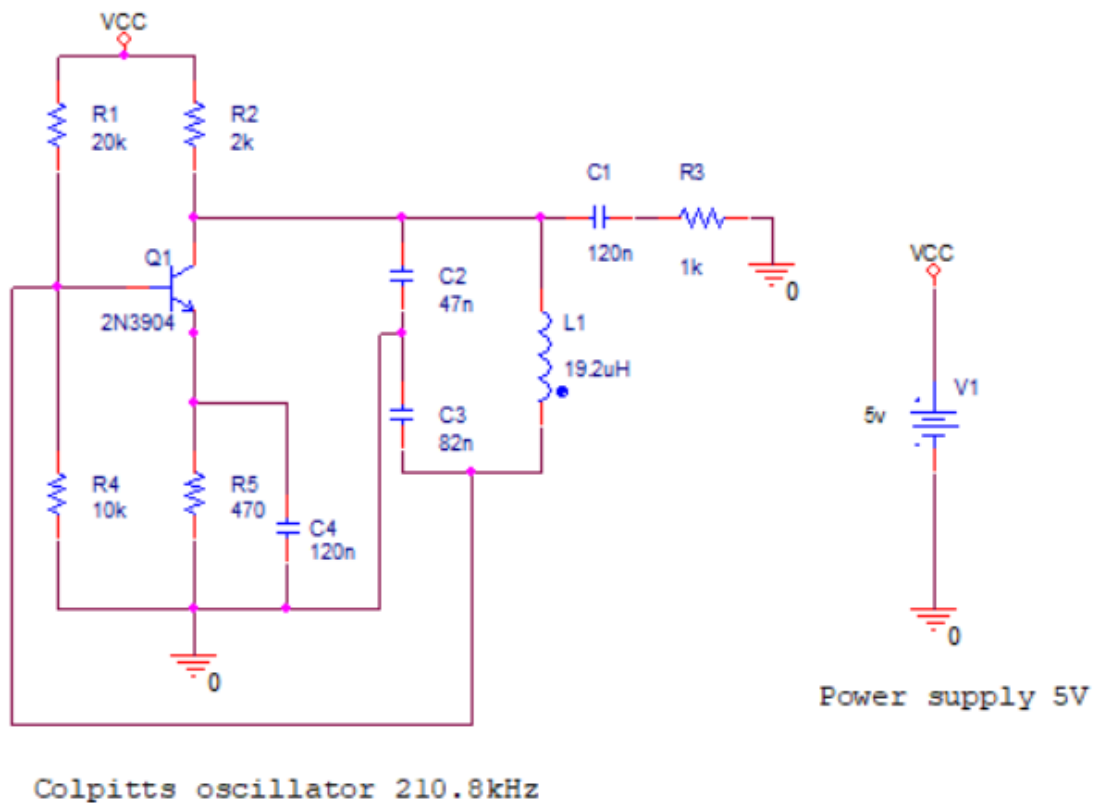


Figure 17: Common emitter Colpitts oscillator

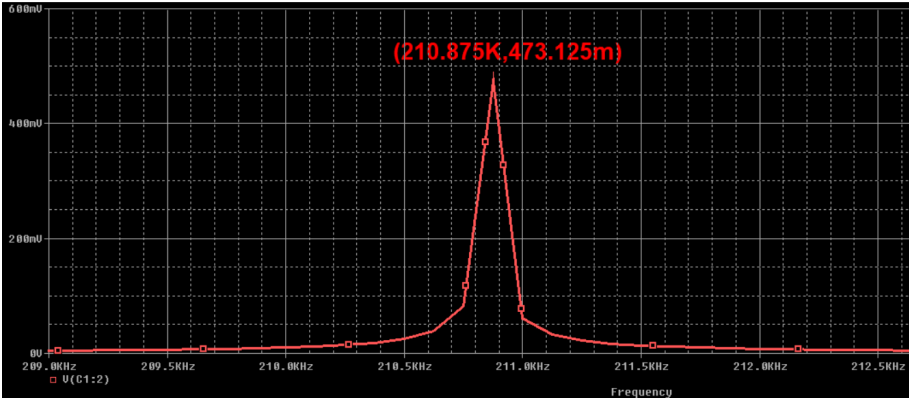


Figure 18: FFT simulation common emitter Colpitts oscillator

10.8.3 Third iteration Colpitts oscillator

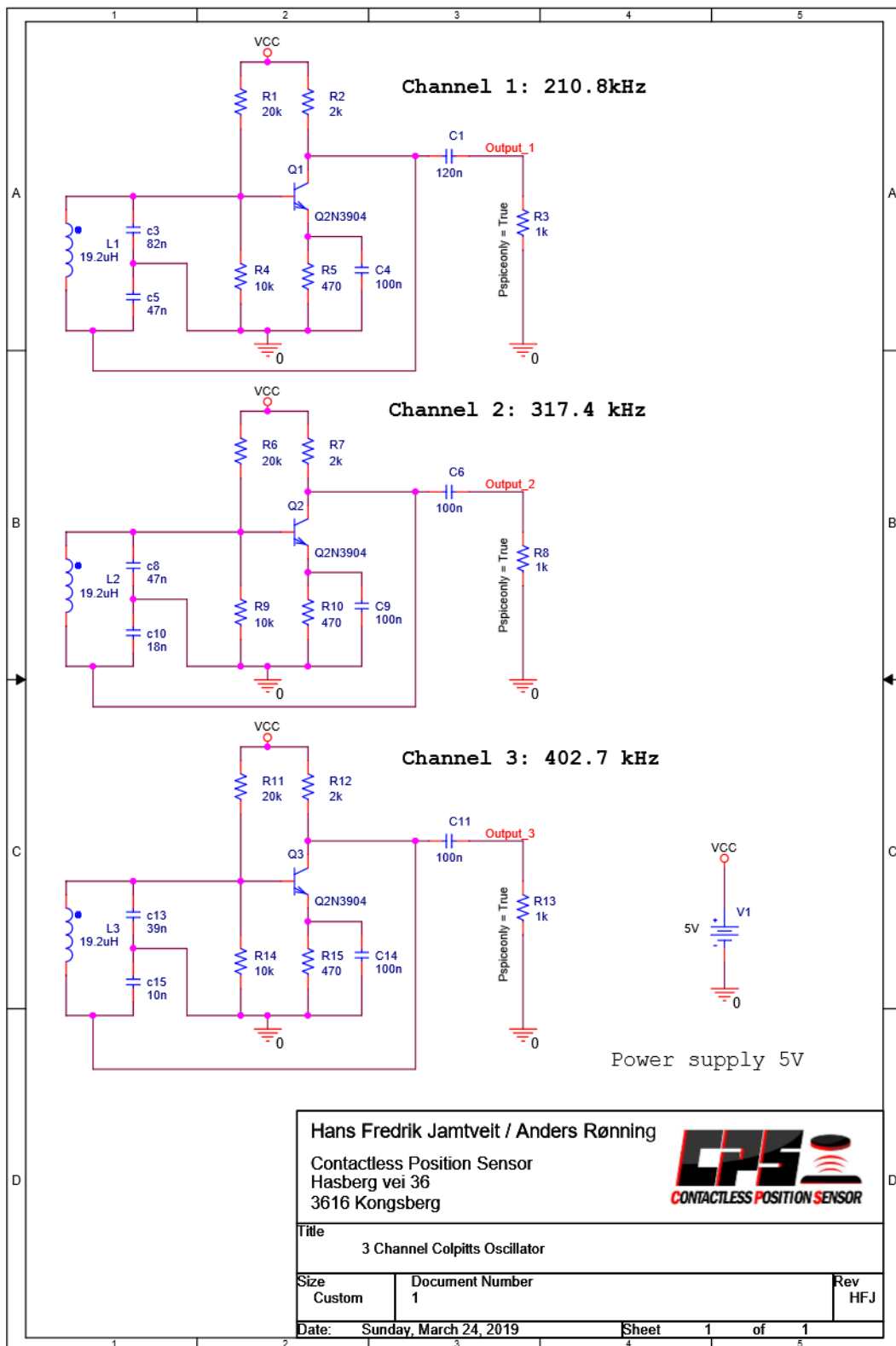


Figure 19: 3 channel Colpitts oscillator

The 3 Channel Colpitts oscillator is based on the common emitter oscillator Colpitts in figure 17. This oscillator is part of the assembly of our first prototype. Figure 20 shows the three output frequencies of the oscillator circuit.

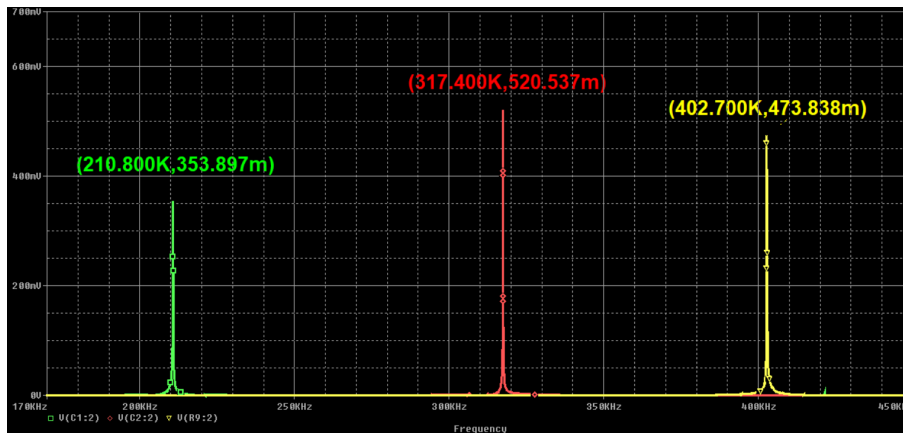


Figure 20: PSpice FFT simulation 3 channel oscillator

Assembly of the third oscillator was done with the prototype from KDA, mounted in a 3D print of the first test station design. This made it possible to see the effect of the copper plate on the coils as well as change in frequency when rotated over the coils.

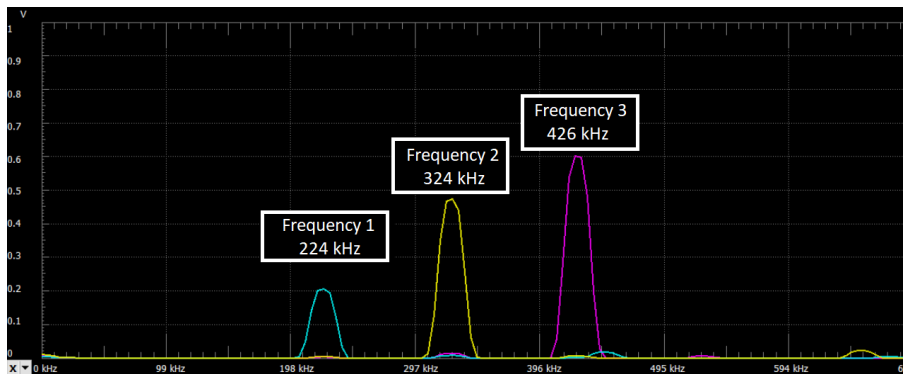


Figure 21: Oscilloscope FFT 3 channel Colpitts oscillator

When the Copper plate covered one of the coils completely, an increase in frequency from 324 kHz to 399 kHz was detected. This is a change of 75 kHz.

In fig 24 the difference in amplitude is because of a missing couplings capacitor and the tank circuit is affecting the signal in the bias network. The AC signal is also difference since the values in the tree circuit have different values.

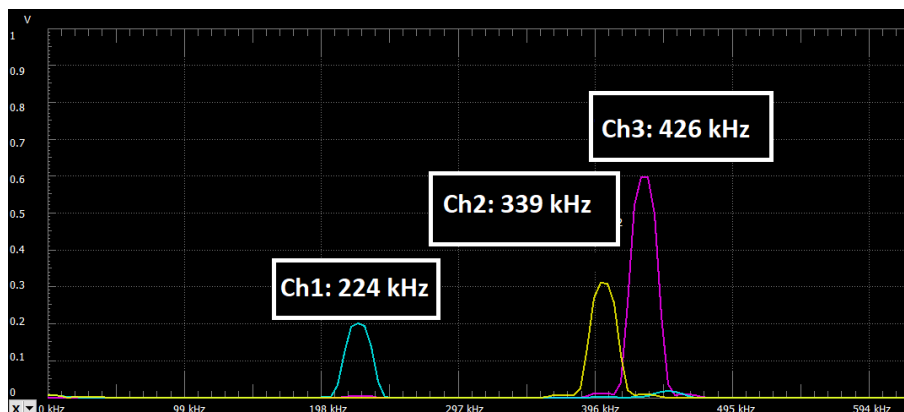


Figure 22: Oscilloscope FFT 3 channel Colpitts oscillator

With this increase, the frequencies of channel 2 and 3 are almost overlapping. This may cause problems when the change in output frequencies shall be used to detect an angle of degree.

10.8.4 Fourth iteration Colpitts oscillator

In the fourth Colpitts oscillator the capacitor values were changed to increase the distance between the 3 output signals.

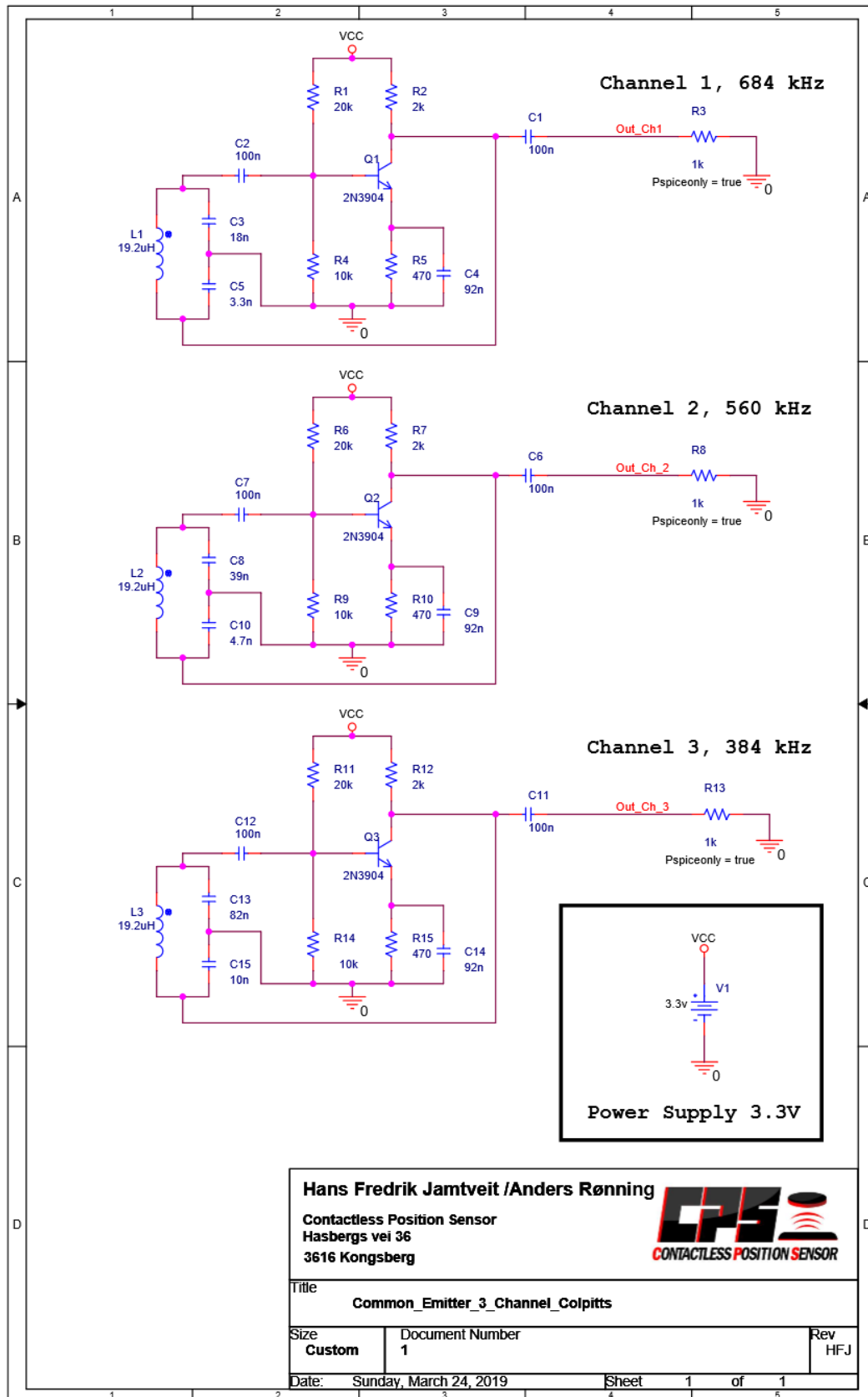


Figure 23: Oscilloscope FFT 3 channel Colpitts oscillator

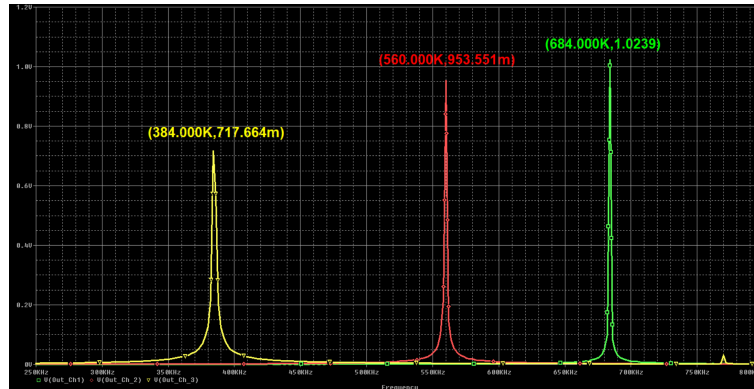


Figure 24: Oscilloscope FFT 3 channel Colpitts oscillator

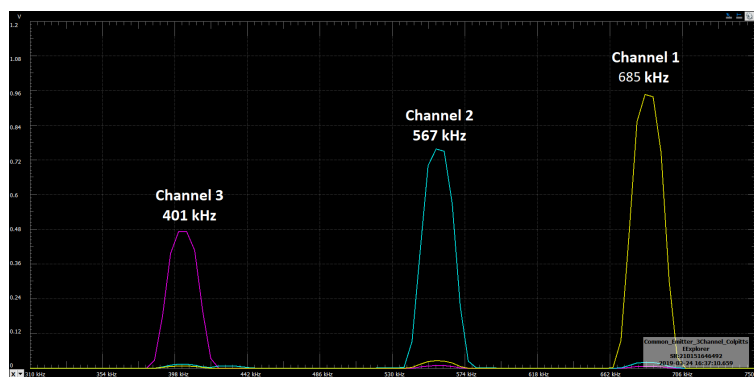


Figure 25: Oscilloscope FFT 3 channel Colpitts oscillator

10.8.5 Fifth iteration oscillators

The fifth iteration on the oscillator circuit began by implementing the new PCB coils into the circuit. The inductance in the new PCB coils were measured at 110 μH which is an increase of 90 μH in comparison to the previous coils. The goal was to implement the new coils in the oscillation circuit and increase the stability of the oscillator. The CPS team had researched LC oscillators that would have a better frequency stability than the Colpitts oscillator, and the team started the design of a Clapp oscillator described in section 10.6.2. This is a modification of the Colpitts oscillator. Clapp oscillator designs with BJT (figure 26) and JFET (figure 27, 28) were tested.

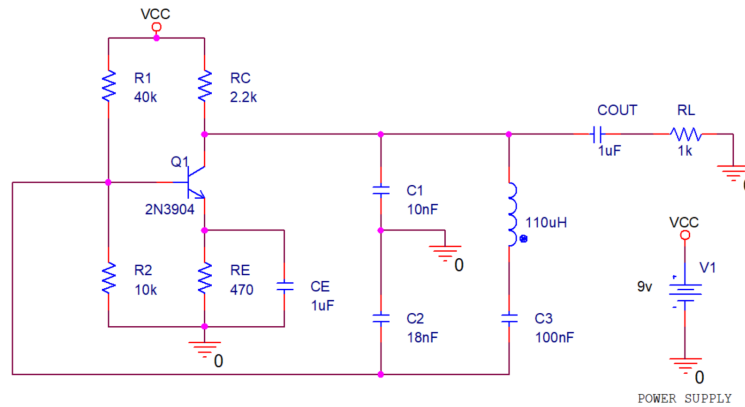


Figure 26: Fifth iteration BJT Clapp oscillator

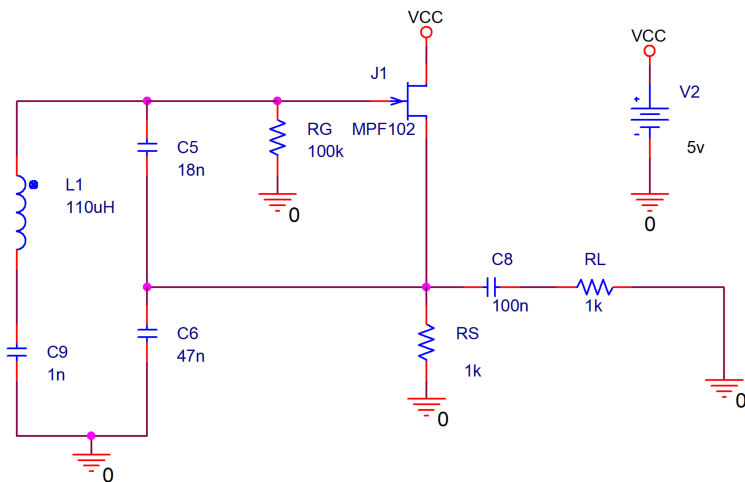


Figure 27: Fifth iteration JFET Clapp oscillator variation 1

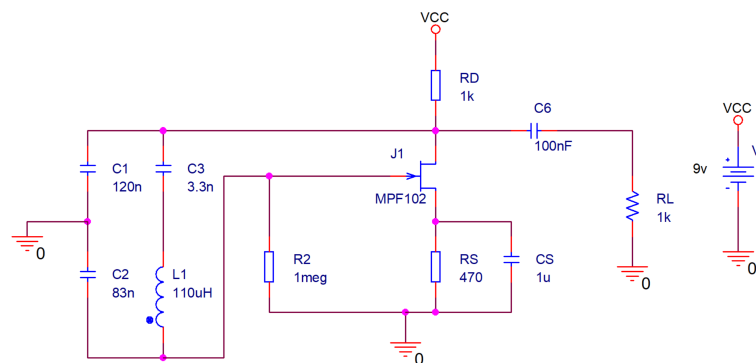


Figure 28: Fifth iteration JFET Clapp oscillator 2

All designs were simulated and assembled on breadboard. Oscillation was achieved but in terms of stability none of the designs attained the stability necessary in the CPS. The BJT

Clapp oscillator was the design that provided the best stability. The decision was made to continue the improvement of the BJT Clapp oscillator in effort to attain the desired frequency stability.

10.8.6 Sixth iteration oscillator

The CPS depends on a integrated counter circuit to detect and measure the output signal of the oscillator, as described in section 18.3.4. Accuracy of the integrated counter circuit is at its most precise when the circuit is provided with a square wave signal. An SN74LS14 Schmitt trigger was used in the sine to square wave conversion before the signal is distributed to the counter circuit. The SN74LS14 was replaced with Schmitt trigger SN74HC14. SN74HC14 has similar characteristics as LS74LS14 and produce the same output. The reason for replacing the SN74LS14 is because of SN74LS14 operating temperature characteristics of 0 °C to +70 °C [55]. 74HC14 has a operation temperature tolerance of -55 °C to +125 °C [56] which is more in compliance with requirement 1.7.1 in Table 26. Bypass capacitors of 100 nF is placed on the VCC input of 74HC14 and SN74LV8154 as recommended in the data sheets. The reason for placing a bypass capacitor on the VCC pin is to ensure a stable VCC and to filter noise from the DC power supply. 74HC14 has with a VCC of +4.5 V a V_{T+} at 2.5 V and a V_{T-} at 1.6 V. In order to function as required the input signal must have a V_{pp} of + 3 V offset of above + 1 V to + 2 V. To apply a offset to the signal of the oscillator a voltage divider is placed on the input of the Schmitt trigger Figure 29. This provides the offset needed. Adding the Schmitt trigger with the voltage divider to the circuit, changed the load of the oscillator circuit. Adaptations had to be made to the bias network of the amplifier in order to make the circuit functional.

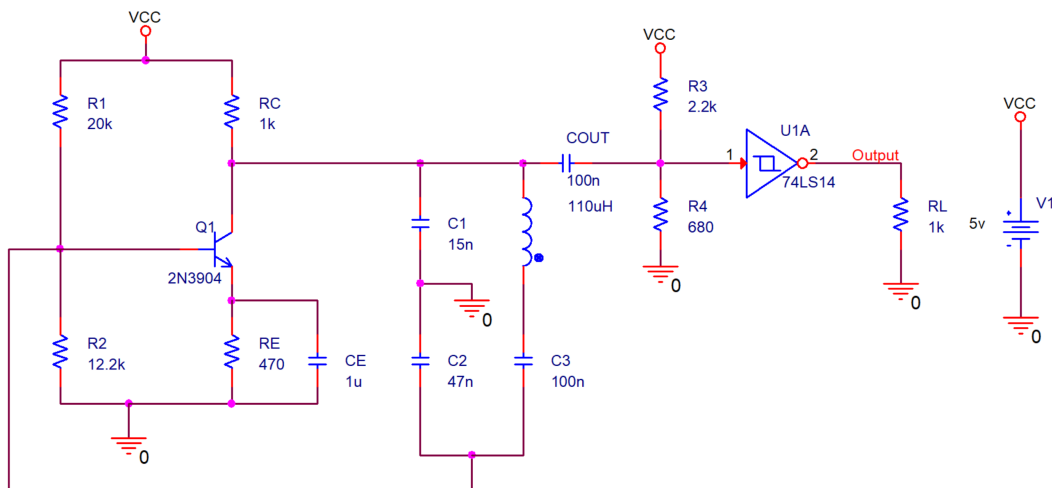


Figure 29: Sixth iteration BJT Clapp oscillator

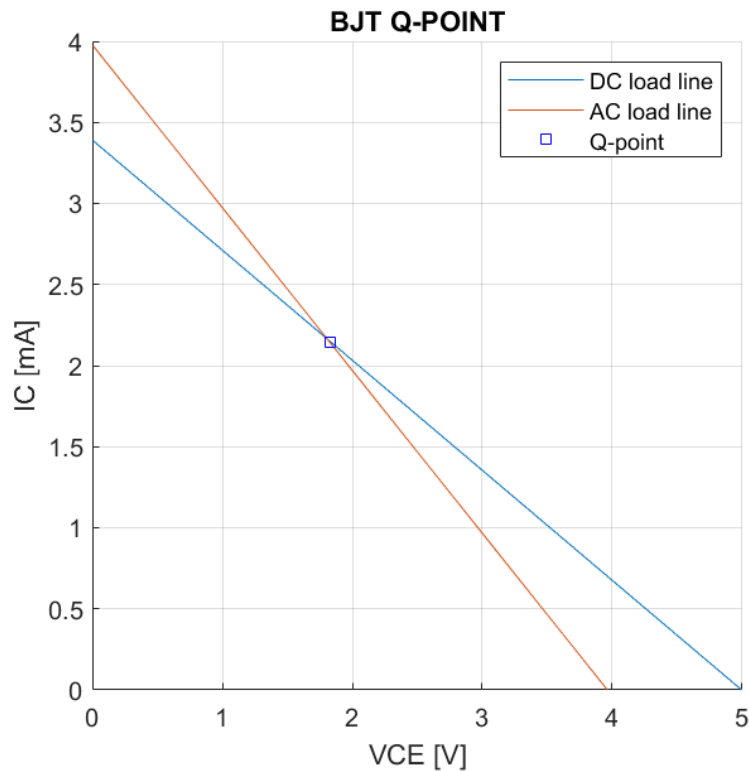


Figure 30: Q-point of amplifier in figure 29

Simulation of the circuit in PSpice showed that the amplifiers input signal was driven into saturation. This was caused by the magnitude of the input signal which was too large in relation to the placement of the Q-point. This saturation became more significant when the circuit output was inspected on an oscilloscope, as illustrated in figure 31.

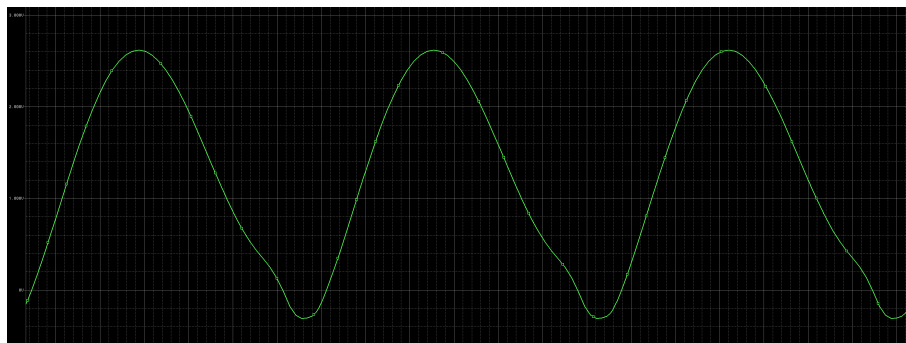


Figure 31: Sixth oscillator iteration input signal on Schmitt trigger

When the circuit was assembled and tested the stability of the circuit did not meet the demand.

10.9 Final oscillator iteration

After review of the oscillator in section 10.8.6, the CPS team concluded that the reason why the the oscillator achieved the desired stability was mostly due to the large gain in the amplifier circuit, and low capacitance tolerance in terms of temperature and voltage variation in the capacitors. The AC voltage gain of the amplifier is

$$\frac{R_c}{r'_e} = A_v, \quad (17)$$

where r_c is R_C and R_L in parallel, R_L becomes the resistance of the voltage divider on the input of the Schmitt trigger

$$R_c = \frac{R_C R_L}{R_C + R_L}, \quad (18)$$

$$R_c = \frac{1 \text{ k}\Omega \cdot 520 \text{ }\Omega}{1 \text{ k}\Omega + 520 \text{ }\Omega} = 341 \text{ }\Omega, \quad (19)$$

and r_e is

$$r'_e = \frac{25 \text{ mV}}{IE}, \quad (20)$$

with IE calculated to 2.16 mA r'_e is

$$r'_e = \frac{25 \text{ mV}}{2.16 \text{ mA}} = 11.57 \text{ }\Omega, \quad (21)$$

this leads A_v to become

$$A_v = \frac{341 \text{ }\Omega}{11.57 \text{ }\Omega} = 29.74. \quad (22)$$

A voltage gain of 29.74 would cause the oscillator to become unstable. This gain exceeds the gain needed to start and then retain the oscillation with the capacitors $C_1 = 15 \text{ nF}$ and $C_2 = 47 \text{ nF}$ present in the circuit. A more comprehensive calculation was conducted with MATLAB in order to acquire the correct Q-point and AC voltage gain of the amplifier.

10.9.1 Common emitter amplifier

When designing the Clapp oscillator, an amplifier stage is needed to start the oscillation. When the oscillation has reached its desired amplitude, the amplifier must sustain the oscillation created in the oscillation tank. The amplifiers in the CPS oscillator circuits are BJT common emitter with an voltage-divider and biased amplifier illustrated in figure 32. The values of the circuit were calculated in MATLAB. The design and simulations of these amplifiers were done in Orcad capture with PSpice. The components chosen in the design-process of the amplifier stage are shown in table 34. Tolerance of the components is important in order to achieve the required stability. Component tolerances is displayed in table 35. Tolerances of capacitors is found in data sheet[12]. Tolerance of the resistors is found by inspection.

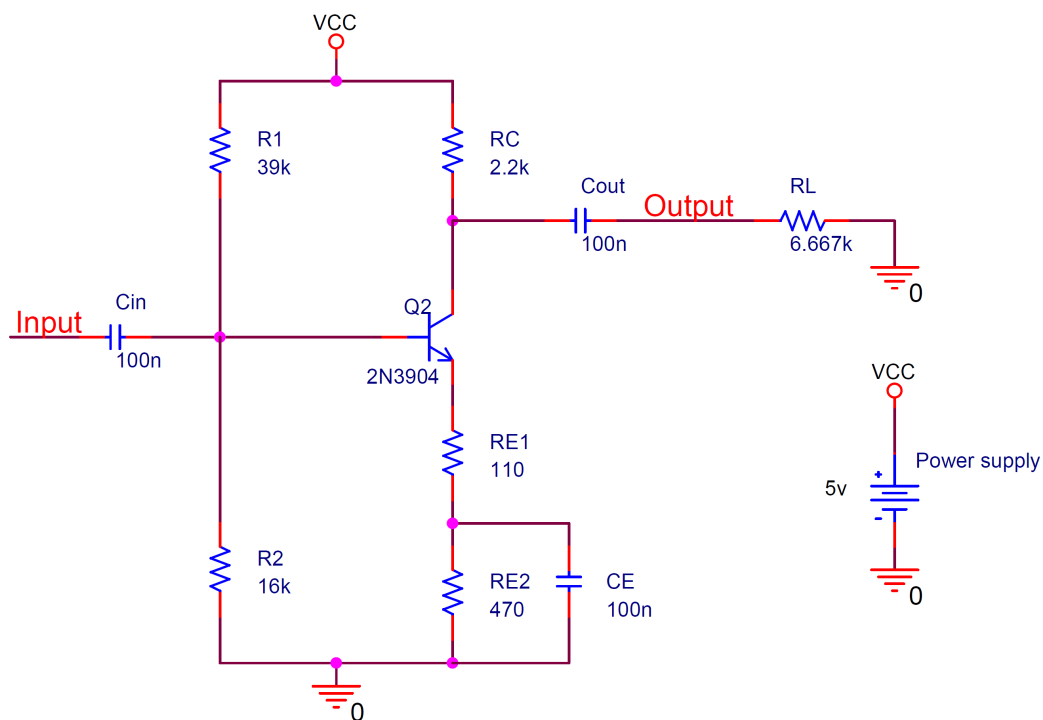


Figure 32: Common emitter amplifier

Table 34: BJT amplifier components

Component	Type	Reference	Component value	Unit
Voltage supply	DC supply	V1	5	V
Transistor	2n3904	Q1	100	β
Resistor	Metal film	R1	39	k Ω
Resistor	Metal film	R2	16	k Ω
Resistor	Metal film	RC	2.2	k Ω
Resistor	Metal film	RE1	91	Ω
Resistor	Metal film	RE2	470	Ω
Capacitor	Ceramic X7R	CE	100	nF
Capacitor	Ceramic X7R	Cin	100	nF
Capacitor	Ceramic X7R	Cout	100	nF

Table 35: Capacitor tolerance amplifier

Component	Type	Temp. tolerance	Unit. tolerance
R1	Metal film	50 ppm	$\pm 1\%$
R2	Metal film	50 ppm	$\pm 1\%$
RC	Metal film	50 ppm	$\pm 1\%$
RE1	Metal film	50 ppm	$\pm 1\%$
RE2	Metal film	50 ppm	$\pm 1\%$
CE	Ceramic X7R	$\pm 15\%$, - 55 to + 125 C °	± 10 F
Cin	Ceramic X7R	$\pm 15\%$, - 55 to + 125 C °	± 10 F
Cout	Ceramic X7R	$\pm 15\%$, - 55 to + 125 C °	± 10 F

The DC analysis [19] of the amplifier started with calculating the base voltage of the transistor. The voltage-divider must supply the base of the transistor a minimum of 0.7v in order to have the transistor operate in its active region. The Thevenin equivalent of the base voltage provided is given by

$$V_{TH} = \left(\frac{R_2}{R_1 + R_2}\right)V_{CC}, \quad (23)$$

hence

$$V_{TH} = \left(\frac{16 \text{ k}\Omega}{39 \text{ k}\Omega + 16 \text{ k}\Omega}\right)5 \text{ V} = 1.4545 \text{ V}. \quad (24)$$

A V_{TH} of 1.4545 V exceeds the required 0.7 V the transistor needs on the base terminal in order become active. When the BJT is active current is allowed to flow from collector to emitter. Work point of the transistor(Q-point) is the point on the DC load line where the current going trough the collector(I_CQ) and the voltage drop across the collector and emitter(V_{CEQ}) intersect. Calculation of I_CQ and V_{CEQ} is needed in order to determine the Q-point. To find the emitter current the Thevenin equivalent of the base resistance R_{TH} is calculated. This is given by

$$R_{TH} = \frac{R_1 R_2}{R_1 + R_2}, \quad (25)$$

hence

$$R_{TH} = \frac{39 \text{ k}\Omega \cdot 16 \text{ k}\Omega}{39 \text{ k}\Omega + 16 \text{ k}\Omega} = 11.345 \text{ k}\Omega. \quad (26)$$

The emitter current is found by

$$I_E = \frac{V_{TH} - V_{BE}}{(R_{E1} + R_{E2}) + (R_{TH}/\beta_{DC})}, \quad (27)$$

resulting in

$$I_E = \frac{1.4545 \text{ V} - 0.7 \text{ V}}{(91 \text{ }\Omega + 470 \text{ k}\Omega) + (11.345 \text{ k}\Omega/100)} = 1.1187 \text{ mA}, \quad (28)$$

where $V_{BE}= 0.7 \text{ V}$ and β_{DC} are found in the data sheet of 2N3904 [11]. The current of the collector is given by

$$I_C = I_E - I_B. \quad (29)$$

In order to calculate the base current the resistance of the base terminal is found

$$R_{IN(BASE)} = \frac{\beta_{DC} V_{TH}}{I_E}. \quad (30)$$

This results in

$$R_{IN(BASE)} = \frac{100 \cdot 1.4545 \text{ V}}{1.1187 \text{ mA}} = 130 \text{ k}\Omega, \quad (31)$$

which leads to

$$I_B = \frac{V_{TH}}{R_{IN(BASE)}}, \quad (32)$$

and by substituting with values

$$I_B = \frac{1.4545 \text{ V}}{130 \text{ k}\Omega} = 11.19 \text{ }\mu\text{A}, \quad (33)$$

the base current can be found. From I_E and I_B , I_C can be calculated

$$I_C = 1.1187 \text{ mA} - 11.19\mu\text{A} = 1.1076 \text{ mA}. \quad (34)$$

In order to find the DC Q-point calculation of the emitter voltage(V_E) and collector voltage(V_C) is required.

$$V_E = (R_{E1} + R_{E2})I_E, \quad (35)$$

$$V_E = (91\Omega + 470\Omega)1.1187 \text{ mA} = 627.6 \text{ mV}, \quad (36)$$

$$V_C = V_{CC} - R_C I_C, \quad (37)$$

$$V_C = 5 \text{ V} - (2.2 \text{ k}\Omega \cdot 1.1076 \text{ mA}) = 2.5634 \text{ V}. \quad (38)$$

The voltage between the collector and emitter terminal of the transistor is calculated

$$V_{CE} = V_C - V_E, \quad (39)$$

$$V_{CE} = 2.5634 \text{ V} - 627.6 \text{ mV} = 1.9357 \text{ V}. \quad (40)$$

$I_{C(SAT)}$ is found by

$$I_{C(sat)} = \frac{V_{CC}}{R_C + R_{E1} + R_{E2}}, \quad (41)$$

$$I_{C(sat)} = \frac{5 \text{ V}}{2.2 \text{ k}\Omega + 91 \text{ }\Omega + 470 \text{ }\Omega} = 1.8109 \text{ mA}. \quad (42)$$

$V_{CE(off)}$ is the same as V_{CC} and equals 5 V. DC load line coordinates become $(V_{CE(off)}, I_{C(sat)}) = (5 \text{ V}, 1.8109 \text{ mA})$. The AC load line of the amplifier is $(V_{ce(off)}, I_{c(sat)})$. R_c is the resistance the AC signal sees on the output of the amplifier. R_c is the parallel connection of R_C and R_L . R_L of the amplifier is the two resistors in the voltage divider placed on the input of the schmitt trigger figure40. The voltage divider provides the offset voltage to the oscillator output signal before the signal enters the schmitt trigger. The R_{TH} of the voltage divider of the schmitt trigger is 6.667 k Ω . R_c is

$$R_c = \frac{R_C R_L}{R_C + R_L}, \quad (43)$$

$$R_c = \frac{2.2 \text{ k}\Omega \cdot 6.667 \text{ k}\Omega}{2.2 \text{ k}\Omega + 6.667 \text{ k}\Omega} = 1.654 \text{ k}\Omega, \quad (44)$$

This leads the AC load line ($V_{ce(off)}, I_{c(sat)}$) to become

$$V_{ce(off)} = V_{CEQ} + (I_{CQ} r_c), \quad (45)$$

$$V_{ce(off)} = 1.9357 \text{ V} + (1.1076 \text{ mA} \cdot 1.654 \text{ k}\Omega) = 3.7678 \text{ V}. \quad (46)$$

$I_{c(sat)}$ is calculated by

$$I_{c(sat)} = I_{CQ} + \left(\frac{V_{CEQ}}{r_c}\right), \quad (47)$$

$$I_{c(sat)} = 1.1076 \text{ mA} + \frac{1.9357 \text{ V}}{1.654 \text{ k}\Omega} = 2.2778 \text{ mA}. \quad (48)$$

Amplifier Q-point is plotted in figure 33.

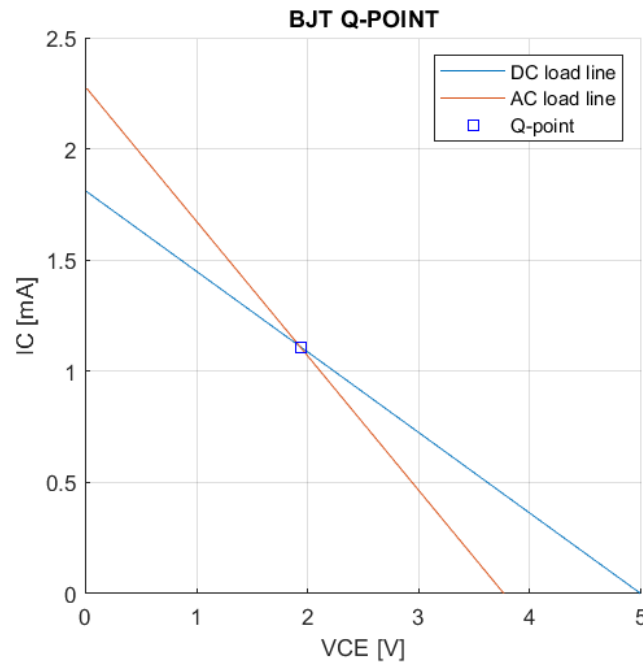


Figure 33: Q-point of amplifier in figure 32, $(V_{CEQ}, I_{CQ}) = (1.9357 \text{ V}, 1.1076 \text{ mA})$

This places the Q-point at $(V_{CEQ}, I_{CQ}) = (1.9357 \text{ V}, 1.1076 \text{ mA})$. The Q-point is placed near the center of the DC load line in order to prevent clipping.

Voltage gain of the amplifier is the output resistance of the amplifier divided by the emitter resistance

$$A_v = \frac{R_c}{r'_e + R_{E1}}, \quad (49)$$

where r'_e is

$$r'_e = \frac{25 \text{ mV}}{I_E}, \quad (50)$$

$$r'_e = \frac{25 \text{ mV}}{1.1187 \text{ mA}} = 22.346 \Omega, \quad (51)$$

hence

$$A_v = \frac{1.654 \text{ k}\Omega}{22.98 \Omega + 91 \Omega} = 14.59. \quad (52)$$

By itself this voltage gain of this amplifier would lead to clipping if a signal with the same magnitude as the signal provided by the oscillation tank circuit. Because this amplifier is used as the amplifier in a oscillator circuit the gain is just right to get the oscillation started and to maintain the oscillation when the desired amplitude of the output signal is reached. This improves the stability of the oscillator.

10.9.2 Clapp oscillator circuit

Advancements in the development of the counter circuit Section 18.3.4 has given the ability to separately monitor the 3 outputs of the CPS sensor. This gives the advantage of having 3 oscillator with the same output frequency. Form the test 73 it was found that the inductance stability in the frequency range from 40 kHz to 690 kHz was $111 \mu\text{H} \pm 2 \mu\text{H}$. Due to the stability of the region this is a appropriate frequency region to operate in. Calculations in Section 19 proves that the ideal frequency range is between 400 kHz and 600 kHz. Operating in this region gives the delta frequency need to achieve the required accuracy of the CPS. The components used in the tank circuit of the final oscillator is shown in table 36.

Table 36: Clapp oscillator tank components

Component	Type	Reference	Value	Unit
Coil	6 layer curved rectangular PCB	L	110	μH
Capacitor	Ceramic COG	C1	2	nF
Capacitor	Ceramic COG	C2	6.8	nF
Capacitor	Ceramic COG	C3	100	nF

From(10) and(54) frequency of oscillation can be determined

$$C_t = \frac{1}{2 \text{ nF}} + \frac{1}{6.8 \text{ nF}} + \frac{1}{100 \text{ nF}} = 1.5219 \text{ nF}, \quad (53)$$

$$f_r \cong \frac{1}{2\pi\sqrt{110 \mu\text{H} \cdot 1.5219 \text{ nF}}} = 388.983 \text{ kHz.} \quad (54)$$

The gain of the amplifier voltage gain(52) is reduced by implementing R_{E1} and the increasing R_L by increase of the resistors in the voltage divider in the Schmitt trigger stage of the CPS. This leads to $A_v = 14.594$. This is enough to start the oscillation and low enough to have a stable output frequency.

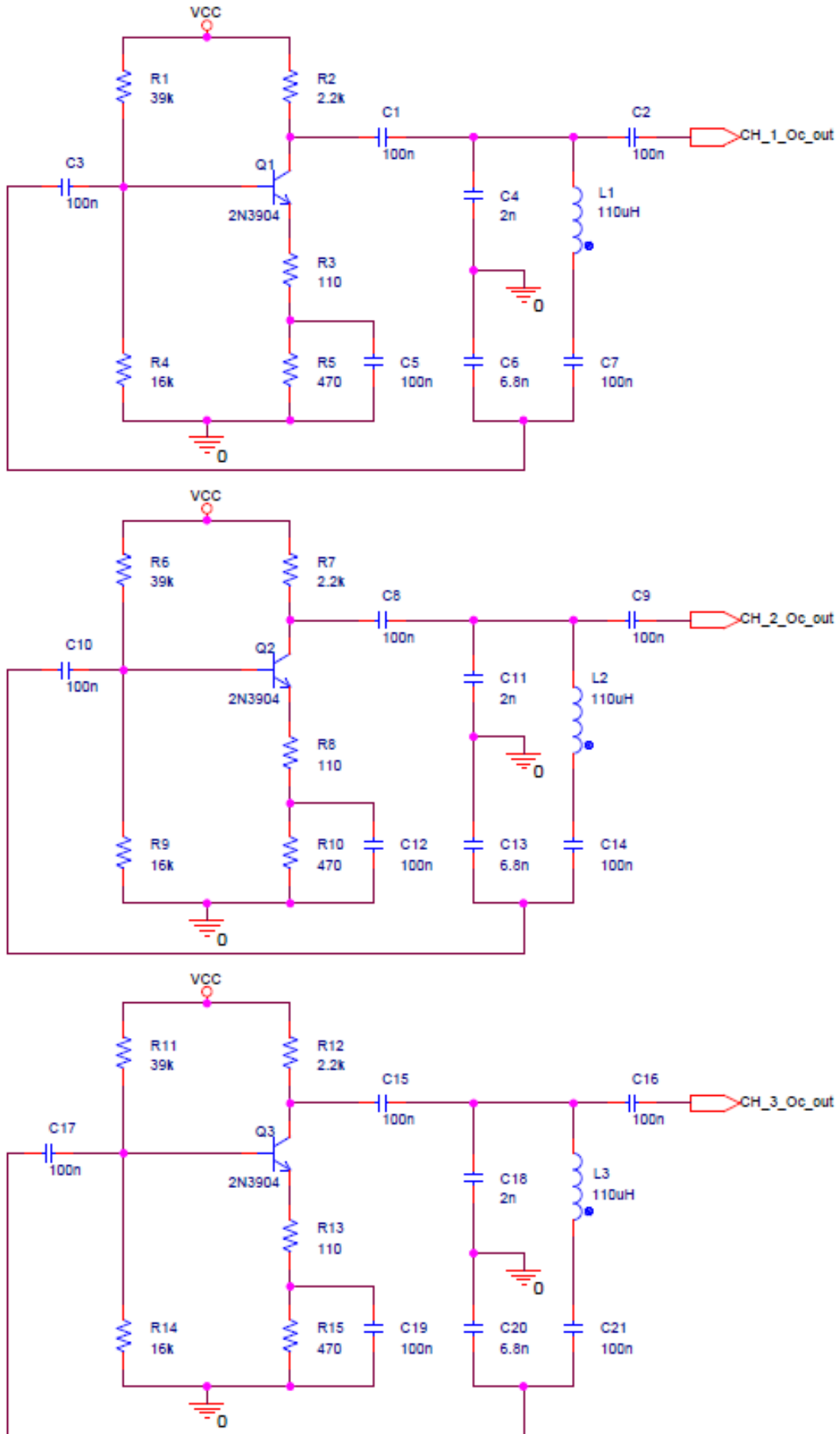


Figure 34: Final iteration, 3 channel Clapp oscillator

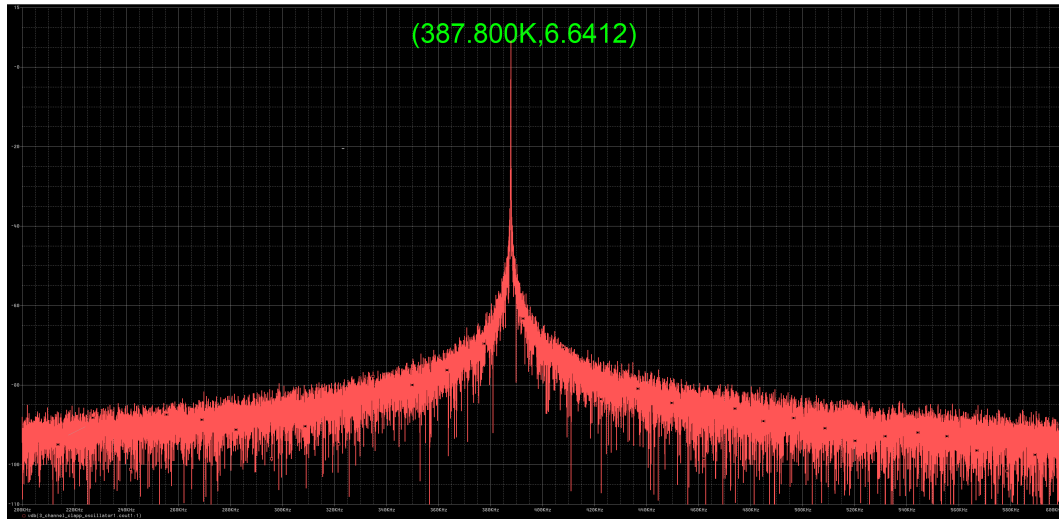


Figure 35: PSpice FFT final oscillator

FFT simulation was conducted PSpice. Plot of simulation is shown in figure 35. VdB peak value = 6.6412 dB, $f=387.800$ kHz. X-axis = 200 kHz to 600 kHz, Y-axis= - 110 dB to + 15 dB.

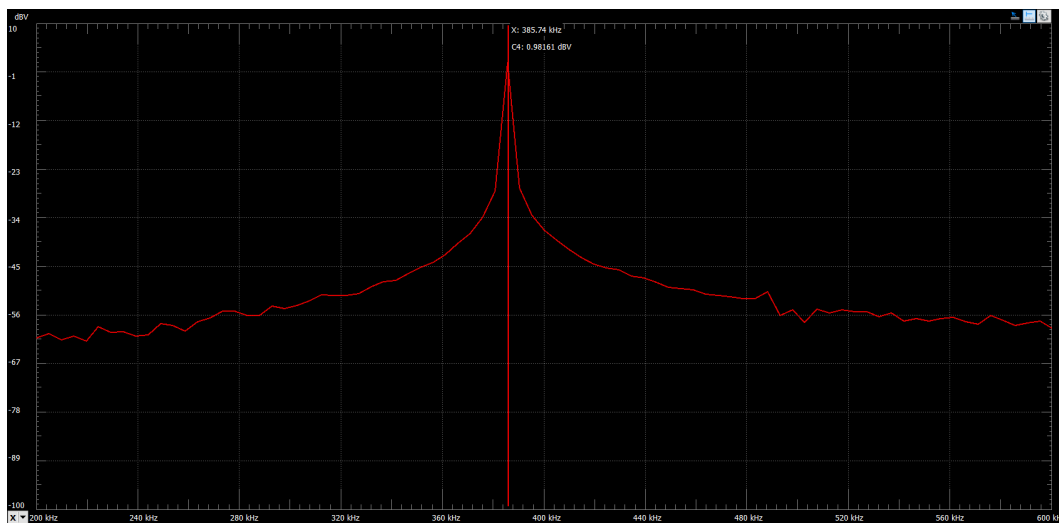


Figure 36: Oscilloscope FFT of final oscillator output signal.

FFT analysis was conducted on oscilloscope plot of analysis shown in figure 36. VdB peak value = 0.9816 dB, $f = 385.750$ kHz. X-axis = 200 kHz to 600 kHz, Y-axis= -100 dB to + 10 dB. This is compared with the simulation displayed in figure 35. A reduction of 5.6596 dB in vdB peak value is noticed. The frequency of the oscillator plot has is 2.05 kHz lower than the simulation. Spectrum analysis was conducted on the spectrum analysis application on the EE board. Result of analysis is shown in figure 37. X-axis = 200 kHz to 600 kHz, Y-axis= - 100 dB to + 10 dB.

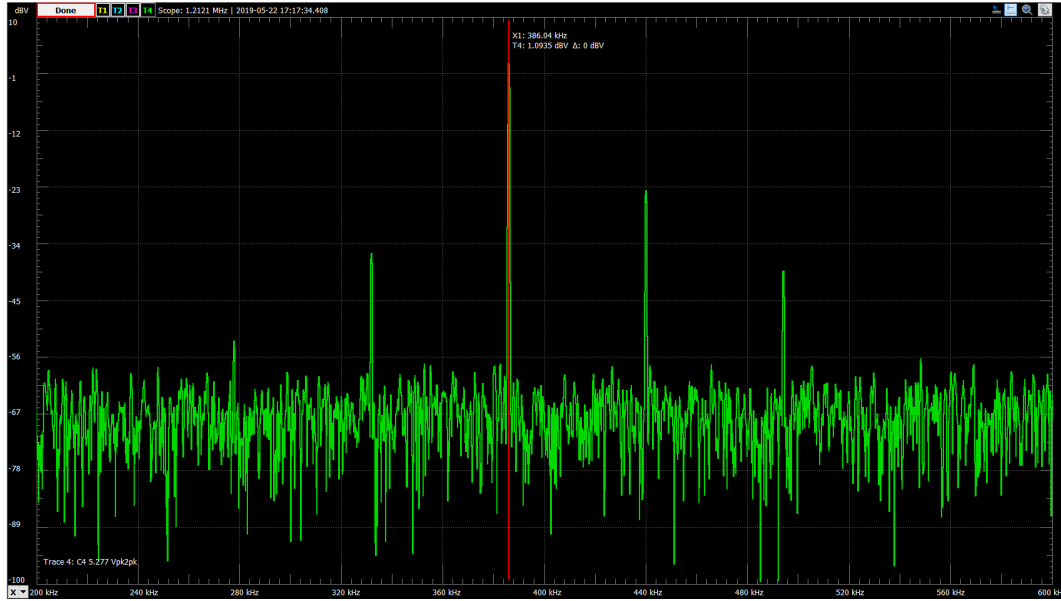


Figure 37: Spectrum analysis of signal from final oscillator

The improvements done to the bias network of the amplifier corrected the placement of the Q-point. PSpice simulation of signal is shown in figure 38. Plot from oscilloscope showing output signal can be seen in figure 39.



Figure 38: PSpice simulation of output signal of final oscillator

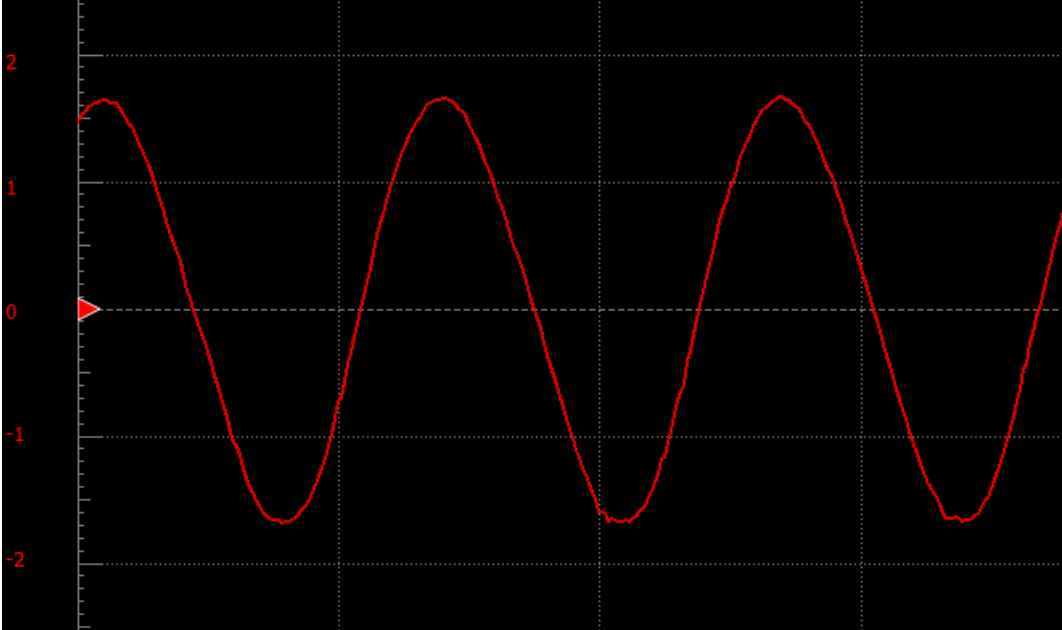


Figure 39: Plot from oscilloscope of output signal of final oscillator

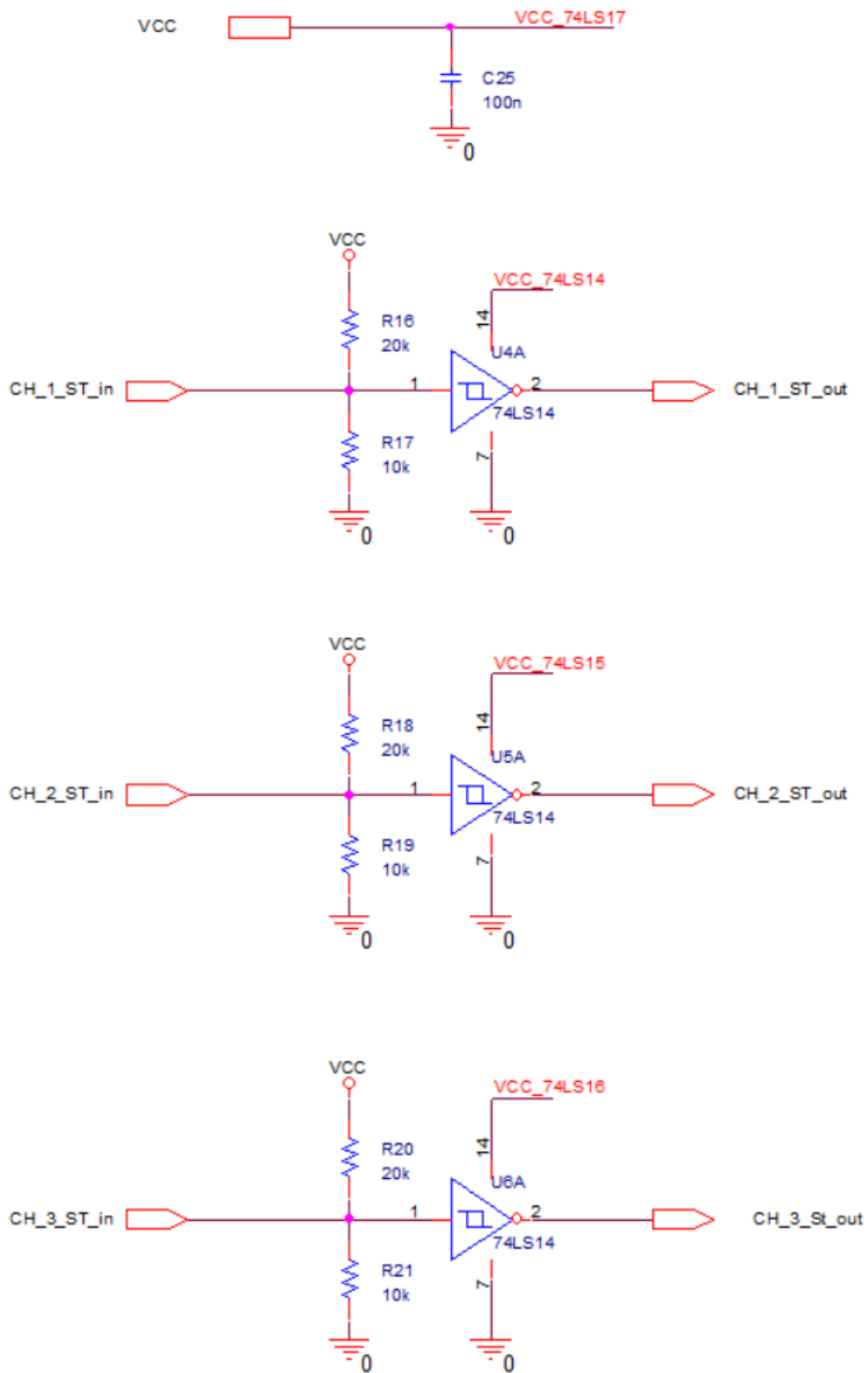


Figure 40: Schmitt trigger and voltage divider

The complete CPS circuit is shown in fig 41.

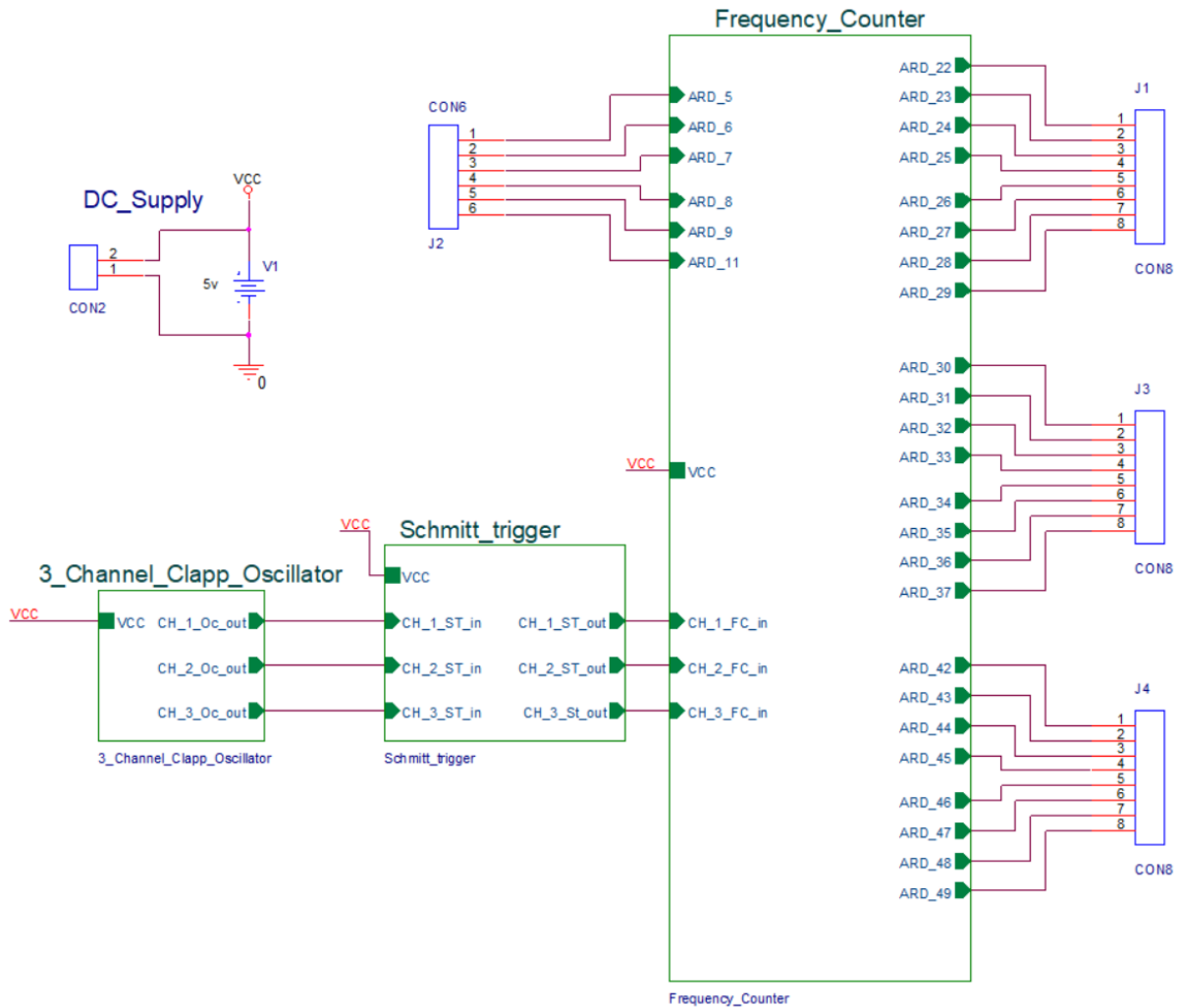


Figure 41: Complete CPS circuit

The final 3 channel BJT Clapp oscillator have a output frequency of 385.740 kHz measured with oscilloscope. A oscillation frequency stability of ± 1 Hz is achieved.

10.10 CPS PCB design

Requirement 1.12.1 from table 26, states that the PCB of the stator and rotor shall have the same mechanical interface as the potentiometer currently in use. The measurements of the potentiometers rotor and stator PCB were given as a mechanical blueprint from KDA. PCB coils shall be placed on the stator PCB and parts of the rotor PCB shall contain copper plating. An oscillator circuit shall be designed with the PCB coils on the stator PCB as the inductive element. To achieve the accuracy needed to meet the requirements, the design and placement of the coils is crucial. The stator depicted in figure 44 PCB is the part that shall contain the coils of the CPS. The coils must be placed so that when the copper plate segment of the rotor (figure 57), rotates over the stator, one or more of the

coils is/are affected. This means that the angle and spacing of the coils is of great importance. The area of the rotor containing copper, shall be shaped to suit the location of the coils and the angle of the coil's short side. To meet requirement 1.3.1, 1.14.1, 1.2.1, 1.10.1, 1.10.2 and 1.10.3 in table 26. There has to be a change in the coils magnetic field, due to the increasing or decreasing area of the coil covered and affected by the proximity of the copper present in the rotor. The change in frequency has to be as proportional as possible to the area covered by the rotor. A change in the magnetic field of a coil will cause a reduction of the inductance in that particular coil. The CPS team hopes this change in inductance will be proportional to the area of the coil which is covered by the rotor. The reduction in the coils inductance will lead to a change in the oscillation frequency of the oscillator having the that coil as the inductive element of the LC tank circuit. To achieve this, the increase/decrease of coil area covered and affected by the the copper plated section of the rotor, has to be linear to the rotational distance of the rotor. In an attempt to achieve this, the two edges of the copper plated section of the rotor moving perpendicular to the rotational direction of the rotor and the shortest sides of the coils is designed to be parallel.

10.10.1 Material selection in PCB

The dielectric material in the PCB is FR-4 laminate. Other material choices like polyimide materials were considered due to uncertainty regarding FR-4 material integrity and expansion ratio when exposed to temperatures stated in requirement 1.7.1 and 1.7.2 in table 26. After research of FR-4 used in this particular temperature range and discussion with KDA regarding PCB materials used in their current sensor, FR-4 was found to be a qualified choice of material in the CPS.

10.10.2 Pins and vias

Through the hole pins and vias, are holes that are drilled through the PCB and then coated with copper internally. The connection pins included in the coil part (Figure 42) have an electrical conductive connection to each layer of the stator PCB. Connection pins are placed on each end of the coil (Figure 43) to enable soldering of wires to the coil. Vias are similar to the through hole pins but they are completely filled with copper after they are drilled. Vias included in the coil part (42) connect the layers of the coil. Trough hole pins and vias are designed with the Padstack Editor software for Orcad.

Table 37: Pin and via dimensions

Part	Drill	Regular pad	Thermal pad	Anti pad	Soldermask
TH pin coil	C, 1.2 mm	2.0 mm	2.3 mm	2.3 mm	2.1 mm
Via coil	C, 0.4 mm	0.6 mm	0.82 mm	0.82 mm	0.65 mm
Via ground	C, 0.4 mm	0.8 mm	0 mm	0 mm	0 mm

When designing vias, the aspect ratio is the ratio between the board thickness and the diameter of the drill hole in the via

$$A_R = \frac{B_{TH}}{D_{DH}}, \quad (55)$$

where B_{TH} is board thickness, D_{DH} is diameter of via drill hole and A_R the aspect ratio. With board thickness of 1.99136 mm and a Via drill hole diameter of 0.4 mm the aspect ratio becomes

$$A_R = \frac{1.99136 \text{ mm}}{0.4 \text{ mm}} = 4.9784 \approx 5, \quad (56)$$

giving an aspect ratio of 5:1. A small aspect ratio will give a stronger and more tolerant via and also makes plating of the drill hole easier which helps ensuring the vias connection to traces and planes in the PCB.

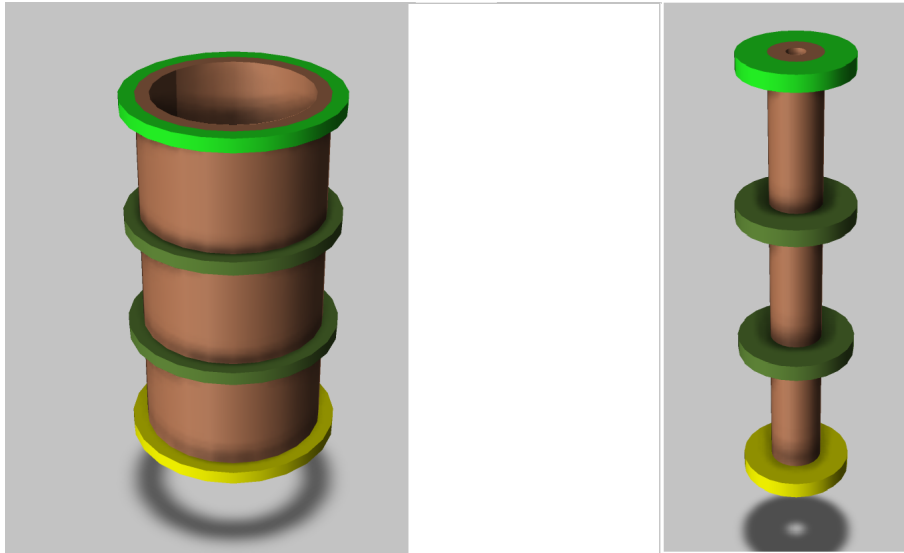


Figure 42: Trough hole pin diameter 1.2 mm (left) and Via diameter 0.4 mm (right)

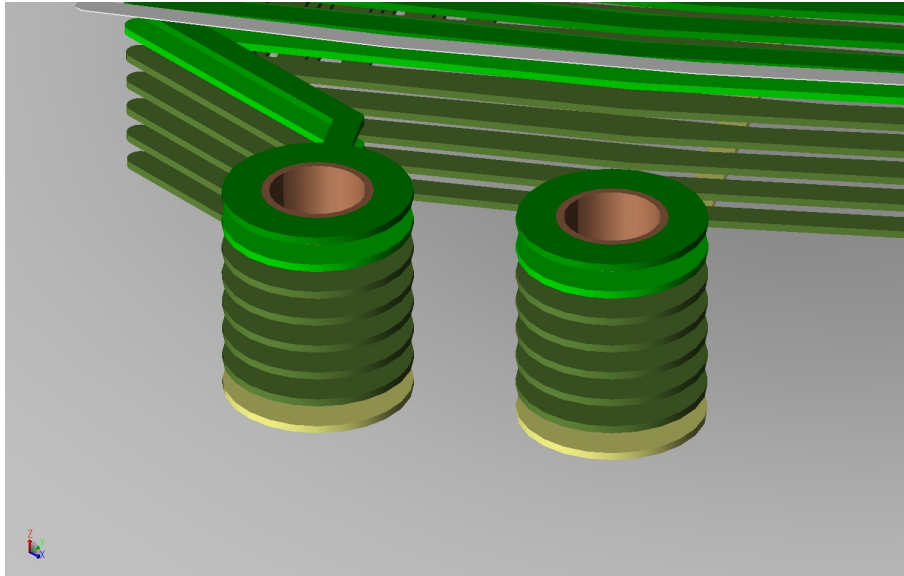


Figure 43: Coil connection through hole pin

10.10.3 Stator

The stator PCB is created with the same mechanical interface and dimensions stated in the blueprint from KDA. The only deviation from this are the areas regarding hole pins of the coil on the PCB. This deviation has been approved after conversation during status meeting with KDA. Blueprint of stator states that the thickness shall be 2 mm \pm 0.18 mm. The PCB is a 6 layer FR-4 laminate board. Dielectric layers consist of FR-4 laminate with thickness of 203.2 μ m. Conductor layers have a copper thickness of 71.12 μ m on all layers. Total thickness ends at 1.99136 mm which places the thickness within the acceptable range.

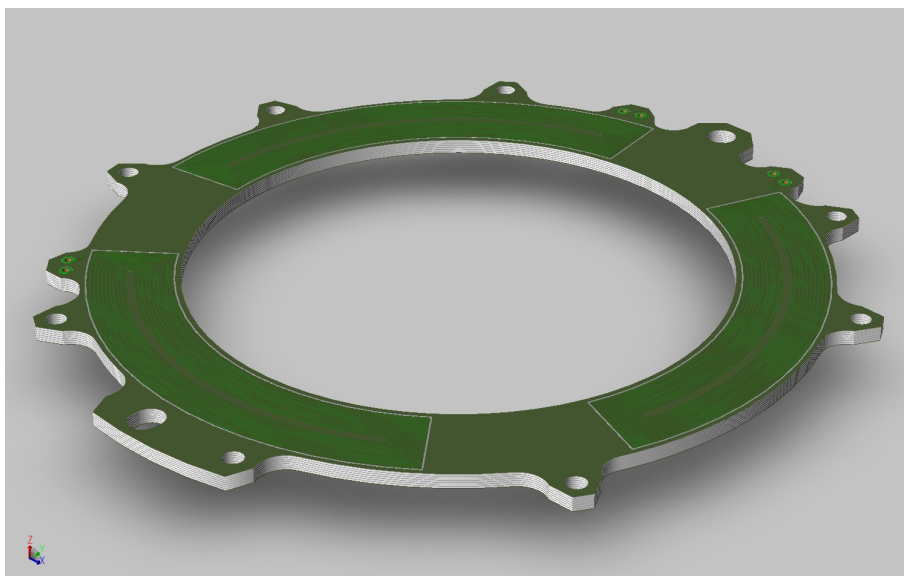


Figure 44: 3D model of stator PCB top side

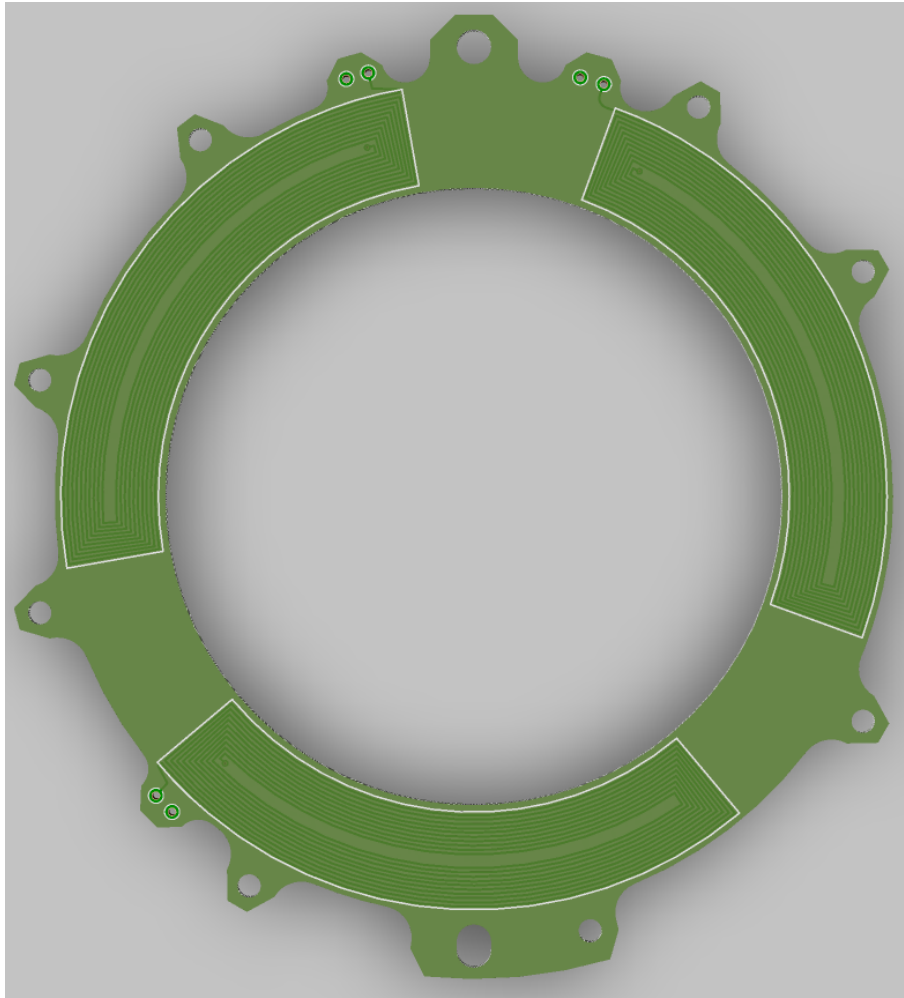


Figure 45: 3D model of stator PCB top side. Position of the coils marked with white silkscreen line

The coil parts cover a area of 45° in each direction form part centre (black cross in figure 46) which totals to 90° . The coils are placed (figure 46) with part centre 42.1 mm from centre of the stator. First coil is placed with part centre at 25° , the second at 145° and the third at 265° . This places the coil centres 120° apart with a spacing between coils of 30° . The coil connection pins (red square in figure 46) are located 51 mm from stator centre. Rotors radius is 48 mm which leads to coil connection pins being located outside of the rotor's radius. This contributes to making the magnetic field surrounding the coils as uniform as possible over the coil's entire length, regardless of rotational direction of the rotor. Placing the connection pins outside of the area containing the coils makes soldering wires easier.

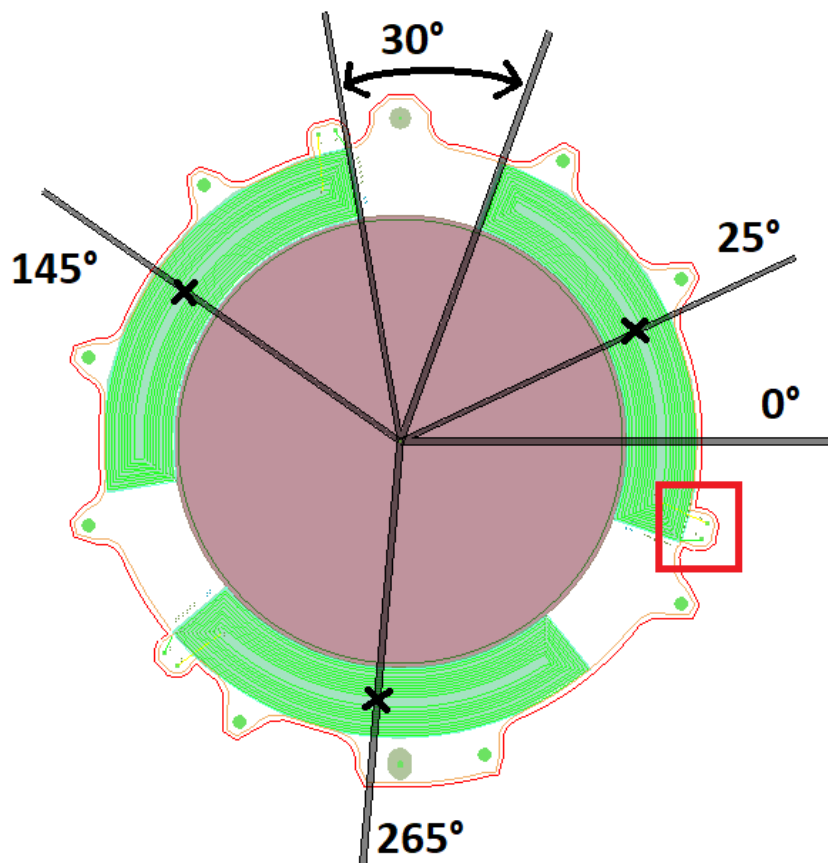


Figure 46: Angel of coil placement and spacing between coils. Coil part centre(black cross). Coil connection pin(red square)

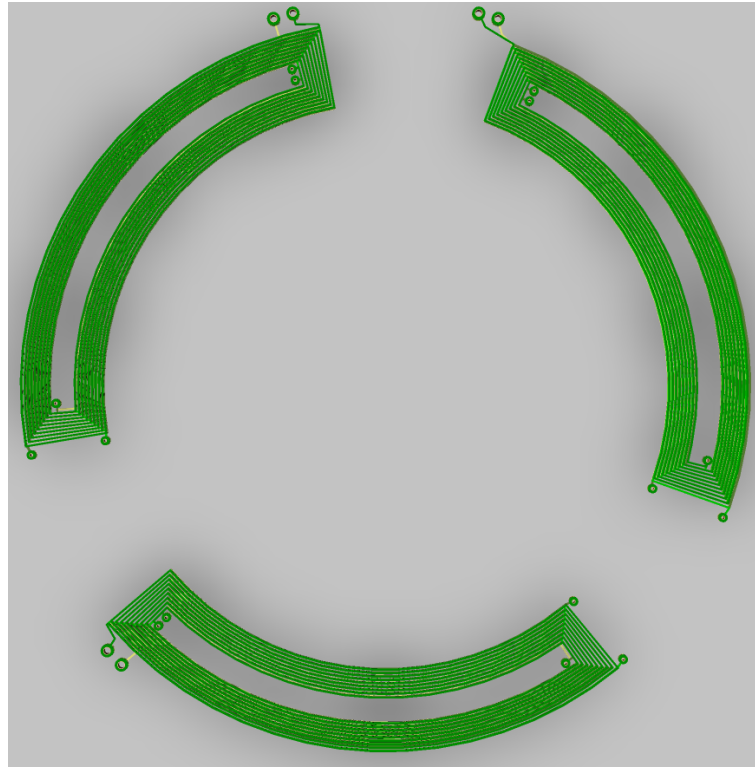


Figure 47: 3D coil placement in stator

10.10.4 Coil design

The shape, dimensions and layers are parameters that determine characteristics of a coil. The shape of the magnetic field is determined by the shape of the coil. Dimensions, number of layers determine the inductance of the coil. Coil design on PCB using OrCAD PCB editor is done by drawing PCB traces or by using the snap grid in the software to ensure that the spacing between the traces is as desired. The spacing in the grid can be adjusted in x and y direction to accurately place the traces. The other method available is to use coordinates (distance and angle from the centre of the drawing) to specify each point the trace shall go through. An Excel sheet is used to calculate the values. Arcs or straight traces can be chosen.

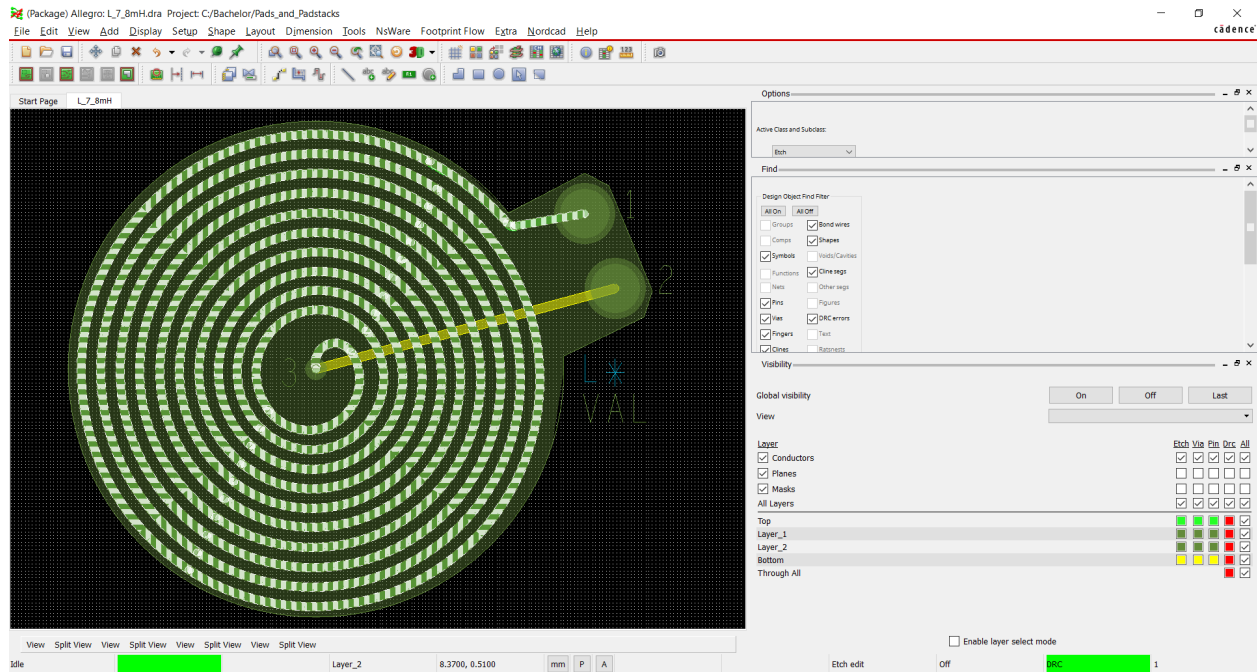


Figure 48: Coil design in Orcad PCB Editor

Many different shapes were designed and reviewed to see if they could be right for the team's design. The initial coils were single layer spiral coils as shown in figure 49. To increase the strength of the magnetic field, multiple layer is added to the coil part. Shown in figure 50 is a three layer spiral coil with through hole pins for connection of wire.

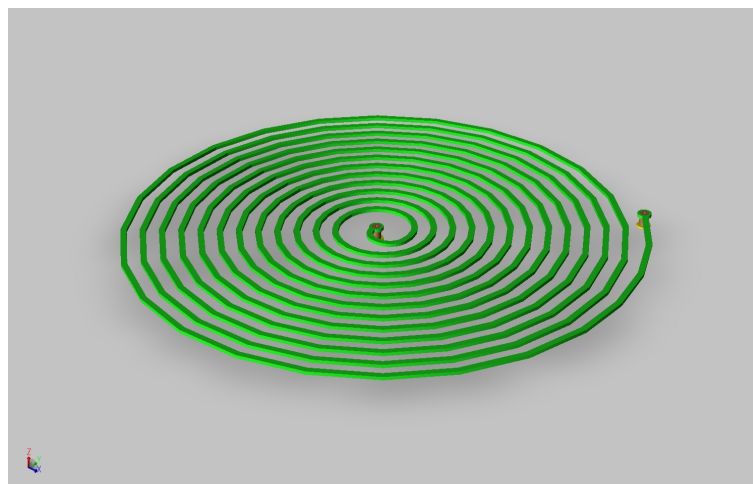


Figure 49: Coil 1: Single layer spiral coil

The initial coil design's grid spacing was used to create the coils. In the later designs, when the shape of the coils became more complex, the method of using a Excel sheet to calculate the distance and angle of the coordinates for the coil became more convenient and accurate than the method of adjusting the grid of the design.

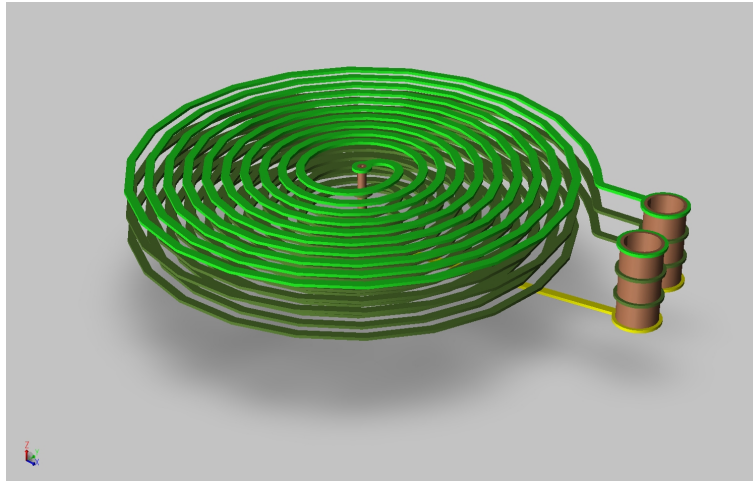


Figure 50: Coil 2: Three layer spiral coil

The shape of the coils that are to be placed within the stator, has to be adapted to fit into the spacing between the centre cutout and the design outline of the stator PCB. This spacing is of 11.4 mm. After reviewing the designs of the spiral shaped coils it was concluded that spiral shaped coils would not meet the criteria mentioned in section 10.10.

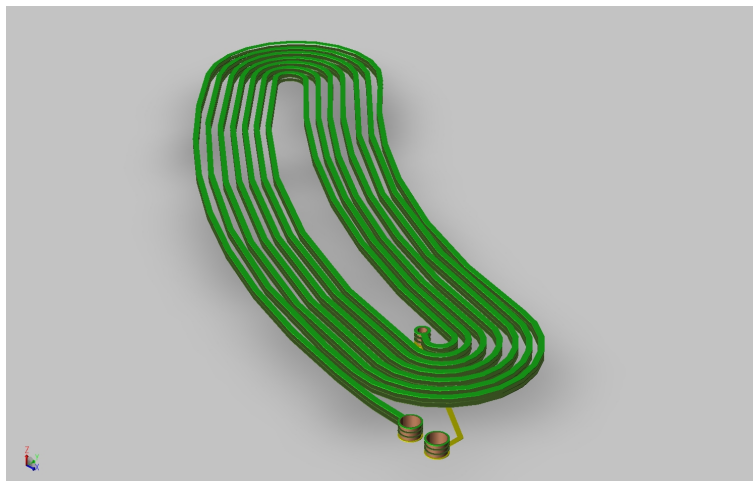


Figure 51: Coil 3: Three Layer curved rectangular coil

Coil design 3 has better characteristics than the two previous design in terms of proportional change in the inductance of the coil due to influence of the rotor. However, the short sides of the coil will possibly cause a change of inductance that is not coherent with the change of inductance over the rest of the coil. This is caused by the half circular ends of the coil not having the same relation between change of inductance and portion of the coil covered.



Figure 52: Coil 4: 7 Layer curved rectangular coil

Coil 4 was designed with an excel sheet design by the CPS team. The Sheet contain an input table (Figure 53) that automatically created the coordinate of distance and angle (Figure 53) used to construct the coil. The coil was designed to have the same relation between the rotational distance of the rotor and area covered over the entire length of the coil. The width of the coil is the same throughout the entire length. The only deviations are the spacing in the middle of the coil, the connection pins and the via in the centre.

Coil Design Sheet		
1. Defines width of pcb trace	1. Trace width [mm]	0.254
2. Defines minimum spacing between traces	2. Min trace spacing [mm]	0.2
3. Defines width of coil	3. Start angle of coil [deg]	340.5
4. Defines angular length of coil	4. End angle of coil [deg]	329.5
5. Defines start point of coil from center	5. Start distance [mm]	36.9
6. Defines end point of coil from center	6. End distance [mm]	47.5
7. Defines minimum trace spacing at inner diameter in degrees	7. Circumference of start distance	231.849538
8. Minimum trace spacing + trace width	8. Min trace spacing at inner diameter [deg]	0.6
9. Defines circumference	9. Spacing constant [mm]	0.5

Figure 53: Coil 4 excel input table

0 deg side															
Distance	Angle	Distance	Angle	N	Distance	Angle	Distance	Angle	N	Distance	Angle	Distance	Angle	N	
47.5	340.5	47.5	340.8	1	47.5	329	36.9	329	1	36.9	328.7	36.9	340.5	1	
47	340.5	47	340.8	2	47	328.4	37.4	328.4	2	37.4	328.1	37.4	341.1	2	
46.5	341.1	46.5	341.4	3	46.5	327.8	37.9	327.8	3	37.9	327.5	37.9	341.7	3	
46	341.7	46	342	4	46	327.2	38.4	327.2	4	38.4	326.9	38.4	342.3	4	
45.5	342.3	45.5	342.6	5	45.5	326.6	38.9	326.6	5	38.9	326.3	38.9	342.9	5	
45	342.9	45	343.2	6	45	326	39.4	326	6	39.4	325.7	39.4	343.5	6	
44.5	343.5	44.5	343.8	7	44.5	325.4	39.9	325.4	7	39.9	325.1	39.9	344.1	7	
44	344.1	44	344.4	8	44	324.8	40.4	324.8	8	40.4	324.5	40.4	344.7	8	
43.5	344.7	43.5	345	9	43.5	324.2	40.9	324.2	9	40.9	323.9	40.9	345.3	9	
43	345.3	43	345.6	10	43	323.6	41.4	323.6	10	41.4	323.3	41.4	345.9	10	
42.5	345.9	42.5	346.2	11	42.5	323	41.9	323	11	41.9	322.7	41.9	346.5	11	
42	346.5	42	346.8	12	42	322.4	42.4	322.4	12	42.4	322.1	42.4	347.1	12	

Figure 54: Coil 4 coordinate excel sheet

10.10.5 Final coil design

The objective of the design process of the final design was to create a coil which generated a magnetic field that was as uniform as possible over the entire length of the coil and had a

high Q-factor(Quality factor).The Q-factor of a coil is determined by

$$Q_U = \frac{\omega_0 L}{R_s}, \quad (57)$$

Where ω_0 is

$$\omega_0 = \frac{1}{\sqrt{LC}}, \quad (58)$$

L is the inductance of the coil and R_s is the series loss of the inductor and have the unit Ω . Serial connection of the 6 layers of the coil will result in a higher inductance since

$$L_{tot,serial} = L_1 + L_2 + L_3 + L_4 + L_5 + L_6, \quad (59)$$

while a parallel connection of the layer would result in

$$L_{tot,parallel} = \frac{1}{L_{tot}} = \frac{1}{L_1} + \frac{1}{L_2} + \frac{1}{L_3} + \frac{1}{L_4} + \frac{1}{L_5} + \frac{1}{L_6}. \quad (60)$$

This proves a serial connection causes a higher inductance. In terms of (58) increasing the inductance of the coil in relation to R_s leads to a increase in Q-factor. To further increase the inductance of the coil the current should flow in the same direction in all layers. Having the windings in every layer run in the same direction will ensure this. Having the current flow in the same direction on every layer increases the positive mutual inductance between the layers [23] which will add to the total inductance of the coil. In design of the final coil the excel sheet used in design of coil 4 was modified in order to produce a coil where the layers where connected in series. First layer of the final coil is identical to the first layer of coil 4. Coordinates describing the placement of the segments in the other 5 layers were calculated so that all windings run in the same direction. The last coil segment in the layer is connected to the next layer by a via. Specifications of the coil is stated in table 38.

Table 38: Specifications final coil design

Specification	Parameter	Value	Unit
Build method	Coupling of layers	Series	Method
Layers	Number of layers	6	Layers
Trace	Trace width	0.254	mm
Trace	Total length	1247.0366	mm
Trace	Length per layer	207.83	mm
Trace	Min.spacing between traces	0.2	mm
Windings	Number of windings	9	Windings
Diameter	Inner diameter of coil	3.7	mm
Diameter	Outer diameter of coil	11	mm
Trace	Total length	1247.0366	mm

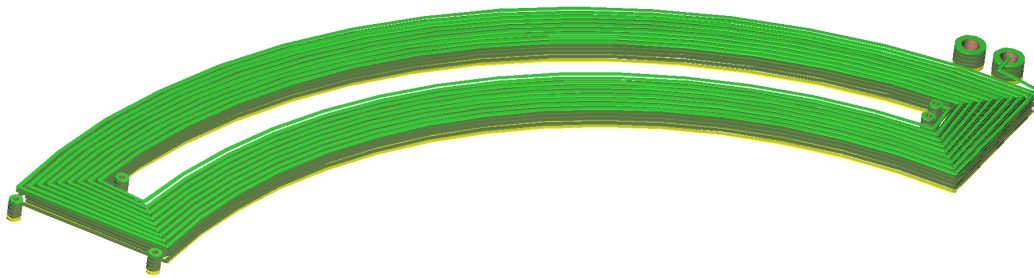


Figure 55: Final coil design, layers connected in series with vias

From test24.3 the inductance of the final coil is found to be 110 μ H. The oscillator has a oscillation frequency of $f = 390.800$ kHz. Resulting in ω_0

$$\omega_0 = 2\pi f, \tag{61}$$

$$\omega_0 = 2\pi \cdot 390.800 = 2455468. \tag{62}$$

The resistance of the coil is measured to be 16Ω giving the coil a Q-factor of

$$Q = \frac{2455468 \cdot 110 \text{ }\mu\text{H}}{16 \text{ }\Omega} = 16.88. \tag{63}$$

10.10.6 Rotor

The rotor is a 6 layer PCB containing 6 planes of copper which is grounded by the connection to the screws going through the mounting holes shown in figure 60. Conductor layers have a copper thickness of $71.12 \text{ }\mu\text{m}$. Dielectric layers contain FR-4 laminate with thickness of $203.2 \text{ }\mu\text{m}$. Copper planes covers the rotor from 0° to 210° on layers 1 to 7. Grounding of the 8 mounting holes is ensured by 8 vias (Figure 60) surrounding each mounting hole. The grounding vias have a tin-plate finish on the top layer to prevent oxidation of the exposed copper and to ensure contact between the screw and the ground layers.

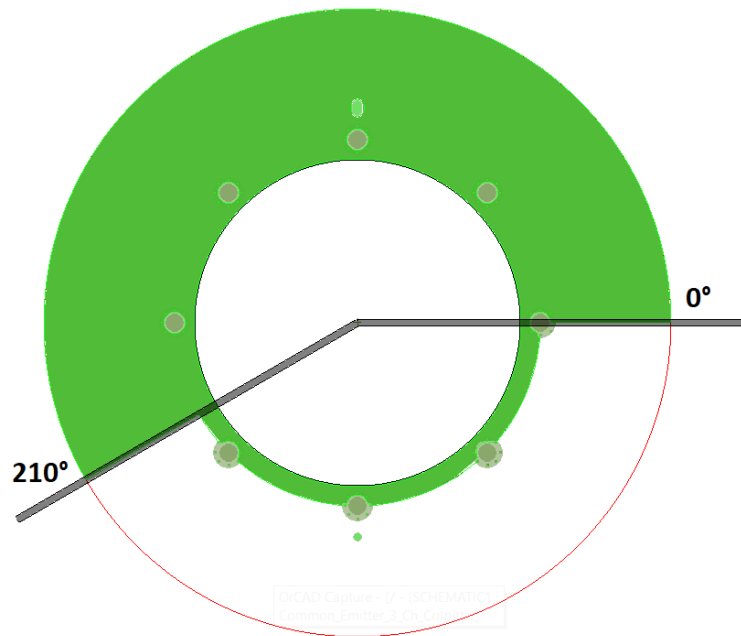


Figure 56: Angle of copper layer in rotor

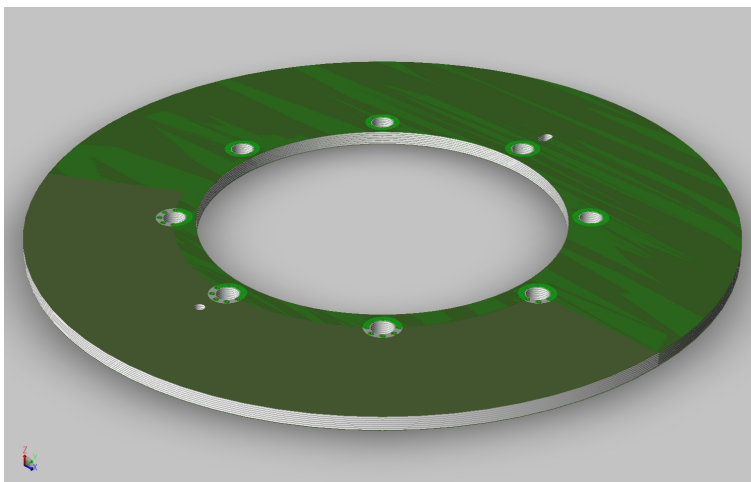


Figure 57: Side view of rotor,

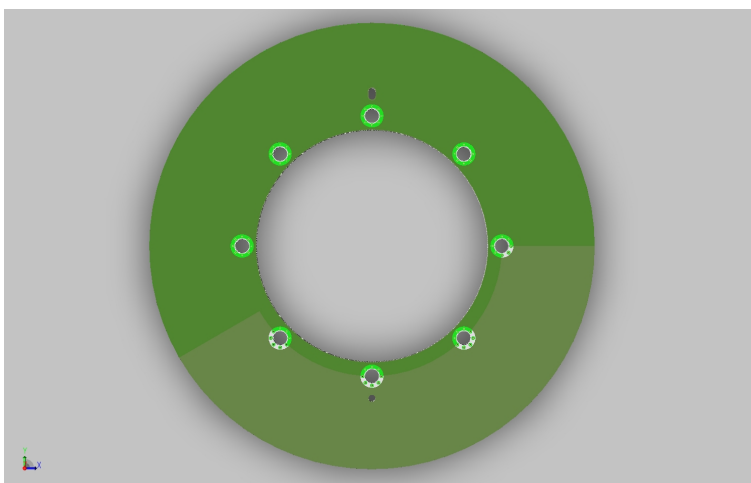


Figure 58: Top view of rotor

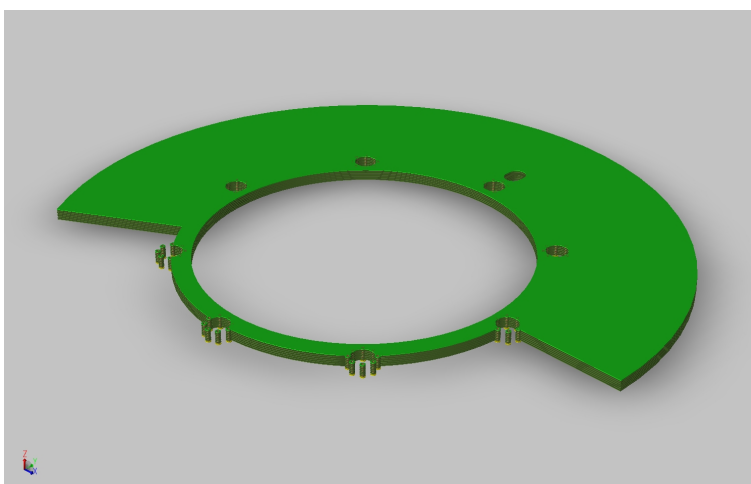


Figure 59: Copper layers in rotor

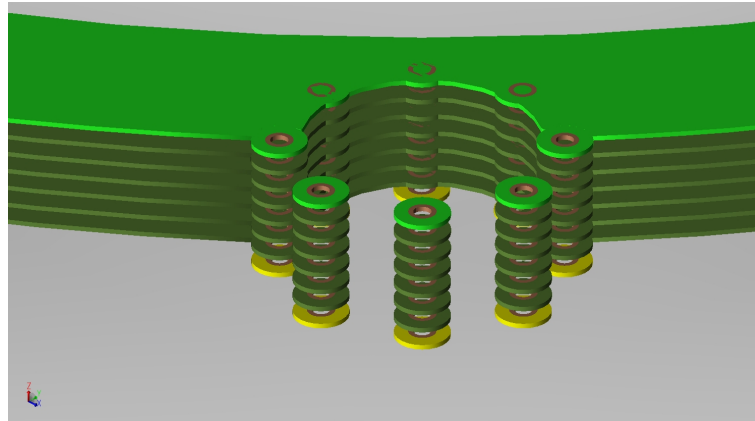


Figure 60: Vias surrounding mounting hole to ensure grounding

10.10.7 PCB ordering

Review and research of multiple manufacturers lead to All PCB.com being chosen as the PCB manufacturer. The main criteria for choosing All PCB.com was price and delivery time. Both the Serial and parallel coil were ordered. Some changes was done to the designs before ordering. The number of layers were reduced form 8 to 6. This was due to extended production time and a large increase in cost of the 8 layer PCB. The order was sent to the manufacturer 04.04.19. 2 types of rotors were ordered. A 6 layer and a 2 layer. The 6 layer rotor has a copper thickness of 35 μ m and the 2 layer rotor has copper thickness of 70 μ m. The shipment with the PCB's arrived 24.04.19. Due to an error in the production one the stator PCB's had to be redone. This delayed the delivery by 3 days.

10.11 Stator and rotor PCB



Figure 61: Finished Stator PCB

The Inductance of the coils were measured and found to be $110\mu\text{H}$.

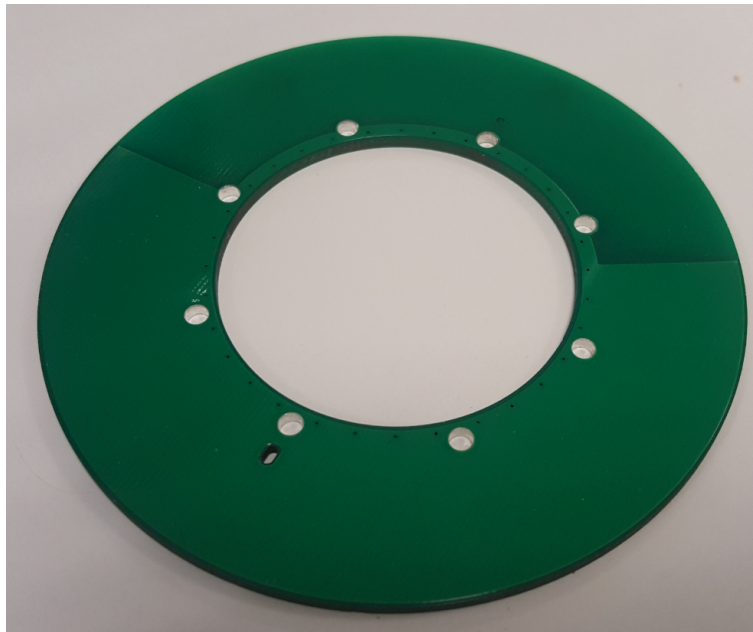


Figure 62: Finished rotor PCB

11 Mechanical concepts

11.1 Document history

Table 39: Mechanical concepts document history

Version	Date	Author	Description
1.0.0	13.02.2019	HS	Document created, added introduction and implementation of mechanical parts
1.1.0	14.02.2019	HS	Added section 12.3.3 to 12.3.7
1.3.0	08.03.2019	HS	Began documentation of concept 3 and 4. Added section 12.4 to 12.4.3
1.4.0	11.03.2019	HS	Added section 12.5 to 12.5.6
1.5.0	23.03.2019	HS	12.6.
1.5.1	24.03.2019	JSS	Proofread
1.6.0	14.05.2019	HS	Added sections 12.7 to 12.8.3
1.7.0	15.05.2019	HS	Added section 13
1.7.1	17.05.2019	MBC	Proofreading and corrections.
1.7.2	22.05.2019	CPS team	Proofreading and corrections.

11.2 Introduction

This chapter will cover iterations and increments of concepts through the design process. Documentation covering the design choices in the concepts will be added throughout the project and in more detail as a concept has been chosen. This includes topics such as bearings, lubricants, springs, shafts and dowel pins. The most crucial requirements the concepts revolves around are precision and temperature. (1.3.1 TR, 1.4.1 T, 1.7.1 T and 1.7.2 T) System requirements are found in section: 7.4. Each increment or concept will have a design review. Tools used in the design process are SolidWorks CAD program and a caliper to measure existing parts.

11.3 Mechanical concept 1, KDA prototype

Concept 1 is a prototype given to the team by KDA. This is a concept they worked on several years ago. The team received this miniature model to get a better understanding of how the system should work. The process of creating the design of the needed parts was mainly done in SolidWorks and through measurements of already existing parts.

11.3.1 Implementation of mechanical parts

The prototype the team received was an assembly consisting of three PCB's with wires, metal housings for the PCB's and a copper rotor. To test this as a complete system, the team decided to mount the parts on an electrical motor, but to do so a bracket and a few parts for a shaft was needed. The created parts are 3D-printed and nuts and screws are bought.

11.3.2 List of parts

- Longs Stepper Motor, model: 17HS3404N
- PCB, stator
- Copper-rotor
- Housing for PCB
- 3D-printed bracket (P.1.0.1)
- 3D-printed cap (P.1.0.2)
- 8 X M1.5 flat head screw
- 4 X M3 screws
- 4 X M2.5 nut
- 4 X M4 nut

- 2 X m5 nut
- 4 X 0,5mm shim
- M5 bolt with half section threaded rod
- Coupling

11.3.3 Bracket, P.1.0.1

Part P.1.0.1 is a bracket mounted directly on the electrical motor. The intention of the part is to give correct spacing between the rotor and stator as well as serving as cup for the PCB housing.

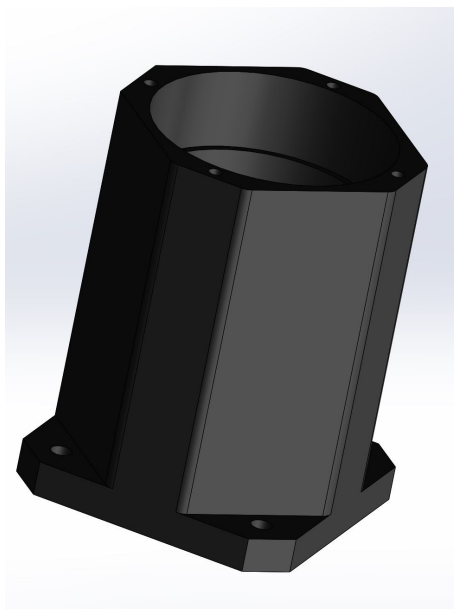


Figure 63: Bracket, KDA prototype, P.1.0.1.1

The holes in the bottom of the part are concentric with the holes on the stepper motor for precise fixturing.

11.3.4 Cap, P.1.0.2

Part P.1.0.2 is the cap for the bracket. The intention of the part is to hold the PCB housing steady. The part will be held in place and mounted on the bracket using four flat head screws.

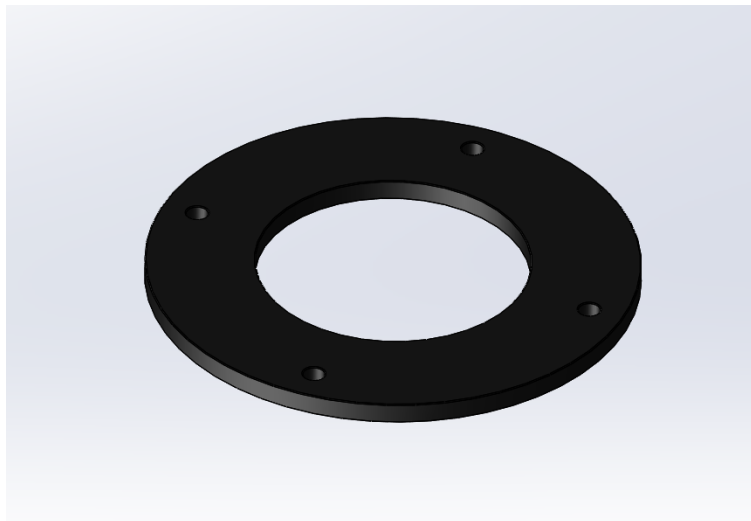


Figure 64: Cap, KDA prototype, P.1.0.2

11.3.5 Assembly of KDA prototype, A.3.0.0

The assembly is created in SolidWorks, and is how the actual prototype is set up.

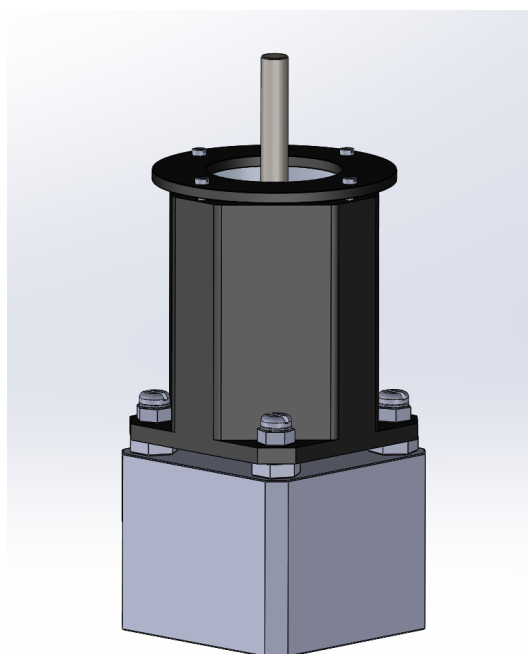


Figure 65: Assembly of KDA prototype, A.1.0.0

11.3.6 Step by step assembly, A.1.0.0

1. Press the PCB housing in the bracket.
2. Fix the PCB to the housing using four flat head M1,5 screws.
3. Set the stepper motor on a flat surface
4. Mount the custom coupling on the shaft of the stepper motor
5. Mount one M5 nut on the threaded rod of the shaft.
6. Put one M4 nut over each hole on the stepper motor
7. Place the bracket on top of the nuts and align the holes on the bracket with the holes of the stepper motor.
8. Guide the wires from the PCB out from the bracket through the holes between the stepper motor and the bracket.
9. Fix the bracket to the motor by using four m3 screws with a 0,5mm shim between the screw head and bracket.
10. Place the rotor on the shaft and let it rest on the M5 nut. Place a new M5 nut above the rotor and tighten.
11. Place the cap over the PCB housing and align with the holes of the bracket. Fix the cap using four M1.5 flat head screws.

11.3.7 Design review, DR-A.1.0.0

After doing tests on the prototype, the team got a better understanding of what the system should look like, and which components should be included. Because the first prototype is a miniature and is an incomplete system, the team was not able to test all requirements. From the mechanical aspect of the prototype, the biggest problem was getting a precise rotation on the rotor. One or more bearings around a shaft would help obtain a straighter rotational axis. For the next increment the goal is to have the correct scale of a model with a functional, concentric rotating, precise system. The system shall also have the correct mechanical interface.

11.4 Mechanical concept 2

In this section the CPS-team will explain the ideas behind the design of concept 2. By studying the prototype given by KDA and doing extensive research on contactless systems, the team started looking at different concepts. This concept is meant to be a part of the design process and an increment of prototype 1 to get a better understanding of spacings and parts needed to be included in the system and in the test station. This means that choice of materials and alloying are not calculated, but are considered. The concept is a full scale CAD model.

11.4.1 Rotor and stator

The current system used by KDA consists of a rotor and two stators. The second stator is included for the system to be considered redundant. The first iteration of the concept will focus on having one stator and a rotor just to achieve a functional system ready for testing.

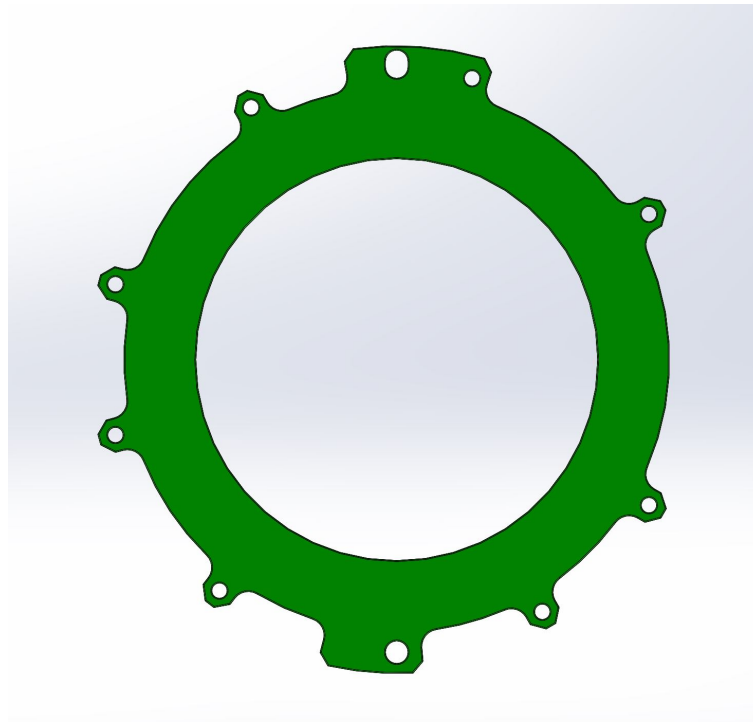


Figure 66: 3D model of stator, concept 2

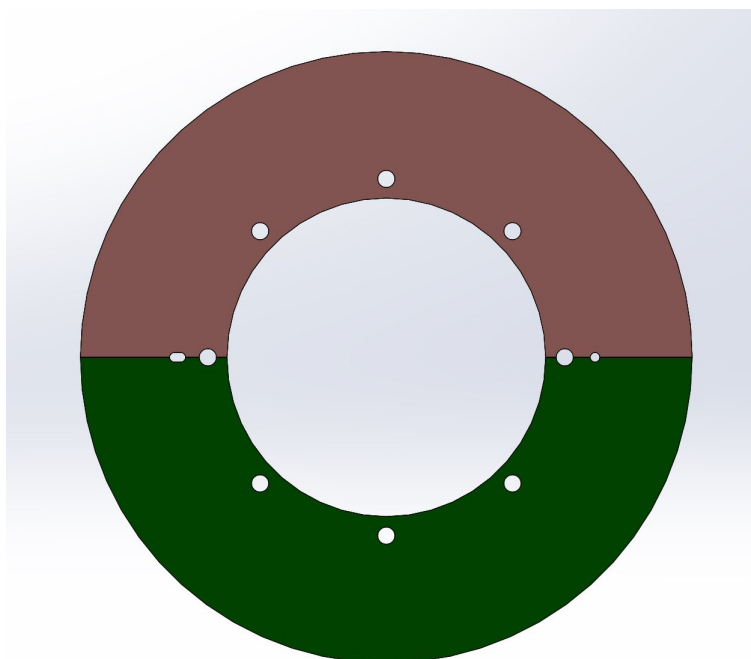


Figure 67: 3D model of rotor, concept 2

According to the requirements given by KDA, the mechanical interface must be the same as the current system in use. This means the location of the holes on both stator and rotor have to be designed identical to the potentiometer and current rotor. To ensure this, the team designed a stator and rotor identical to KDA's in SolidWorks. The team was recommended by KDA to use PCB in both the stator and rotor and cover half of the rotor in copper because of its conductive properties. More information on PCB materials can be found in section 10.10.1

11.4.2 Housing

Precision is a key factor for the sensor the CPS team is designing. To ensure that the stator and rotor are mounted in a precise position dowel pins will be used. Dowel pins allows extremely high tolerances for concentricity and centricity and are used to guide one part over another. The pin itself has strict tolerances in straightness and fitting to the hole it is fitted to. To ensure a precise rotary axis, bearings shall be used around a shaft. The outer diameter of the bearings will decide the design of the center hole. A provisional bearing has been added to get an idea of how the system can look.

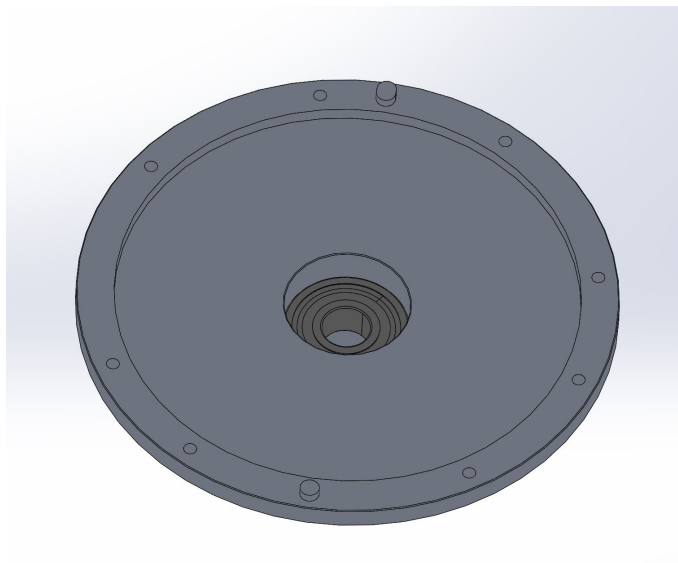


Figure 68: Housing top view

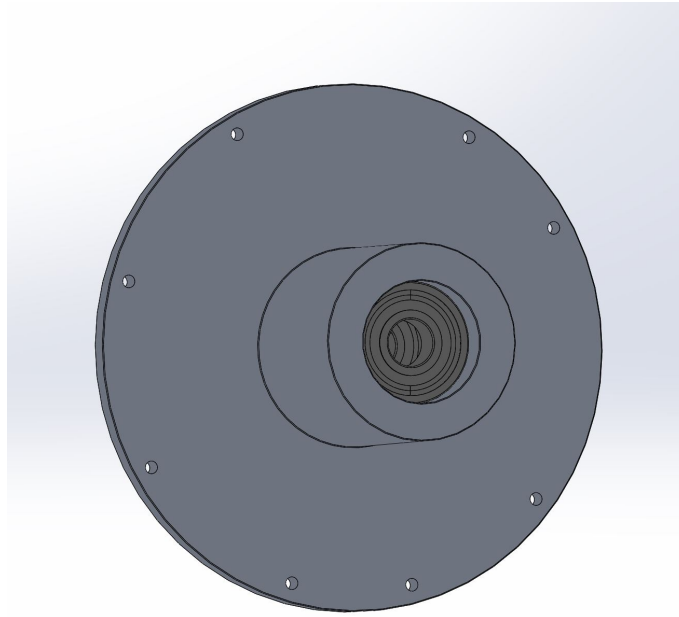


Figure 69: Housing bottom view

11.4.3 Shaft

The intention of the shaft is to have a rotor rotating over the stator. The tolerances between bearings, housing and the shaft have to be strict to fulfil and measure requirements, 1.3.1 TR, 1.4.1 T. The shaft will be preloaded on the inner ring of the bearing and follow its rotation. The outer ring of the bearing will be held in place by the housing. Bearing information and calculations will be added once a bearing has been chosen. The shaft will be directly mounted to an electric motor by using a bellow coupling. The holes on the shaft are made to mount the rotor fixture. The small holes are mainly made to make sure the position of the rotor mount is precise, while the center hole will hold it in place with a M6 bolt. The two small holes may be replaced by dowel pins, this will be discussed in the design review.

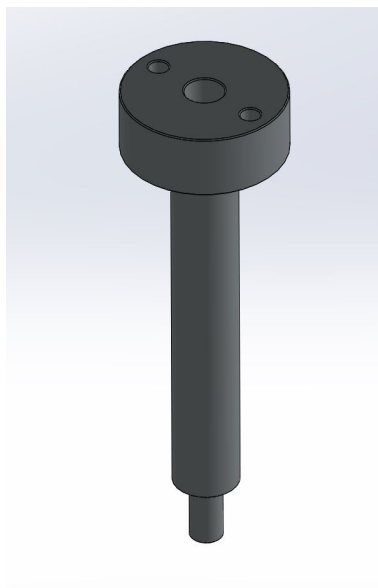


Figure 70: Shaft, concept 2

11.4.4 Rotor fixture

The rotor fixture is a part that will sit between the shaft and the rotor. The intention of the part is to fix the rotor in a precise concentric position over the shaft and stator. The hole locations on the part match the holes on the rotor and have the same dimension as the rotor currently in use by KDA. Dowel pins will also be used to guide the rotor in the correct position over the rotor fixture.

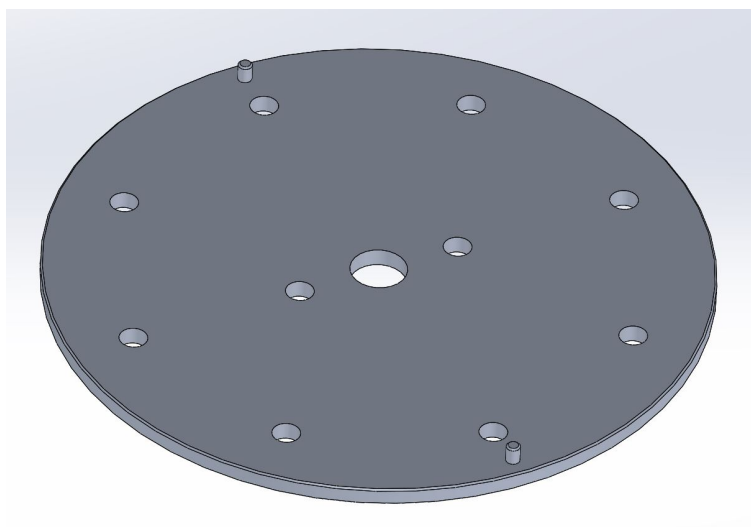


Figure 71: Rotor fixture, concept 2

11.4.5 Assembly, A.2.0.0

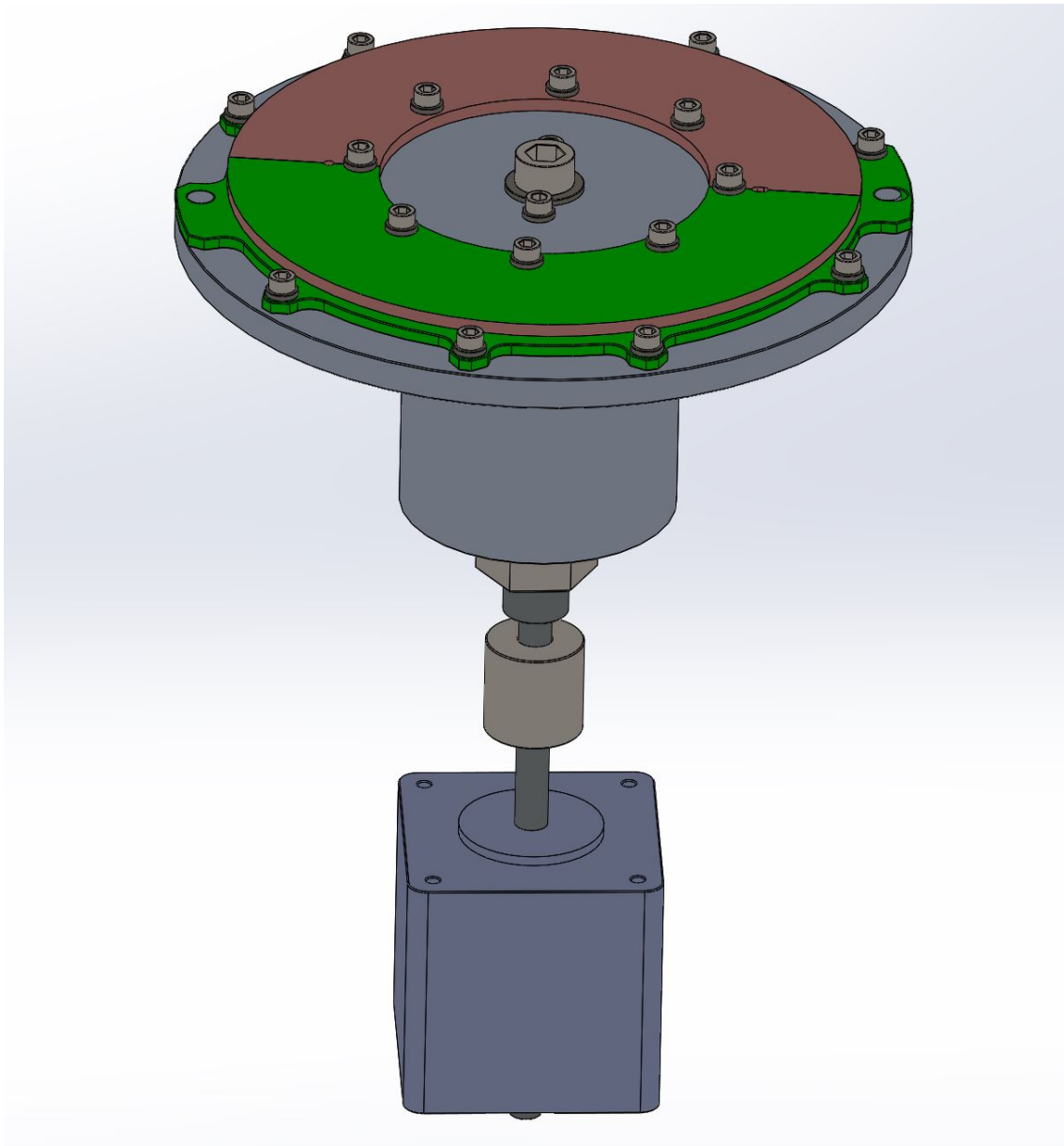


Figure 72: Assembly of concept 2, A.2.0.0

11.4.6 Cross section, A.2.0.0

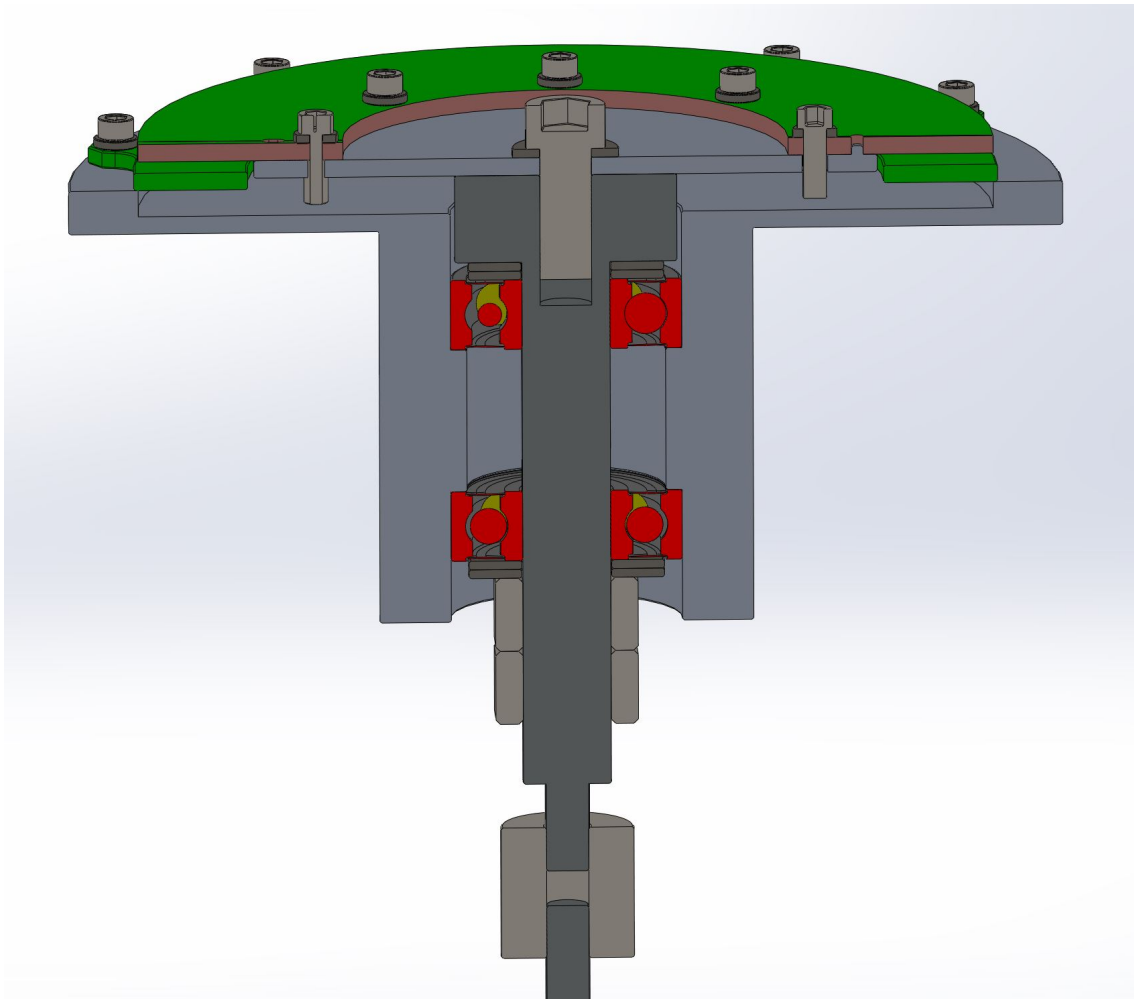


Figure 73: Cross section, A.2.0.0

11.4.7 List of parts

- Stator
- Rotor
- Housing
- 2x, Ball bearing, provisional
- Rotor Fixture
- Stepper Motor
- Shaft
- 4x, 1mm shim
- 2x, M10 nuts
- 18x, M3 bolts with shims
- M6 bolt, with shim

11.4.8 Design review, DR-A.2.0.0

Review of the assembly was done in collaboration with KDA. The goal of a precise rotary concept was met by using two provisional ball bearings and a shaft. A few changes needed to be made to go forward with the ideas behind the concept.

1. Fixture for a second stator considering the redundancy requirement. (1.8.1 AR)
2. Fixture for the stepper motor.
3. Using spring between the nut and the bearing to deal with the thermal expansion was discussed.
4. Study bearings, lubrication and preloading. (Study of bearings, section: 12)
5. Find a coupling that fits the stepper motor from KDA.
6. Change the design of the shaft for milling.
7. Redesign the housing for milling.
8. Design the housing so it can be fixed to a flat surface. Preferred to a table with hole locations 50mm X 50mm.

11.5 Mechanical concept 3

This section will cover a full scale SolidWorks CAD model of concept 3. The changes discussed in the DR-A.2.0.0 (Section: 11.4.8) have been made. This is an increment and iteration from Concept 2.

11.5.1 Assembly concept 3, A.3.0.0

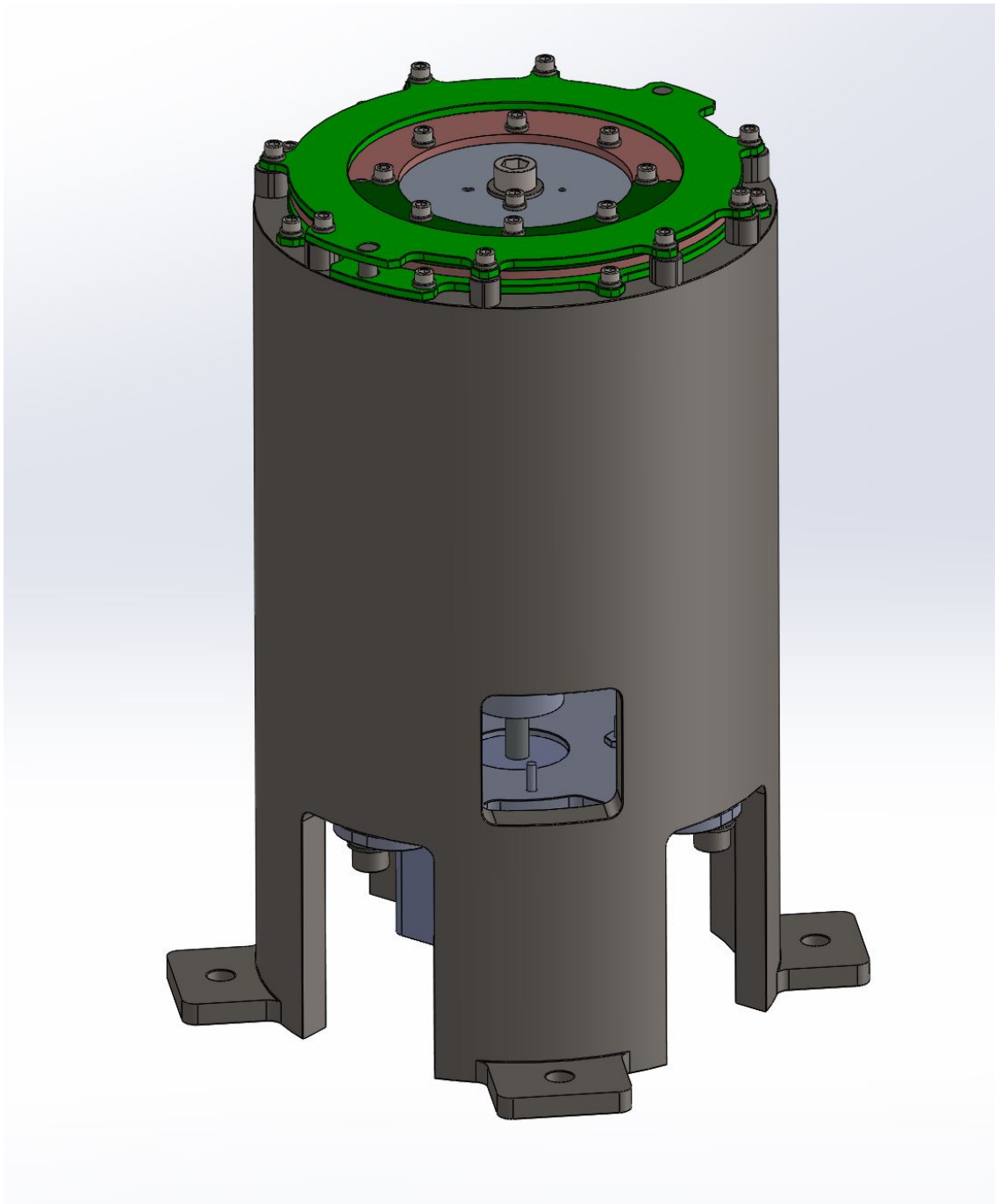


Figure 74: Assembly of concept 3, A.3.0.0

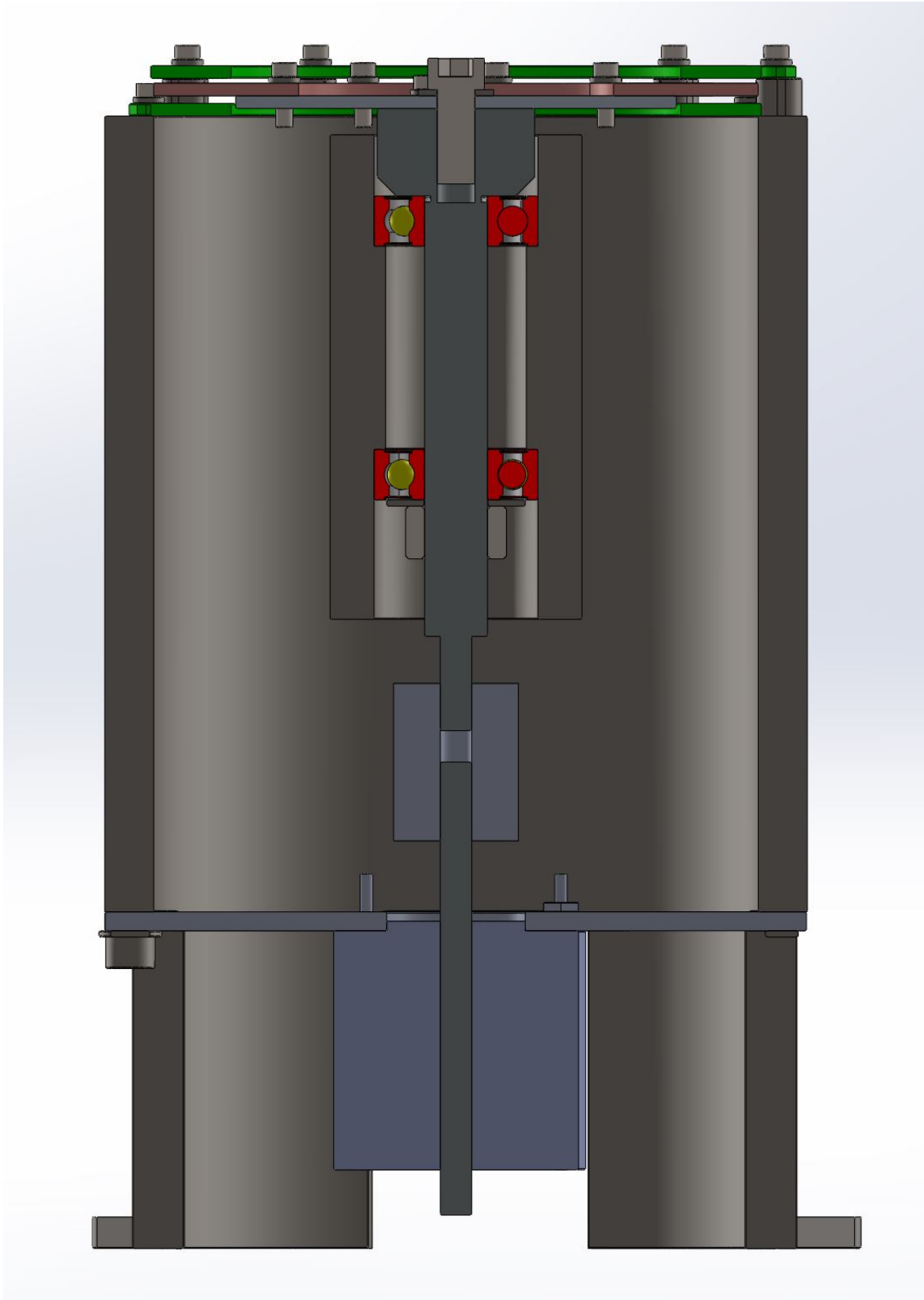


Figure 75: Cross section, A.3.0.0

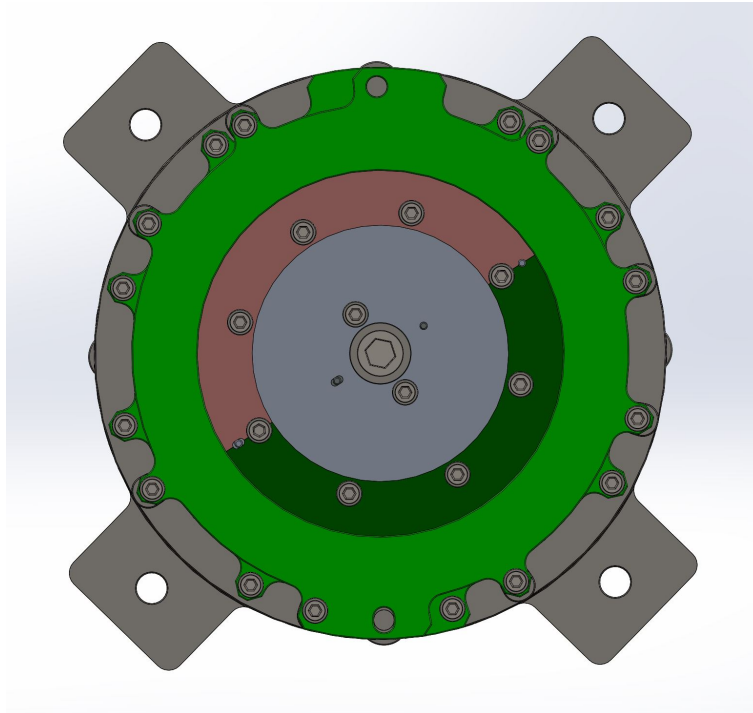


Figure 76: Top view, A.3.0.0

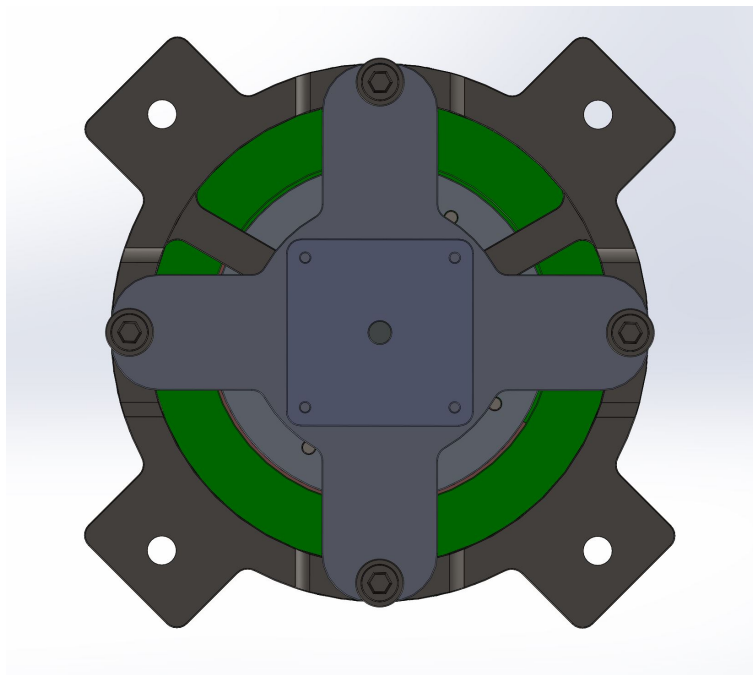


Figure 77: Bottom view, A.3.0.0

11.5.2 List of parts

- 2 x Stator
- Rotor
- Housing, P-C3-3.0.1
- 2 x deep groove single row ball bearing
- Rotor Fixture, P-C3-3.0.3
- Stepper Motor
- Fixture for electrical motor, P-C3-3.0.3
- Shaft, P-C3-3.0.4
- 1mm shim
- Disc spring
- M10 nut
- 24 x M3 bolts with shims
- M6 bolt, with shim
- 4 x M4 bolt with shims
- 4 x M2 nut with shims

11.5.3 Housing, P-C3-3.0.1

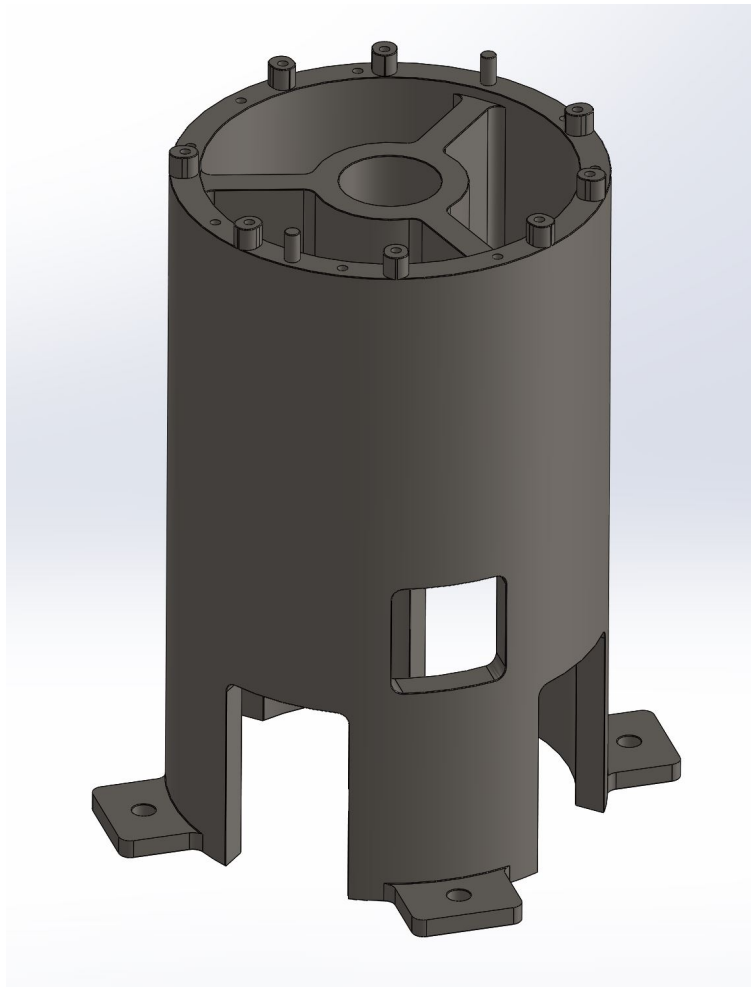


Figure 78: Housing concept 3, P-C3-3.0.1

The changes made to the housing come from the DR-A.2.0.0 section 11.4.8. The reason behind the bucket shape is mainly to have a way to mount the stepper motor to the rest of the system, and to make the milling cheaper. Due to the requirements of position and repeatability it is optimal to have as few parts as possible. This leads to a larger and complex part. Splitting the housing into smaller parts would be more optimal for cost and production, but would compromise precision, due to tolerances between parts. The top part of the housing has fixtures for two stators. The second set of holes are mirrored around the top plane of the part. It is made so that every bolt is reachable without removing any of the stators. The square hole on the side of the housing is made to tighten the coupling between the shaft and the stepper motor during installation. The ears are made for mounting the station to a surface (50mm x 50mm).

11.5.4 Rotor fixture

The fixture for the rotor is similar to concept 2 (section 11.4). The only changes are the holes for the dowel pins on the shaft for better concentric guidance.

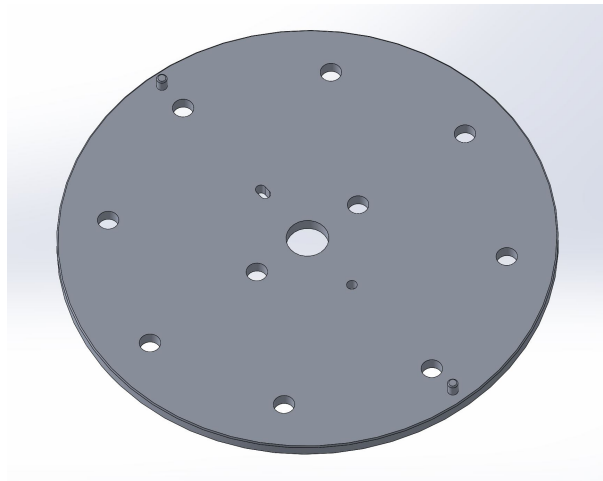


Figure 79: Rotor fixture concept 3, P-C3-3.0.3

11.5.5 Shaft, P-C3-3.0.4

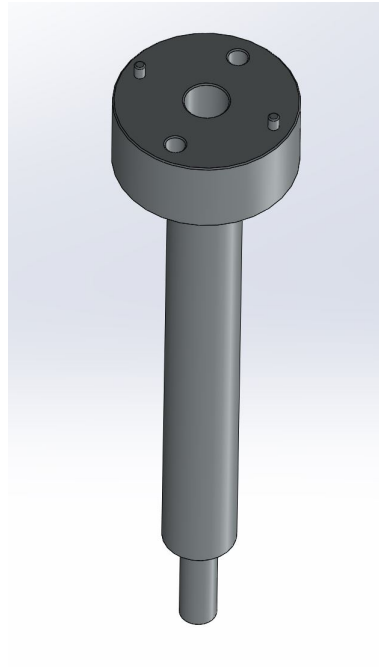


Figure 80: Shaft concept 3, P-C3-3.0.4

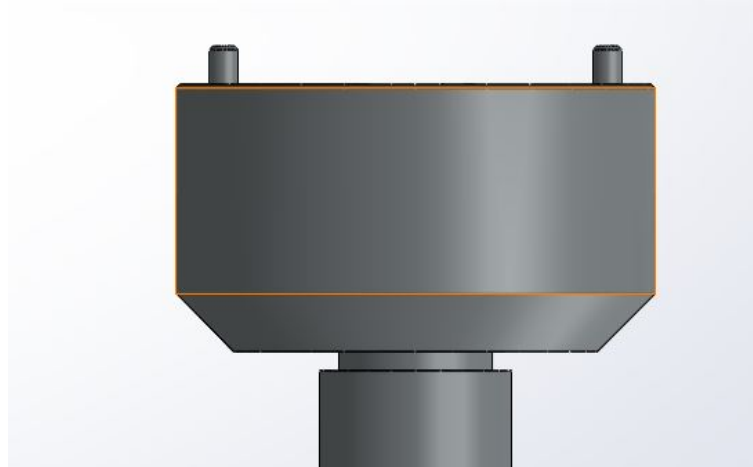


Figure 81: Shaft top section side view, P-C3-3.0.4

A few changes were made to the shaft from concept 2. Dowel pins were designed for guiding the rotor fixture in the correct position. The small cut in the shaft was made for milling. Due to the low load the system inflicts, the bearings have to be preloaded around the shaft with a spring, also dealing with the thermal expansion and shrinking. The preload has to be high enough to make sure that the shaft do not slip on the inner race of the bearing [51].

11.5.6 Fixture for electrical motor, P-C3-3.0.3



Figure 82: Fixture for electrical motor, P-C3-3.0.4

The intention of the part is to mount the motor to the part P-C3-3.0.1. The large hole in the center of the part is made for the shaft of the electrical motor and for concentricity. The four small holes are for mounting the part to the motor. This will be done with a M2 nut and a shim. The four larger holes are for fixturing to the part P-C3-3.0.1. It will be mounted with four bolts with shims, M4.

11.5.7 Design review, DR-A.3.0.0

The concept covers the basic requirements of precision by having bearings around the shaft. Depending on the lubricant, spring, preload of bearings and choice of materials, the requirements dealing with temperature can be met. Documentation covering these topics will be added. It was discussed to have the system concealed under a lid.

11.6 Mechanical concept 4

This section will cover a full scale SolidWorks CAD model of concept 4. Some parts from concept 3 are included in concept 4 and will not be further discussed in the sections below.

11.6.1 Assembly concept 4, A.4.0.0

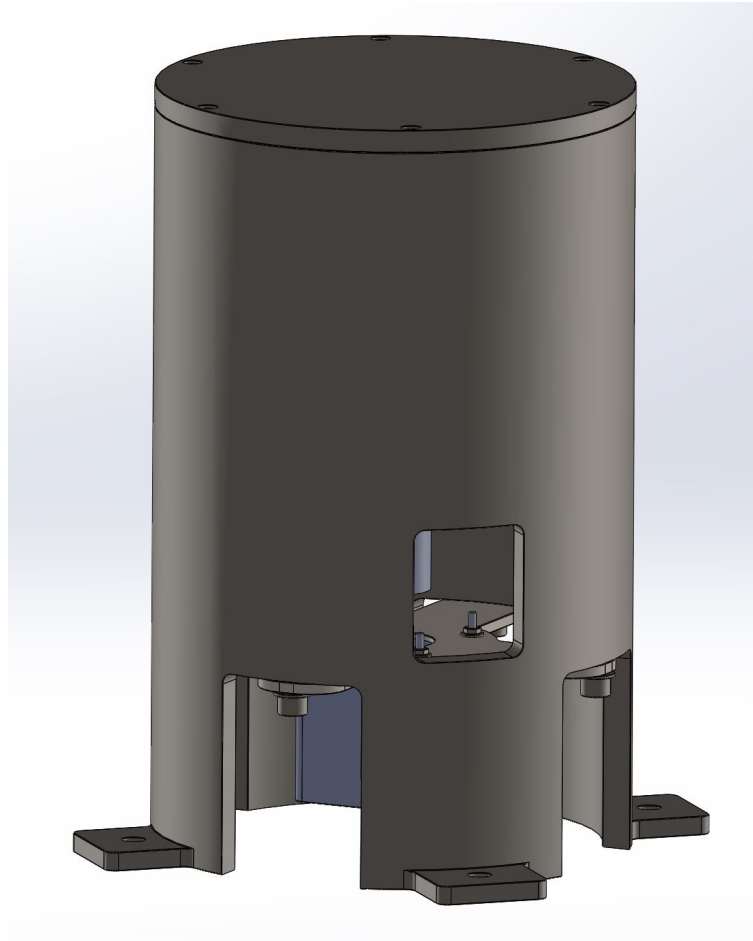


Figure 83: Assembly of concept 4, A.4.0.0

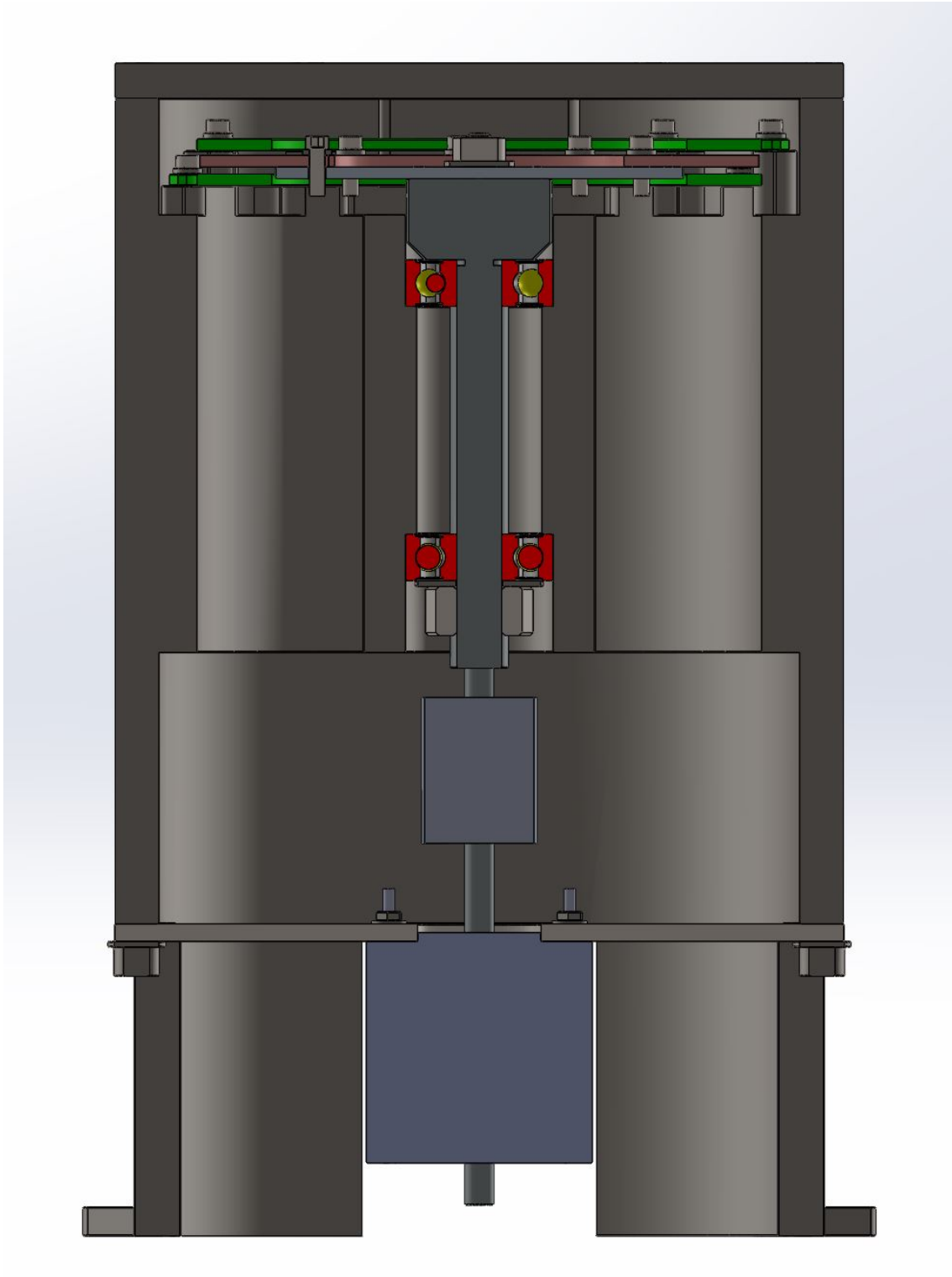


Figure 84: Cross section, A.4.0.0

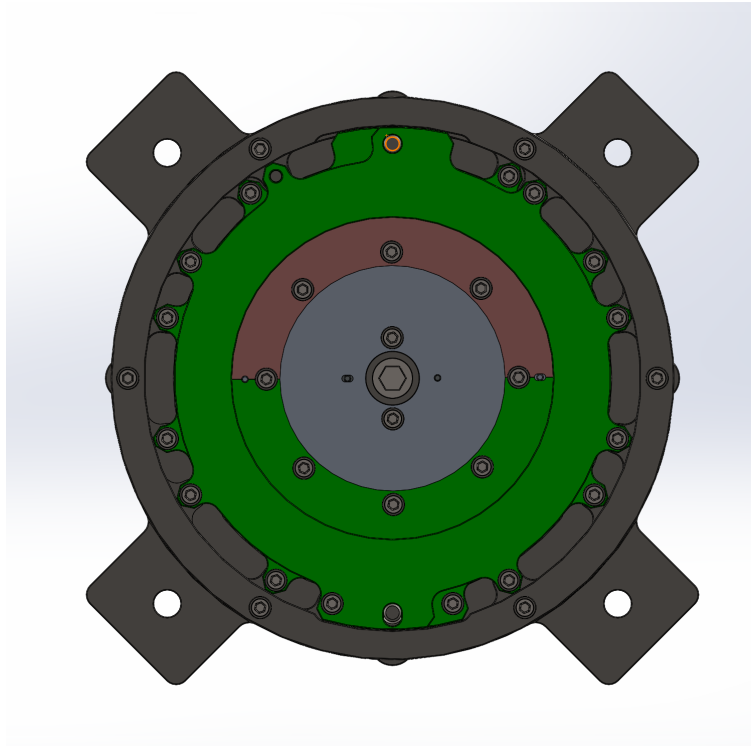


Figure 85: Top view, A.4.0.0

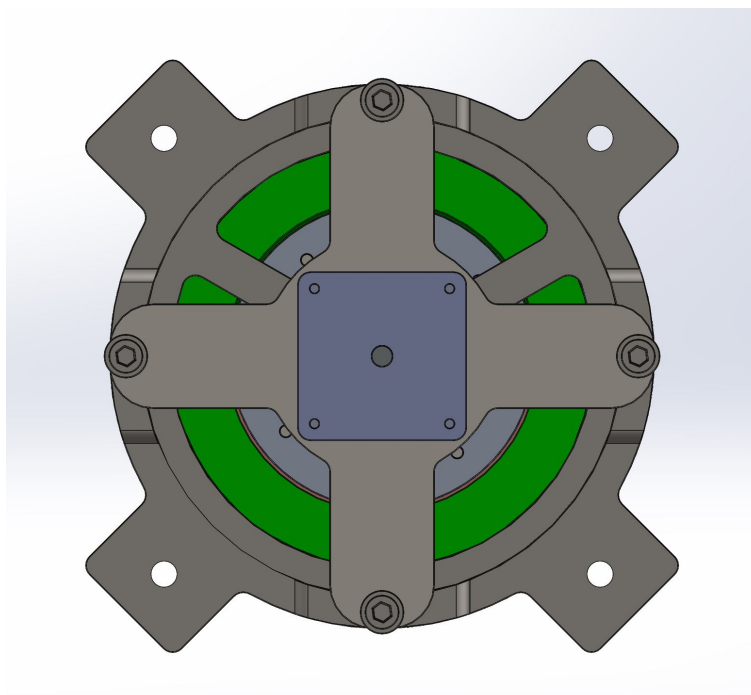


Figure 86: Bottom view, A.4.0.0

11.6.2 List of parts

- 2 x Stator
- Rotor
- Housing, P-C4-4.0.1
- 2 x SKF 6000-2RSH deep groove single row ball bearings
- Rotor Fixture, P-C3-3.0.3
- Stepper Motor
- Fixture for electrical motor, P-C4-3.0.3
- Shaft, P-C3-3.0.4
- 1 x, 1mm shim
- 1 x Disc spring
- 1 x M10 nut
- 30 x M2,5 bolts with shims
- M6 bolt, with shim
- 4 x M4 bolt with shims
- 4 x A2 nut with shims

11.6.3 Housing, P-C4-4.0.1

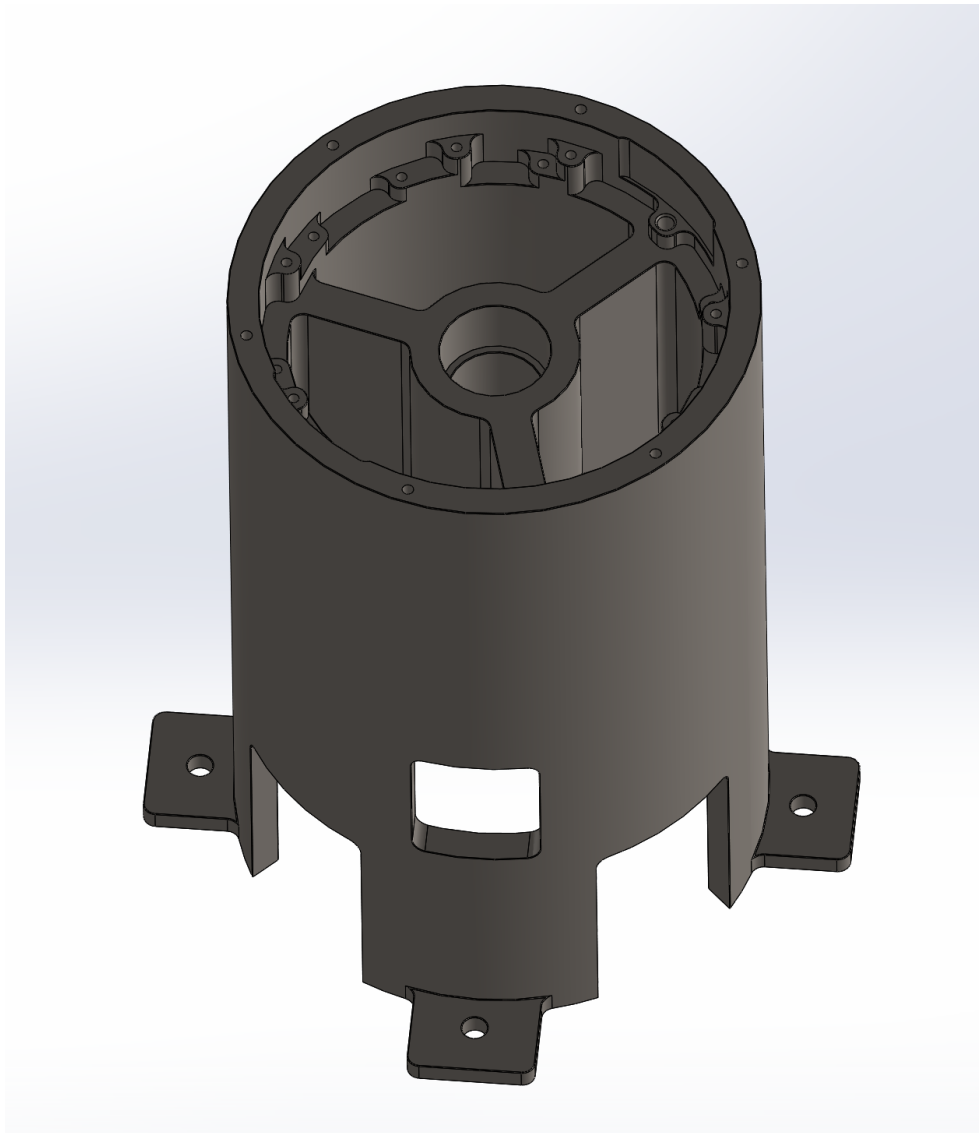


Figure 87: Housing concept 4, P-C4-4.0.1

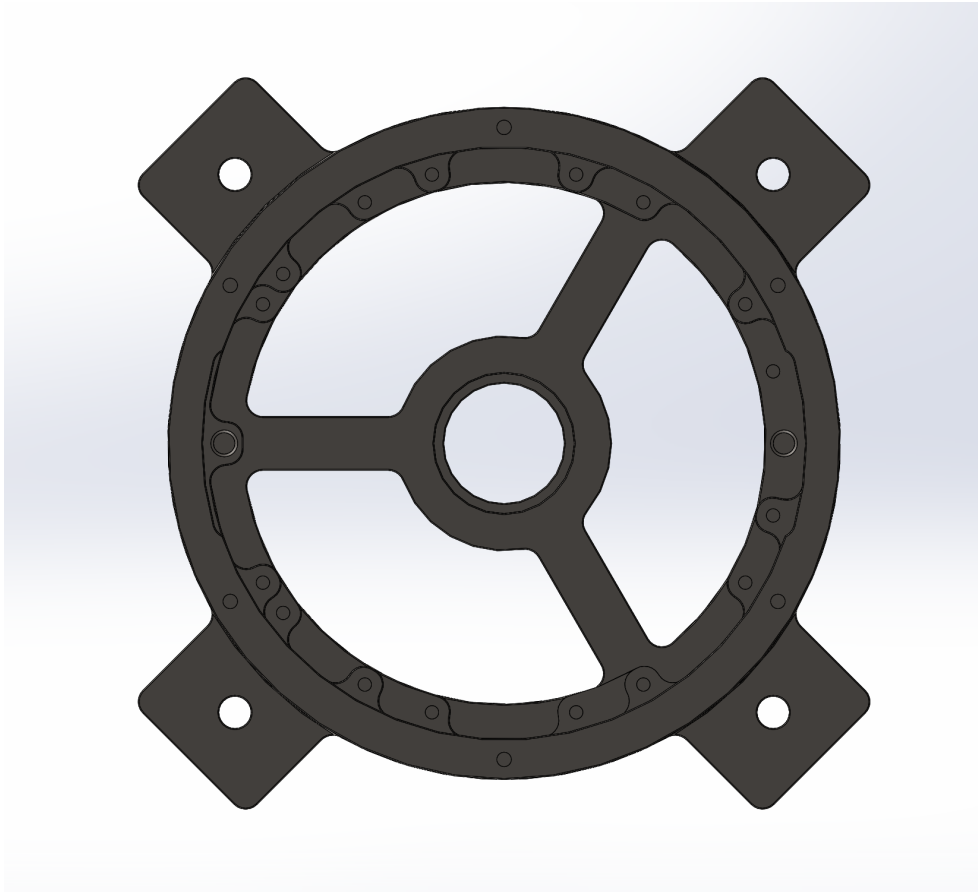


Figure 88: Housing top view, P-C4-4.0.1

The concept of this housing is to have the sensor concealed under a lid. This is to avoid parts sticking out, and protect the sensor from some external electromagnetic noise. The dowel pins guiding the PCB's have been replaced by holes. This is so the dowel pins can be made in a separate operation rather than milling of the housing.

11.6.4 Design review, DR-A.4.0.0

As the concept is very similar to concept 3, many of the design choices has already been mentioned in section 11.5.7. Due to the stators being concealed under a lid, holes for electrical wiring has to be taken in to consideration.

11.7 Choice of concept

The process of designing the concepts has been done in an incremental way, which leads to each concept going a step further from the previous. This means that the team will go forward with Concept 4. KDA approved the design. Going forward, research of materials and surface finish will be done.

11.7.1 2D

All 2D drawings are generated through SolidWorks. 2D-drawings are made so that the manufacturer easily can see the measurements of the parts. The drawings contain both general and specific tolerances 11.7.2. The drawings will also include specified materials and surface finishes.

11.7.2 Tolerances

The tolerances in the mechanical aspect of the system is a part of deciding how precise the outcome and quality of the tests will be. Making sure that the rotary plane is parallel to the stationary plane and that the position of the rotor is concentric to the spools on the rotor. Going from a 3D model and to production of parts, it is the tolerances that ensures how precise the parts are. The most significant tolerances for the created parts is the positioning of the holes, parallelism, perpendicularity and flatness [2]. A general linear tolerance of ± 0.2 mm and angular tolerance of 0.1° has been set for the drawings where specific tolerances have not been specified. The reasoning behind the specific numbers of the tolerances come from the requirements from KDA, wanting to have the same mechanical interface as they use to this date.

The specific positioning tolerances refer to two or more letters, A, B and C. These letters either refer to a surface or a plane, where the tolerance of position is relative to these letters.

Table 40: Tolerance symbols and descriptions, [21]

SYMBOL	GEOMETRIC CHARACTERISTIC	TOLERANCE TYPE	CONTROL SUMMARY
	FLATNESS	FORM (NO RELATION BETWEEN FEATURES)	CONTROLS FORM (SHAPE) OF SURFACES AND CAN ALSO CONTROL FORM OF AN AXIS OR MEDIAN PLANE DATUM REFERENCE IS NOT ALLOWED
	STRAIGHTNESS		
	CYLINDRICITY		
	CIRCULARITY (ROUNDNESS)		
	PERPENDICULARITY	ORIENTATION (NO RELATION BETWEEN FEATURES)	CONTROLS ORIENTATION (TILT) OF SURFACES, AXES, OR MEDIAN PLANES FOR SIZE AND NON-SIZE FEATURES DATUM REFERENCE REQUIRED
	PARALLELISM		
	ANGULARITY		
	POSITION	LOCATION	LOCATES CENTER POINTS, AXES, AND MEDIAN PLANES FOR SIZE FEATURES ALSO CONTROLS ORIENTATION
	PROFILE OF A SURFACE		LOCATES SURFACES ALSO CONTROLS SIZE, FORM, AND ORIENTATION OF SURFACES BASED ON DATUM REFERENCE
	PROFILE OF A LINE		
	TOTAL RUNOUT	RUNOUT	CONTROLS SURFACE COAXIALITY ALSO CONTROLS FORM AND ORIENTATION OF SURFACES
	CIRCULAR RUNOUT		
	CONCENTRICITY	LOCATION (DERIVED MEDIAN POINTS)	LOCATES DERIVED MEDIAN POINTS OF A FEATURE <i>NOT COMMON...CONSIDER USING POSITION, RUNOUT, OR PROFILE</i>
	SYMMETRY		

Table 41: Specific tolerances and values

Tolerance type	Range of values [mm]
Position	0.02 - 0.5
Parallelism	0.02
Perpendicularity	0.02
Flatness	0.02

11.7.3 2D-drawings of final concept

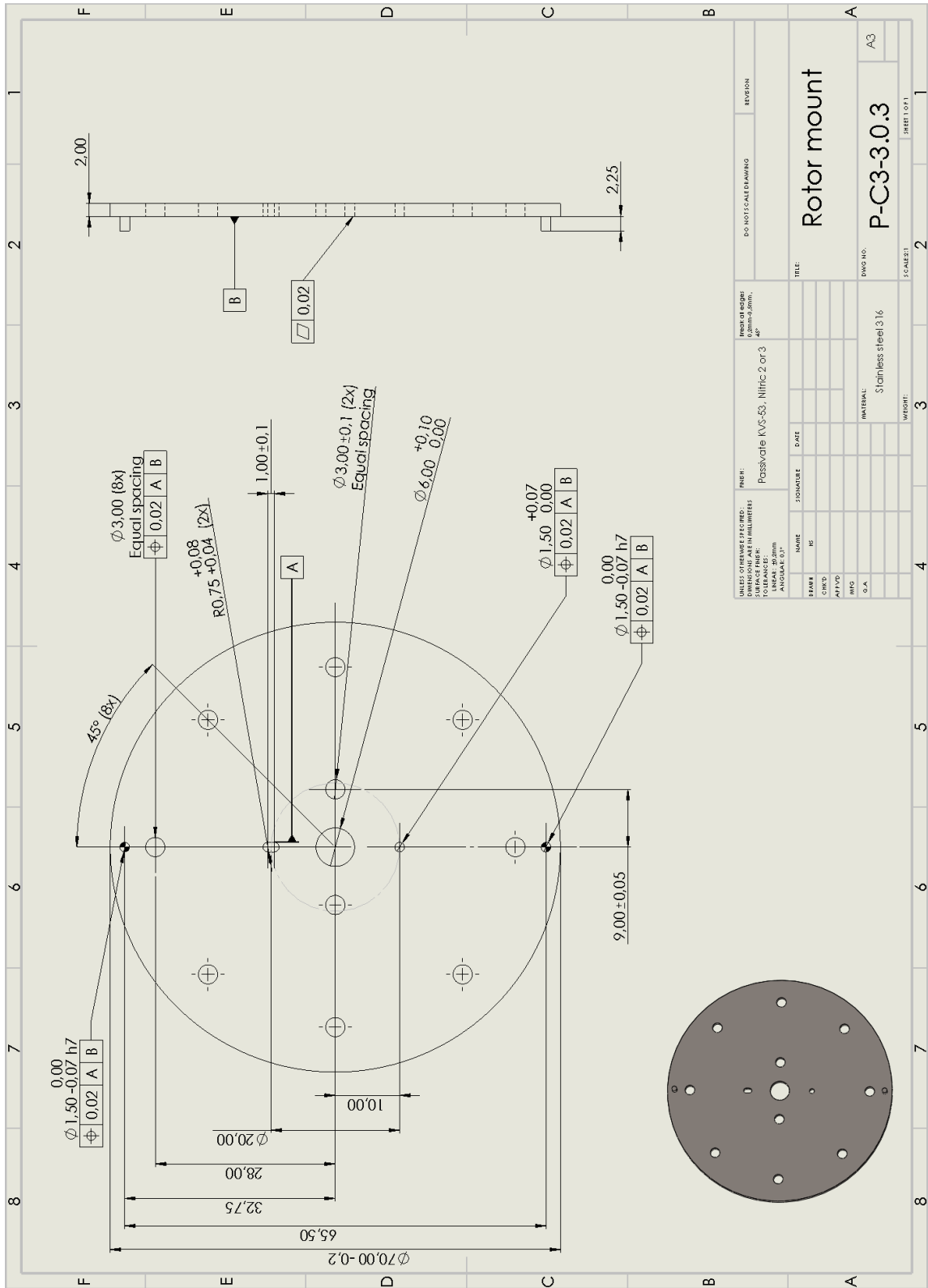


Figure 89: Rotor Mount measures and tolerances, 2D

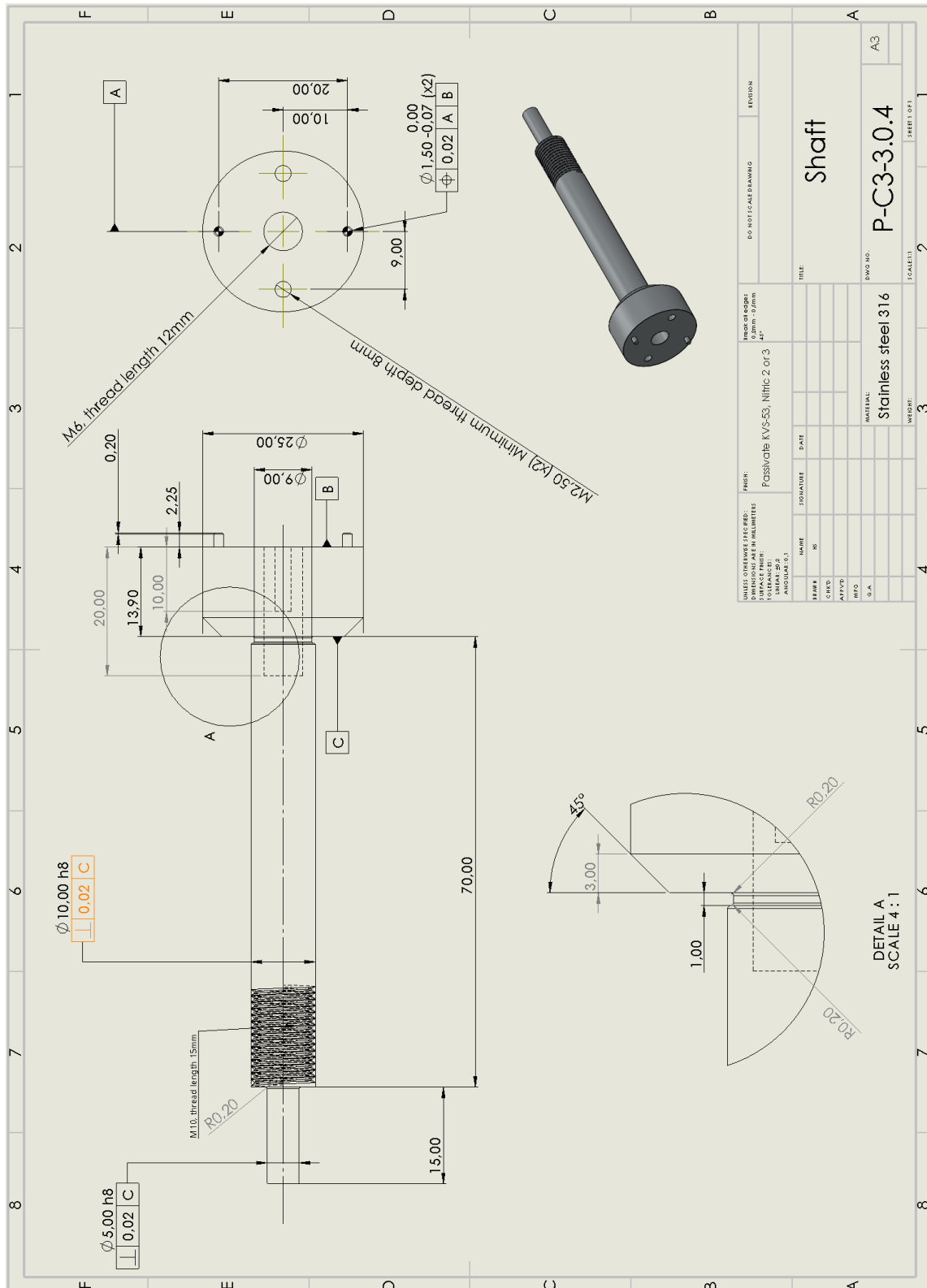


Figure 90: Shaft measures and tolerances, 2D

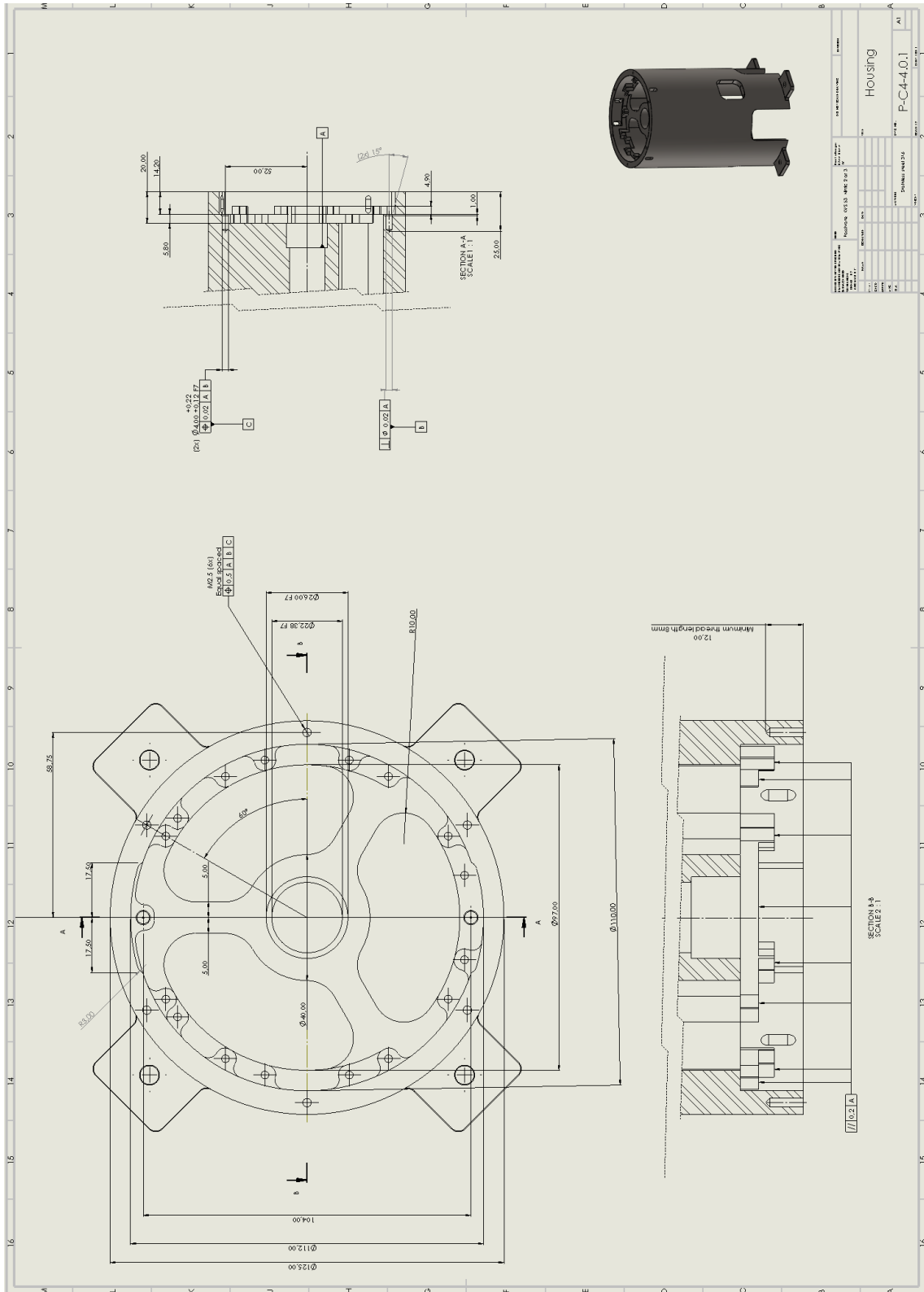


Figure 91: Housing measures and tolerances, sheet 1, 2D

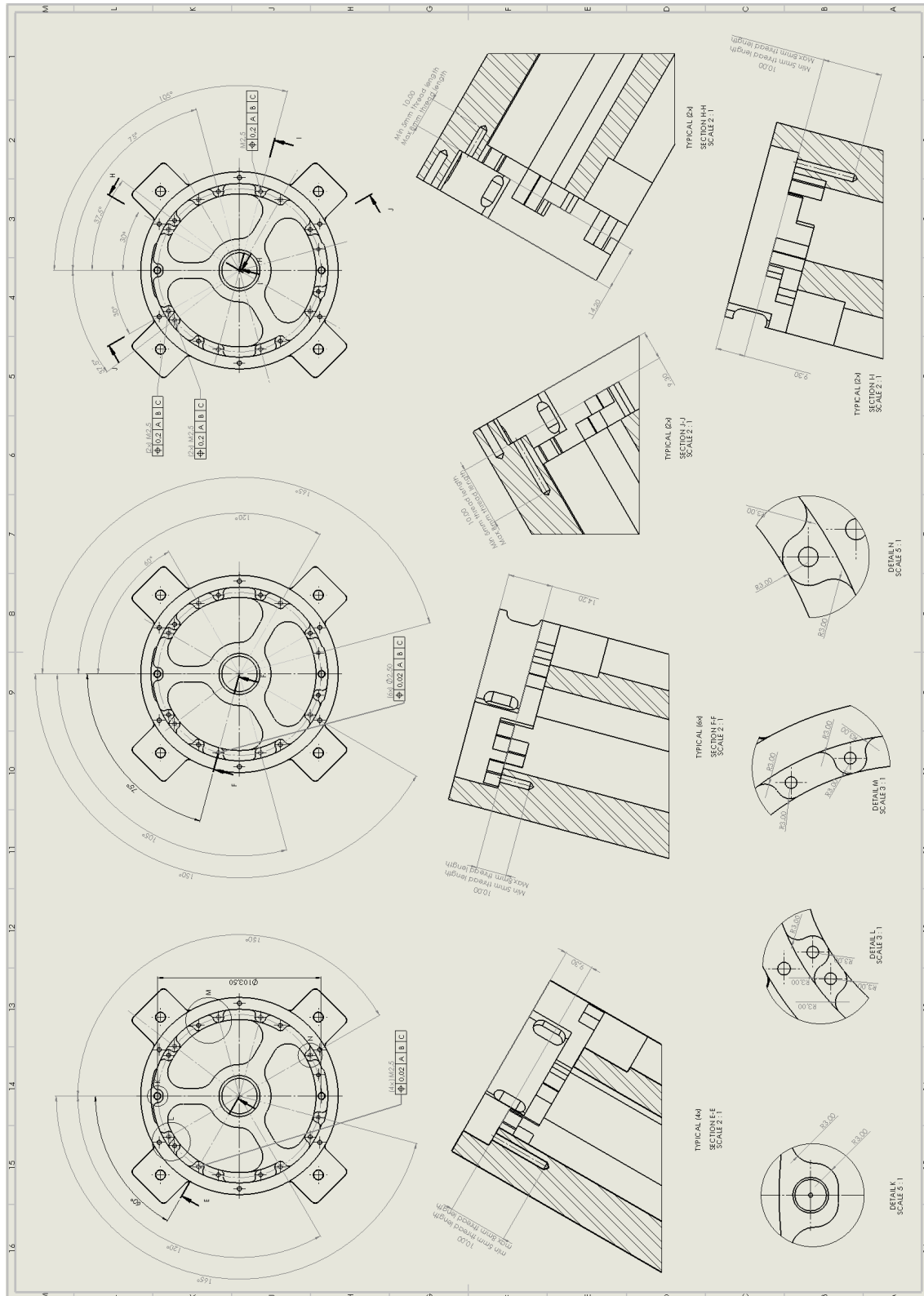


Figure 92: Housing measures and tolerances, sheet 2, 2D

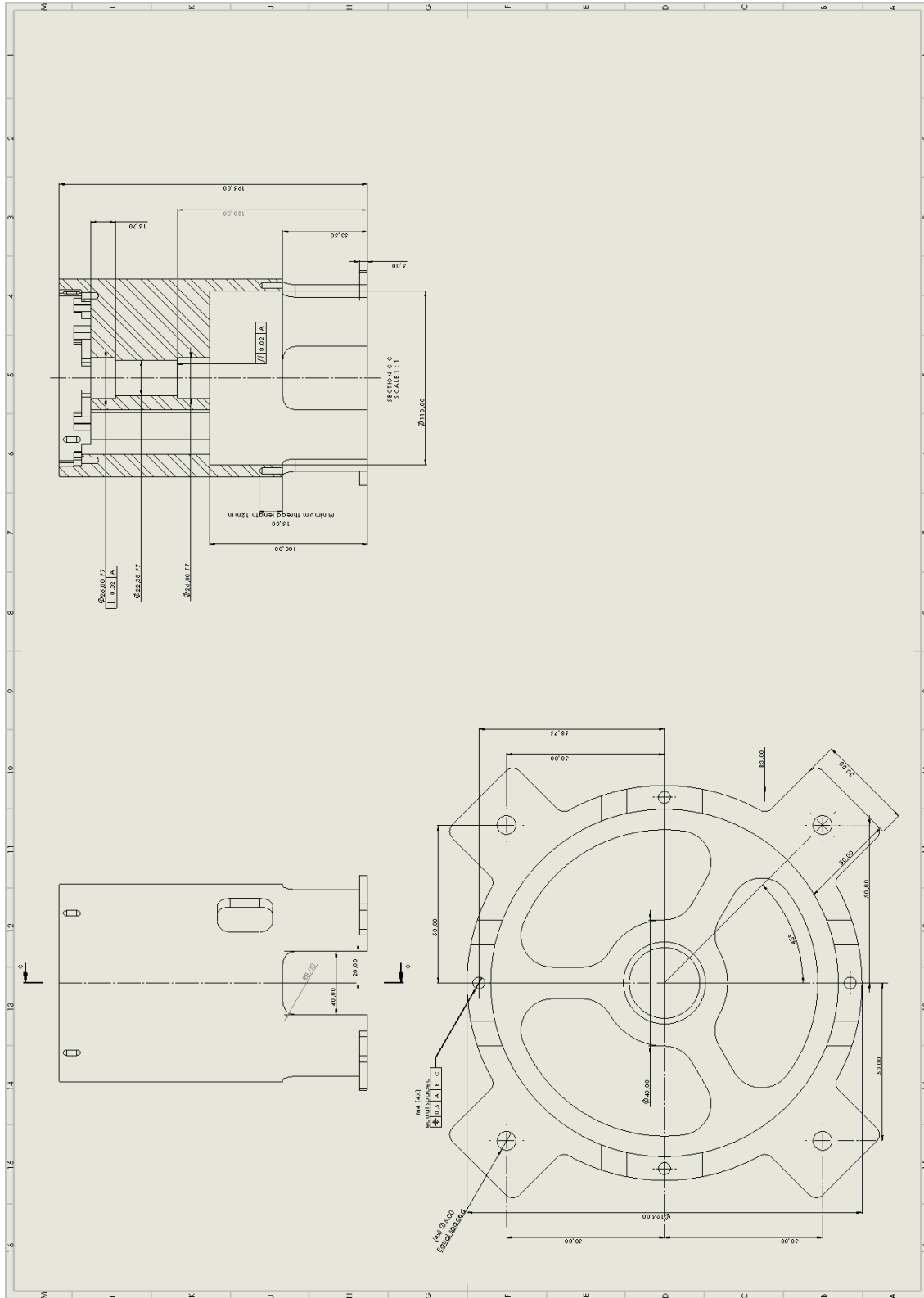


Figure 93: Housing measures and tolerances, sheet 3, 2D

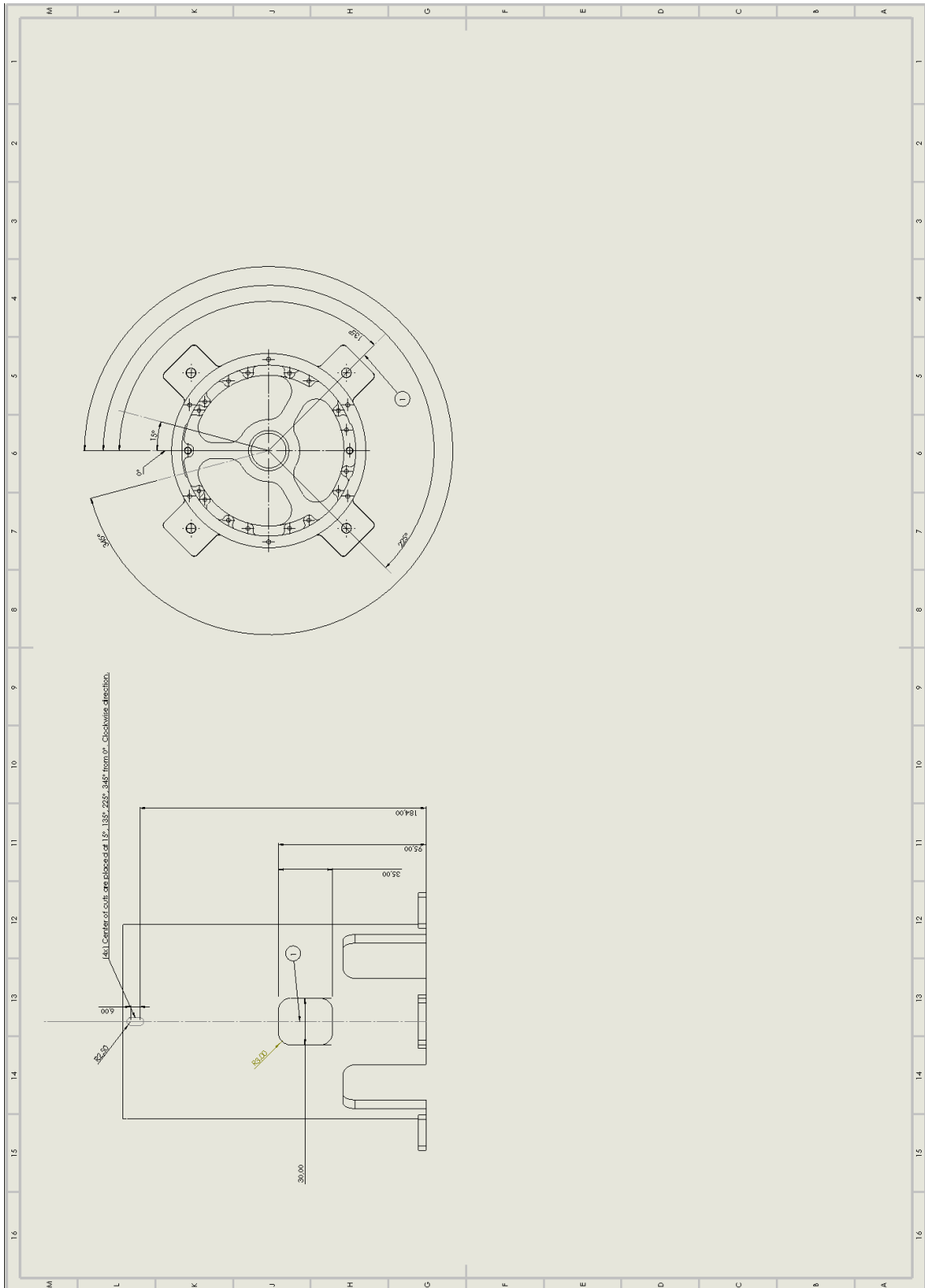


Figure 94: Housing measures and tolerances, sheet 4, 2D

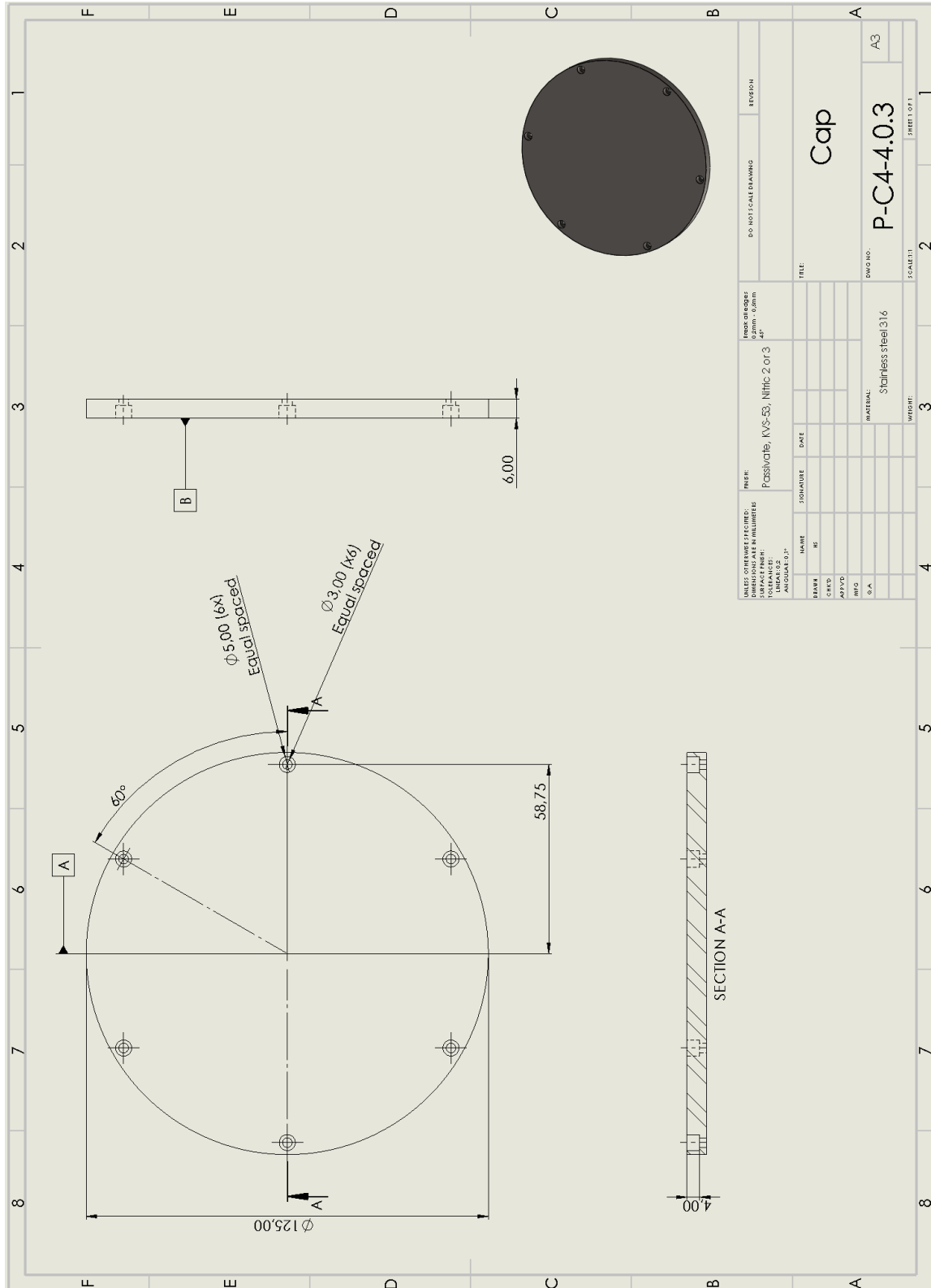


Figure 95: Cap measures and tolerances, 2D

11.8 Edits during manufacturing

During the process of ordering and manufacturing parts for the final system regarding the test station, a few changes was made. The changes was mainly done due to lack of time at the final stages of the project. To have the test station ready in time for testing before the hand in of the final report plain steel was used as material in the parts, and no surface finish was done. Stainless steel is more difficult and takes more time to work with regarding threading small holes. This was one aspect that shortened the time by using plain steel. A spring is not implemented for now either as the group focuses on testing in room temperature and achieving the requirements dealing with precision.

12 Bearings

12.1 Document history

Table 42: Bearings document history

Version	Date	Author	Description
1.0.0	12.02.2019	HS	Document created.
1.2.0	13.03.2019	HS	Added section 12.3 to 12.3.1.
1.3.0	19.03.2019	HS	Added section 12.4 to 12.5.
1.3.1	24.03.2019	JSS	Proofread.
1.6.0	08.05.2019	HS	Added sections 12.6 to 12.7
1.7.0	09.05.2019	HS	Added section 12.8
1.3.2	17.05.2019	MBC	Proofread and corrections.

12.2 Introduction

In this chapter the CPS team will explain the general idea behind bearings and how they can be applied to the project. The team will investigate different types of bearings and come to a conclusion of which bearing is best suited for the test station. A specific bearing will be chosen in the end of the bearing study, after all relevant requirements have been taken into consideration. This includes potential loads the system will endure, thermal factors and precision.

12.3 General

The purpose of a bearing is to allow linear or rotational movement and reduce friction between mechanical parts while handling stress. The bearing often guides a rotating, oscillating or sliding shaft, pivot or wheel. For simplicity a shaft will be used in examples. Bearings have the same function and objective no matter the application. This is to keep a shaft moving smooth and consistent while reducing friction. The reduction in friction happens through the bearings rolling internal mechanism. The energy it takes to slide or move a shaft over the surface is greatly reduced. From the requirements given to the CPS team from KDA, 1.2.1, 1.3.1 TR, 1.4.1 T, a very precise rotary system is needed in the test station(Requirements can be found in section 7.4). Bearings will help accomplish this by giving a smooth low friction rotation. Also, requirements dealing with temperature have to be considered in this study for the requirements 1.7.1 T and 1.7.2 T. The lubrication and choice of material of the bearings will cover these requirements.

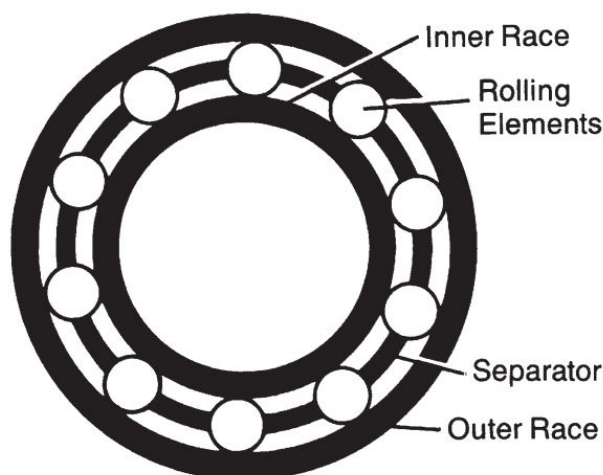


Figure 96: Example, ball bearing [52]

12.3.1 Loads

A bearing is pushed the same way the load is moving. If the load applied to the bearing exceeds the specifications of the bearing, it may fail. Therefore, it is important to understand the load applied to the bearing and in which directions, so a correct bearing can

be chosen. There are three types of loads; radial load, thrust load and angular load. [52]

Radial load; when the direction of the load is at right angles to the shaft. The load pushes down on the bearing. (Figure. 97) Radial means the direction of the radius. A radial load pushes down from the outer race inward to the balls, cage and inner race at the centre of the bearing. [52]

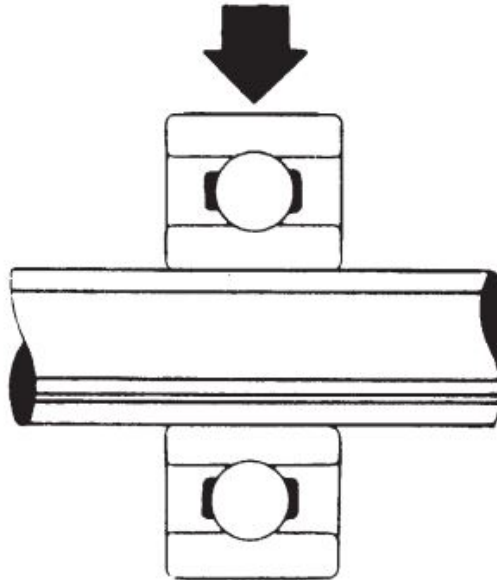


Figure 97: Radial load, [52]

Thrust load; when the direction of the load is parallel to the shaft. The load pushes sideways on the bearing. (Figure. 98) Thrust means a pressure of pushing force exerted sideways, pushing a shaft either right or left. This movement pushes the inner race of the bearing in the same sideways direction. The line of pressure runs parallel to the shaft. [52]

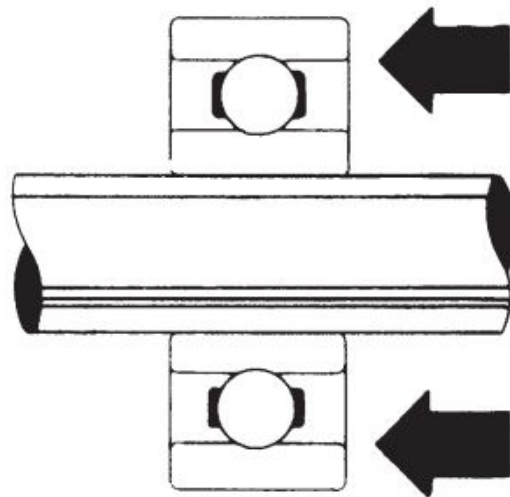


Figure 98: Thrust load, [52]

Angular load; when the load is a combination of radial and thrust load. The load pushes down and sideways on the bearing. (Figure. 99) As the load moves, it pushes against the corner of the race and the pressure is transmitted through the corner of the inner race diagonally to the opposite corner of the outer race. [52]

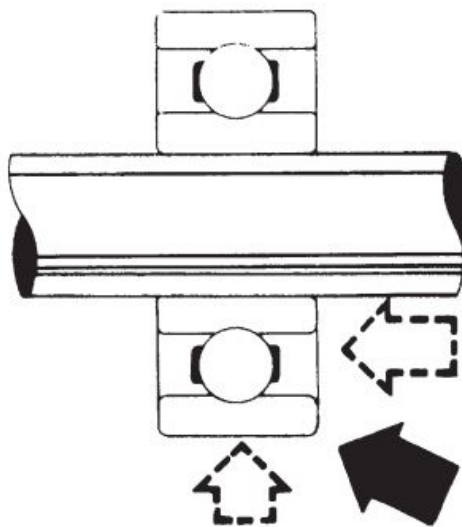


Figure 99: Angular load, [52]

12.4 Rolling bearing types

There are two categories of rolling bearing types; ball and roller. This section will cover the roller types. There are three types of roller bearings; tapered, cylindrical and needle. Cylindrical and needle bearings are non-tapered, this means the centre of each part run parallel to one another. However, for the tapered rollers, the inner and outer race are not parallel, which means if lines are drawn through the races they will coincide at one point. [52]

12.4.1 Tapered roller bearing

Tapered bearings profile resembles more of a conic shape than a circle. This type of bearing can handle all load types, in any combination.

Advantages:

- Cone shaped design which can align the rollers in the bearing perfectly with the cup and cone without guidance by the cage.
- Does not have a minimum load requirement to prevent slipping between the inner race and outer race.

Disadvantages:

- Struggles with dynamic misalignment because of the cone shape.
- Can handle less speed than more spherical bearing options.
- Requires a more complex housing.

12.4.2 Cylindrical bearing

The cylindrical bearing consists of four rolling parts; inner race, outer race, cage and rollers. The rollers are placed evenly by the cage, which guides their movement between the two races. Some of the cylindrical bearings have flanges on one or both races. This permits limited free axial movement of a shaft while supporting the rollers. [52]

Advantages:

- High capacity under radial loads.
- Accurate guiding of the rollers.
- Limited free axial movement (only for bearings with flanges).

12.5 Ball bearing types

Ball bearings are the most common type of bearings used in industry application. There are different types of ball bearings. This section will cover the single row bearing type, which is the most relevant ball bearing type for the projects application.

Similar to the roller bearing types, the ball bearings task is to reduce friction. The main difference between the two bearing categories is the shape of the rolling element. Ball bearings are spherical. This means the contact between the two races will be different than with the cylindrical rolling element. When no force is applied, the load is evenly distributed through the races and balls. Once a load is applied the balls start to roll. This motion causes the race to bulge out in front of the ball. The lower front quadrant of the ball flattens, while the lower rear quadrant bulges [52]. This is a continuous process for each ball as long as the load is in motion. The friction caused by this movement will eventually wear the parts and in the end cause failure. To lessen this friction choosing the correct lubrication is important [52].

12.5.1 Single row ball bearing

As the name indicates, this bearing type has a single row of balls. The single row bearing consists of an inner and outer race, separator, shields and a seal. The separator separates the balls and helps maintaining the distance between them. Shields protect the bearing from dirt and particles to enter the bearing, while allowing excess lubricant to flow through. They are placed between the two races. The seal is a barricade that prevents the lubrication from leaking out. It also protects the bearing from moisture, fine dirt, particles and other contaminants. Single row ball bearings can handle both thrust and radial loads [52].

Advantages

- Good performance for both radial and thrust loads
- The deep groove allows thrust load in any shaft direction
- Contaminant free operation when seals are used.
- Housings can have a simple design depending on the use.

12.6 Bearing selection

When designing the concepts, simplicity has been a key factor of how the system looks. This has also affected the choice of a bearing type. As mentioned above, single row ball bearings are the simplest types of bearings. To keep the rotation plane as parallel to the stationary, a simple housing is wanted. Using a single row ball bearing allows the design of the housing to be cylindrical and simple. Which again leads to less complications of keeping the critical planes parallel. The system is also experiencing very small amounts of force due to the low weight of components and little or no external forces. This is another factor for choosing a simpler type of bearing.

12.6.1 SKF 6000-2RSH

The bearing the team has chosen to use is SKF 6000-2RHS. Specifications of the bearing can be found in the data sheet [50]. The bearing is made of stainless steel, which is the same material as the rest of the parts in the system. The process of selecting a bearing was done by looking at the cad model, and see what dimension was possible to use. The dimensions of the housing the bearings shall be mounted inside can easily be adjusted in the cad model. This means the team had a lot of freedom selecting a bearing. In addition to this, the forces the system is exposed to are very small. This makes a large bearing unnecessary and will favour a smaller type of bearing to save weight.

The bearing itself will most likely not fulfil the requirements dealing with temperature 1.7.1 T. The reason for this is that most of the bearings from SKF have not been tested for lower temperatures than -40 °C. However the use of external lubrication (Molybdenum Disulphide) will create a dry film around the bearing and help isolate 12.7.1. This thin film will work as a layer of isolation and makes it possible to fulfil the temperature requirements.

12.7 Bearing lubrication

Selecting an external lubrication for the bearing is important to be able to fulfil the requirement 1.7.1 T. Lubrication makes sure the bearing operates reliable during operational conditions. Grease and oils are the most common form of lubrication in bearings. Using grease can increase heat build-up. This happens when the bearing is overfilled with grease between the balls and races. This causes the friction to increase, and a rise in heat can occur [53]. However this is mostly a problem in high speed situations, which the test station will not experience. Because of the temperature requirements common oil and grease lubricants are not good enough and will fail for the CPS team's testing purposes. This is mostly because the extreme cold temperature effects on viscosity of both oils and greases.

Because it is wanted from KDA to test in vacuum conditions at a later point, dry lubrication is a solid option [44]. Dry lubrication is often used in extreme condition environments. Particularly Molybdenum Disulphide (MoS_2) dry lubrication films containing MoS_2 are suited for the temperature ranges and vacuum environments the system will experience.

12.7.1 Dry Moly

Dry Moly is a lubricant produced by Ambersil [3]. Applying this lubrication to the selected bearing will make sure it can withstand the changes in temperatures during testing. Due to the low speed, and small amount of forces, the temperature requirement has been the most important factor for choosing a lubrication. The lubrication applies a thin coating around the metal surfaces of the bearing and works as isolation. A different option could be to use a custom bearing using internal Molybdenum Disulphide lubrication suited for the temperature range, instead of applying this externally. However this bearing type would be more expensive, so the team has decided to go forward with the external lubrication solution.

12.8 Bearing preload

A ball bearing has internal radial clearing between the ball and the two races. By applying force, in the form of preloading, the race and the ball will be in constant contact in the same position. The two races of the bearing will be pulled away from each other and apply pressure to the balls.

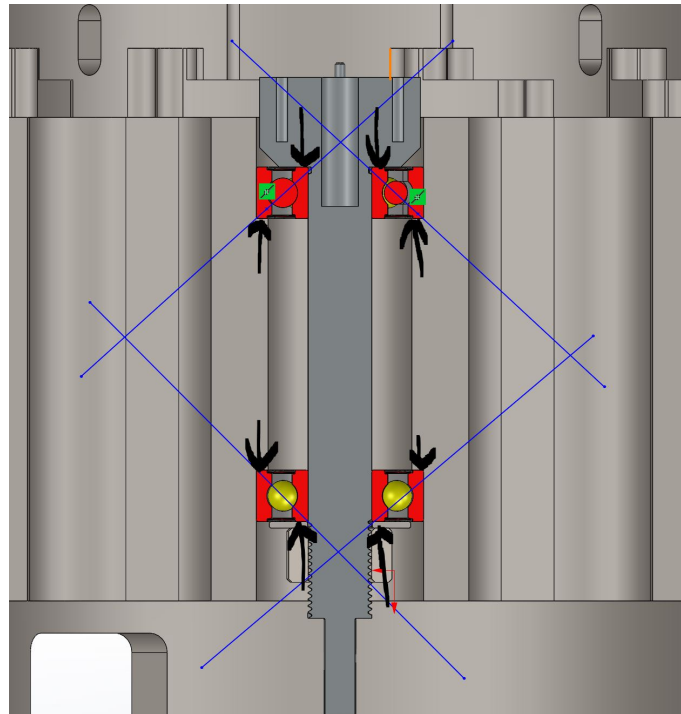


Figure 100: Preloaded bearings, forces on balls

Primary benefits of preloading bearings are: [51]

- Enhanced stiffness
- Reduced noise levels
- Improved shaft guidance
- Compensation for wear and settling
- Extended bearing service life

The most important benefits preloading provides for the system is improved shaft guidance and compensation for wear and settling. Improved shaft guidance will help achieve a more consistent and precise rotational axis. Also considering wear and settling, preloading helps. The state of the bearing will change after use over time, and by preloading, this will be minimized. This also applies for dynamic forces applied to the system such as temperatures. The shaft will either expand or contract depending on the temperatures applied, which will cause the bearing to experience dynamic forces. However just preloading alone will not completely mitigate the effects. The shaft experiences deflection

during change in temperature, by using a spring to mitigate this, the dynamic forces on the bearings will be mitigated.

12.8.1 Springs

The task of the spring is to mitigate the change in dynamic forces the bearing experiences due to elongation of the shaft when temperatures varies. Relevant springs for the system are disc springs. This spring type is circular and fits around the outer race of the bearing. To select a specific spring, the team calculated the bearing preload force in kN [51]

$$F = k \cdot d, \quad (64)$$

where k is a factor for small electrical motors. d, is the bore diameter of the bearing [mm]. The value of k varies from 0.005 and 0.01 normally. However if the preload is to protect the bearing from external vibrations the k-value should be set to 0.02. [51] Calculations will be made for factor 0.005, 0.01 and 0.02.

Table 43: k-value and preload forces

k-value	Preload force [N]
k = 0.005	50
k = 0.01	100
k = 0.02	200

From the calculations the preload should be between 50 N and 100 N under normal circumstances and 200 N when external vibrations are applied. From these values it is possible to select a fitting spring. The selection of a spring will happen by looking at the dimensions needed which are decided by the outer race of the bearing and the diameter of the shaft the spring is sitting around, as well as the spring force and deflection. Taking these factors into consideration, the choice was between a low force disc spring and a regular disc spring. As there is a big difference in the preload force, ranging from 50 N to 200 N it was decided to use two different springs depending on the uses and tests which will be conducted.

12.8.2 Disc spring, 4293

The spring was selected from a lead manufacturer of springs; Lesjöfors. The 4293 disc spring is meant for the 200 N preload and during external vibration testing. From the data sheet, [29], at 25 % deflection of the spring, the spring force is 240.4 N. At 15 % deflection this value is about 40 % less, which means for a preload of 200 N the spring will be deflected between 15-25 %. At 25 % deflection the distance the spring is moved in height is 0,2 mm. From calculations of materials of the shaft the elongation expanding from room temperature will be 0.16 mm (67). This means the spring will be able to mitigate the entire range of elongation the system will experience.

The spring comes in different materials and alloys. The most important property for the spring in this system is operational temperature. Because of this, the selected material is a

stainless steel type called EN 10270-3 – 1.4310 / X10CrNi18-8 / AISI 302. [38] The operational temperature of this spring is between -200 °C and +250 °C.

12.8.3 Disc spring, 5028

5028: DSL 23,7X14,3X0,4 is a low force disc spring [30]. This spring is more suited for the 50 N preload, but cannot exceed the 75% spring force which is at 80.6 N. 50 N preload will deflect the spring between 25-50% which means the spring will have moved between 0,125 mm and 0,250 mm. This spring will also mitigate the entire range of the shafts elongation. This spring will be of the same material as the 4293, as it will experience the same range of temperatures.

13 Choice of materials

13.1 Document history

Table 44: Choice of materials document history

Version	Date	Author	Description
1.0.0	16.05.2019	HS	Document created, added section 13.2 to 13.4
1.2.0	17.05.2019	HS	Added section 13.5
1.2.1	21.05.2019	HS	Proofread

13.2 Introduction

In this section, the team will find a material best suited for the parts of the test station. This section will also cover surface finish. This will mostly revolve around selecting a material that can deal with the temperature requirements and testing in a vacuum environment. Taken this into consideration and from tips given by KDA the team has decided to mainly look into austenitic stainless steel types. As well as looking into the effects of elongation.

13.3 Important elements

To be able to select a material, the team has to know which factors are most important to the system. For the test station, corrosion resistance is important because of moisture occurring during temperature testing. Mechanical properties, the most important being the thermal expansion coefficient. The thermal expansion coefficient and the length of the part decides how much the parts expands or contracts. It is important to understand how much the parts will expand because it is a factor that can influence the precision the sensor will be able to measure. The most critical parts considering temperature is the shaft and housing of the test station. Due to changes in temperature it was decided to implement a spring to make sure constant force always is applied to the bearings 12.8.1. However the spring does not help mitigate the elongation, just the forces caused by elongation.

13.4 Austenitic stainless steel

Austenitic stainless steel types are known for extremely good corrosion resistance and being non-magnetic. They contain chromium and nickel and are referred to as 300-series [59]. A tip from KDA was to look into the specific stainless steel types 304, and 316. The reason aluminium is not considered in the system is mostly because of the high thermal expansion coefficient. Almost double of plain steel alloys [57]

13.4.1 Stainless steel 304, 316

The 304 stainless steel and 316 stainless steel are very similar alloys. Among the two, the 304 type is most commonly used and is the cheapest. What makes the 316 stainless steel more expensive is the Molybdenum value of approximately 2% [59]. 316 also contains slightly more nickel, which makes it more corrosion resistant. This makes the 316 superior to 304 considering corrosion.

Looking at the thermal expansion coefficients of the two, 316 has the lowest coefficient [59]. For the range of temperatures the system will experience, 304 has a coefficient of 17.3, while 316 has 16.0 [57] This means when the materials are exposed to change in temperature 304 will experience a larger change in volume than 316. The thermal expansion coefficient is linear which means how much the material expands is also affected by the size of the part. The thermal coefficient of a material is expressed by, the materials coefficient multiplied by $10^{-6}^{\circ}\text{C}^{-1}$ [58]. For these reasons 316 will be chosen over 304.

13.4.2 Elongation of housing and shaft

To calculate the elongation of the housing and the shaft, the length of the parts and the change in temperature are the key factors. The elongation can be expressed as,

$$\Delta L = L_0 \alpha \Delta T, \quad (65)$$

where ΔL is the elongation. L_0 is the initial length of the part. α is the thermal expansion coefficient of the material. ΔT is the change in temperature.

The elongation of the shaft can be expressed as,

$$\Delta L_1 = 98.9 \text{ mm} \cdot 16(10^{-6} \text{ }^{\circ}\text{C}^{-1})200 \text{ }^{\circ}\text{C} = 0.32 \text{ mm}. \quad (66)$$

This elongation will be the full range from -80 °C to +200 °C. However half of the elongation will be expansion, and half will be contraction. This means the shaft will expand from 20 °C and upwards, and contract from 20 °C and downwards by:

$$\Delta L_2 = 98.9 \text{ mm} \cdot 16(10^{-6} \text{ }^{\circ}\text{C}^{-1})100 \text{ }^{\circ}\text{C} = 0.16 \text{ mm} \quad (67)$$

The elongation of the housing can be expressed as

$$\Delta L_3 = 195 \text{ mm} \cdot 16(10^{-6} \text{ }^\circ\text{C}^{-1})200 \text{ }^\circ\text{C} = 0.62 \text{ mm.} \quad (68)$$

Elongation from 20 °C to 120 °C,

$$\Delta L_4 = 195 \text{ mm} \cdot 16(10^{-6} \text{ }^\circ\text{C}^{-1})100 \text{ }^\circ\text{C} = 0.31 \text{ mm} \quad (69)$$

since the rotor is mounted on the top of the shaft, and the stator is mounted on top of the housing, the expansion can be halved. this is because the parts expand in either direction. from the middle and upwards the housing will expand in one way, and downwards another way. since both rotor and stator are mounted in the top section of the parts, they will follow the expansion in the same direction. Because of this the expansion can be halved looking at the position change in the stator and rotor. Halving (67) the expansion from the middle and upwards is $\Delta L_5 = 0.080 \text{ mm}$. And for the shaft $\Delta L_6 = 0.155 \text{ mm}$. L_{RS} is the distance between the rotor and the stator.

$$L_{RS} = \Delta L_6 - \Delta L_5 \quad (70)$$

$$L_{RS} = 0.15 \text{ mm} - 0.080 \text{ mm} = 0.075 \text{ mm} \quad (71)$$

13.5 Passivation

Passivation is a form of surface finish. It removes free iron from the surface of a creates a thin dense layer of oxide []. Free iron appears on the surface of stainless steels after manufacturing from forming tools. Passivation is a chemical treatment, and uses a specific composition of acids to remove the free irons and other contaminants on the surface of the material.

13.5.1 Nitric acid passivation

KVS-53 is a passivation process developed by KDA [6]. Because of the request of having a high corrosion it was decided to add a surface finish. KDA has a lot of experience in this field and suggested Nitric 2 or 3. What separates Nitric 2 and 3 is the % usage of nitric acid and the temperature of the acid during the passivation process [1].

14 KDA motor and SMC133-2 driver

14.1 Document history

Table 45: KDA motor and driver document history

Version	Date	Author	Description
1.0.0	18.02.2019	JSS	Document created.
2.0.0	14.03.2019	JSS	Added other ways to control stepper.
2.1.0	19.03.2019	JSS	Added unit table and other drivers.
2.1.1	26.03.2019	HS & MBC	Proofreading and corrections.
2.2.0	25.04.2019	JSS	Added sections driver nonfunctional and conclusion.
2.2.1	17.05.2019	MBC	Proofreading and corrections.
2.2.2	22.05.2019	CPS team	Proofreading and corrections.

14.2 Introduction

This chapter will introduce the stepper driver and motor borrowed from KDA and how they were implemented into the second concept for CPS.

14.3 Stepper motor

The stepper motor borrowed from KDA is of unknown model, but from its datasheet, it is of high quality as it is meant to be used for space applications. It is a two-phase stepper motor and has a low operation voltage of 1.8 V and current of 1.6 A. Without microstepping, the motor has 400 steps in one rotation. It has four coils in total, but two are mainly used. The other two can be used for redundancy or to give the motor a stronger torque if needed. The stepper motor will be used to control the rotation of the CPS and as a way to measure the accuracy of the sensor. By counting the steps the stepper motor has moved, the location of the sensor can be determined.

NanoTec also provided a technical manual for the driver [43] that provides detailed setup for the NanoPro software and driver.

14.4 Stepper driver

An SMCI33-2 stepper [41] motor driver was borrowed from KDA. It is desired to measure the angle of the CPS with a 0.01° precision, thus a stepper motor capable of turning with such a small amount each step is needed. The SMCI33-2 driver is capable of micro-stepping down to 1/64-steps [40]. With the stepper motor also borrowed from KDA that has a default step resolution of 400 steps per rotation, the motor is capable of gaining a total 25 600 steps per rotation giving a 0.014° rotation per step. With this it is not possible to measure every degree in a circle down to a 0.01° change of angle, but still possible to test precision of the 25 600 different angles that the stepper motor and driver can give.



Figure 101: Nanotech SMCI33-2 driver

14.5 NanoPro software - first code version

The SMCI33-2 driver has an integrated controller and can with the accompanying software NanoPro [41] be used to gain full control of the stepper motor. In NanoPro it is possible to change the current and drive step angle that suits the motor and tell it to either use full steps, to be able to see the rotation easier for debugging of the arduino code, or 1/64th steps resolution to test the accuracy of the sensor. This software has high functionality if used properly but is difficult to take use of, as there are unknown phrasings and the documentation on the software is light.

The phase current and phase current during idleness is of high importance here. The stepper motor that was borrowed from KDA has a low operating voltage of 1.8 V and an operating current of 1.6 A. This makes it possible to use three 9V batteries in series, as a power supply to deliver just short of the 27 V it is required as low current, to not damage the stepper motor.

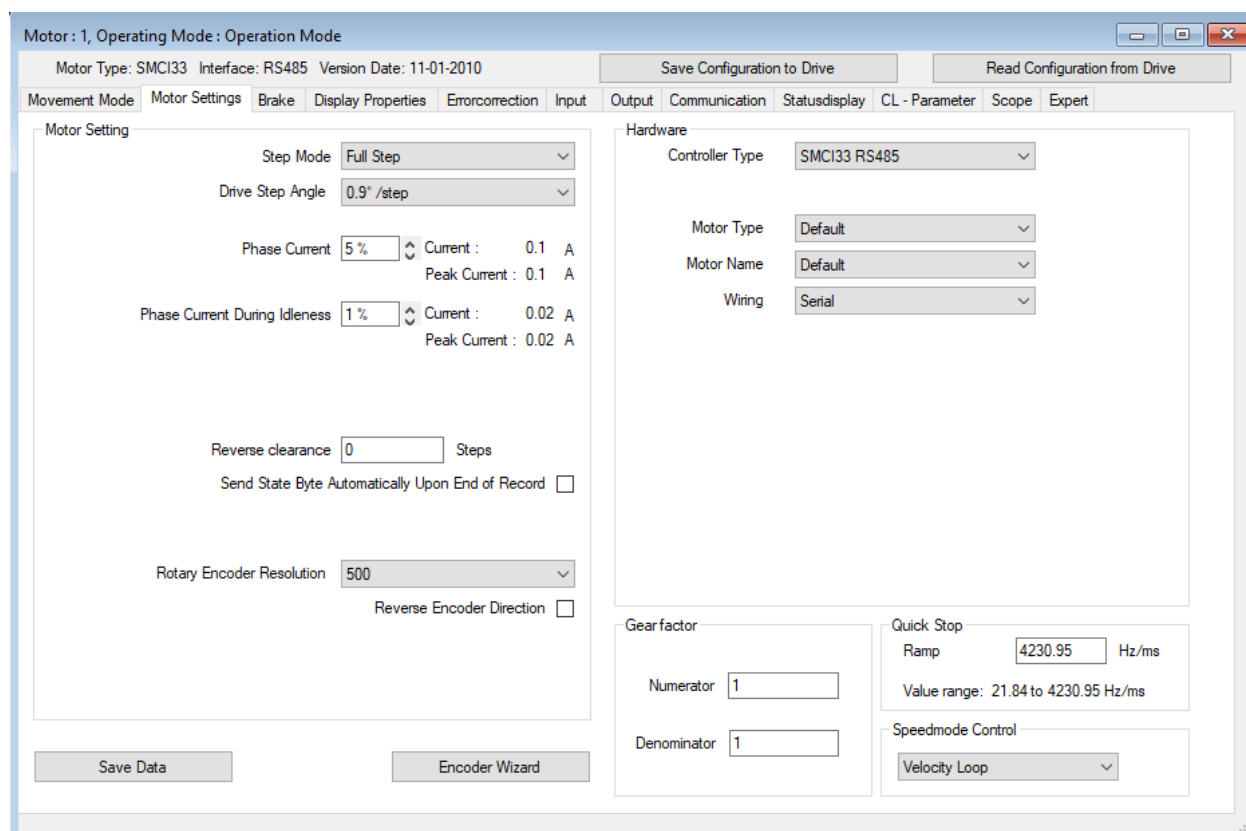


Figure 102: NanoPro motor settings tab

The driver is set to run the motor on a clock pulse on input 6 on the driver, and to change direction depending on if the input 1 is high (counter clockwise) or low (clockwise).

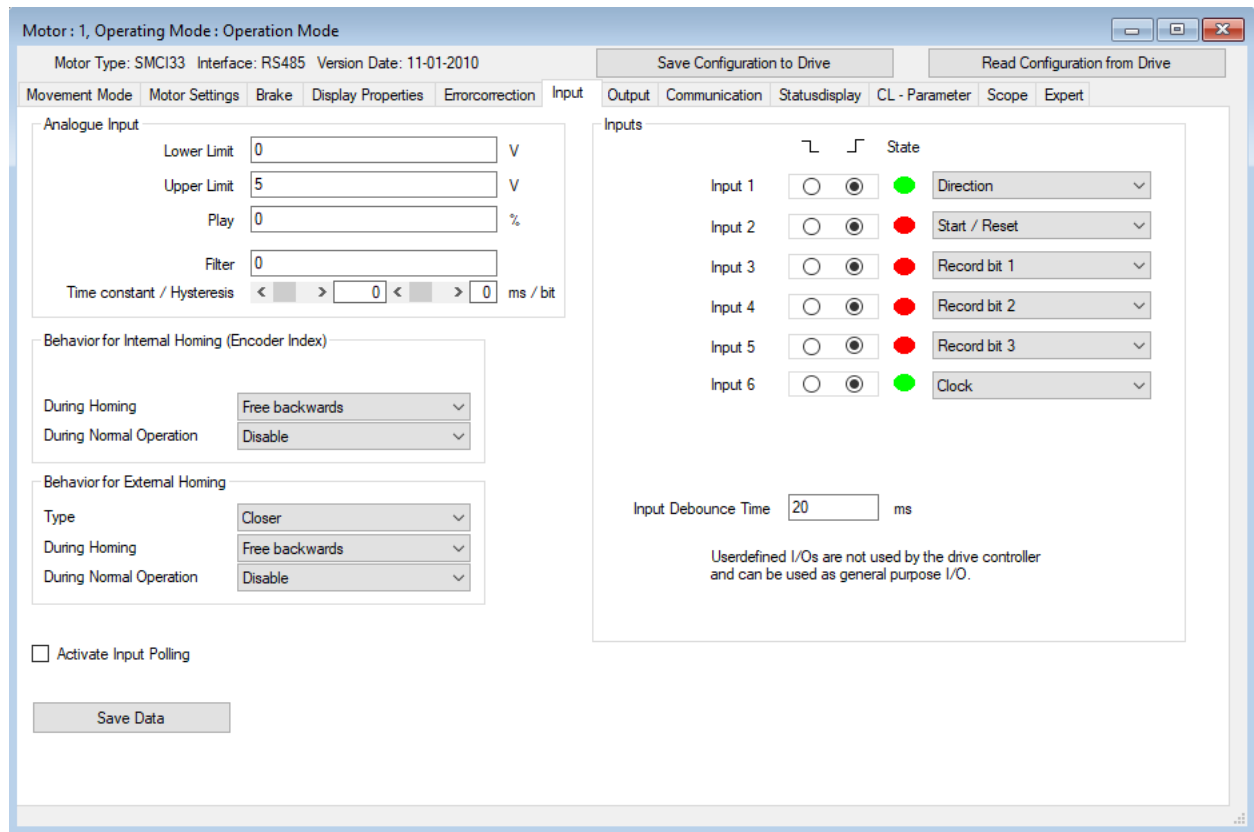


Figure 103: NanoPro input settings tab

14.5.1 Arduino code - first code version

Code listing 1 is the first code version used to test how it is to control the stepper motor through the SMCI33-2 driver using the NanoPro software. It is an endless loop that will take the motor 2000 steps in one direction for so to change and step the same amount back. It was mainly used to get to know the Nanopro software and later to see the sensor in action.

The pin number 6 is used to control the direction of the driver and pin number 9 is used to send a pulse to the driver. "Int steps" is amount of steps the motor will go in one direction and the "stepDirection" is to keep track of which direction it is currently moving.

```
1 /* Initialize pin numbers,  
2 * and variables */  
3 const int dir = 6;  
4 const int clk = 9;  
5 int steps = 2000;  
6 int stepDirection = 0;  
7  
8 /* Deciding pinmode and */  
9 void setup() {  
10 // initialize pins  
11 pinMode(dir, OUTPUT);  
12 pinMode(clk, OUTPUT);  
13  
14 digitalWrite(dir, LOW);  
15 digitalWrite(clk, LOW);  
16 }  
17 /* Running x amount of steps before changing direction and  
18 repeating */  
19 void loop(){  
20 //Counterclockwise  
21 if (stepDirection == 1){  
22     for (int e = 0; e < steps; e++){  
23         digitalWrite(clk, HIGH);  
24         delay(1);  
25         digitalWrite(clk, LOW);  
26         delay(1);  
27     }  
28     digitalWrite(dir, LOW);  
29     stepDirection = 0;  
30 }  
31 //Clockwise  
32 else if (stepDirection == 0){  
33     for (int e = 0; e < steps; e++){  
34         digitalWrite(clk, HIGH);  
35         delay(1);  
36         digitalWrite(clk, LOW);  
37         delay(1);  
38     }  
39     digitalWrite(dir, HIGH);  
40     stepDirection = 1;  
41 }  
42 }
```

Listing 1: Arduino code - first code version

14.5.2 Connection diagram

Illustrated in figure 104 is a simplified connection diagram. Three 9 V batteries was used as a power supply. The wires on the stepper motor is both colour coded and named depending on which coil they belong to.

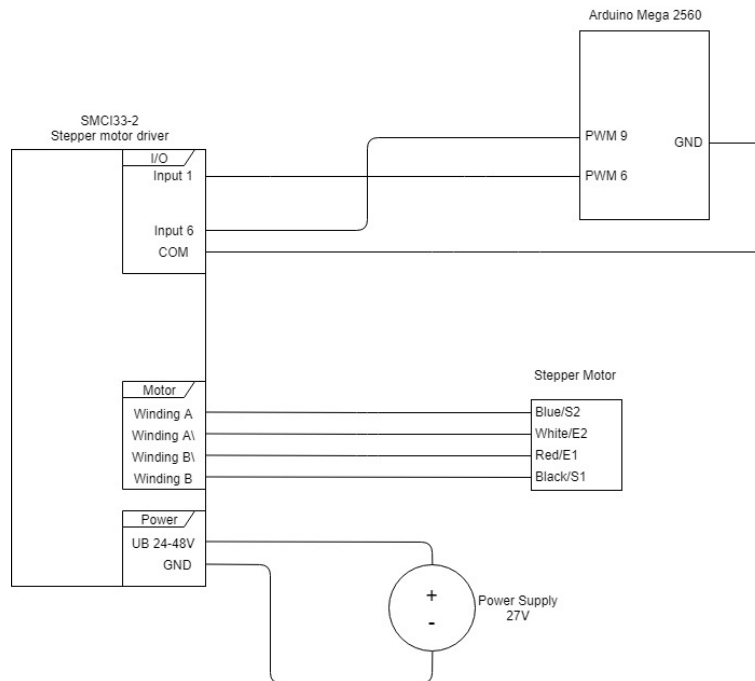


Figure 104: Code version 1 connection diagram KDA driver and motor

14.6 SMCI33-2 driver problems and possible solutions

The nanotec SMCI33-2 driver has been challenging to make use of after the updated stakeholder requirement 1.21 - 1.23 in table 18 where a more fine control of the stepper motor is required to change the speed, acceleration, distance, holding current, current when driving and stepmode. Because of this the previous way of controlling the stepper motor is not adequate as they can not be controlled through our own software.

Other ways Nanotec has for communicating and controlling the stepper motor driver has been explored and problems encountered along the way.

14.6.1 NanoJEasy

NanoJEasy is another software that is compatible with the SMCI33-2 driver and controls it in a different way than NanoPro. In NanoJEasy a program based on the Java programming language can be written and compiled to a program package that is transferred to the driver. This Java program allows for control over all the parameters that are needed to be able to change, but sending in and receiving data from the driver cannot be done through a USB connection this way. Other means of controlling the driver must be researched.

As the driver has 6 input pins, 1 analogue input pin and 3 output pins, it is possible to use these to communicate through a custom-made communication protocol. Sending information from our software to the Arduino is fully possible and translating this information into PWM signals should be possible. Currently there are difficulties to get the analogue input pin on the driver to give a valid reading. If this can be fixed, then the analogue in combination with the 6 input pins can be used to generate the high numbers necessary for the parameters. Output pins from the SMCI33-2 driver can be used to verify that the values were properly transferred and that it is ready to receive new messages. NanoTec has included a programming manual for NanoJEasy [42].

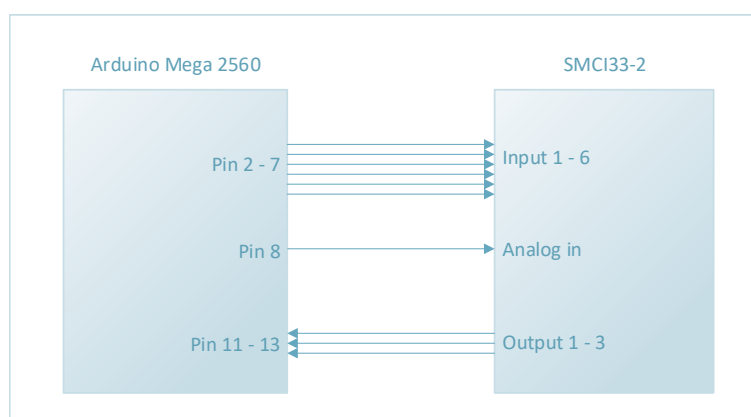


Figure 105: Code version 2 connection diagram KDA driver and arduino

14.6.2 Software development kit

With the NanoPro software there was an accompanying development kit for USB communication with the driver. Upon further research into the SDK, there was a problem of an outdated library that could not function on newer version of the .NET framework. Because of lack of knowledge around .NET, an email was sent to NanoTec in case there was a ready solution for the incompatibility.

The library that is available through the SDK would be easier to implement into the current main software program as using the arduino to translate messages would not be needed. Nanotec has included a programming manual for the library [42].

14.7 SMCI33-2 driver after researching

After extensive research into what the driver has to offer, the different ways to control the driver, the problems to efficiently communicating, the outdated software and C++ library, research for other alternatives that may provide an easier solution has started. The mail to NanoTec was answered and no solution for the outdated C++ library was available.

14.8 Important about the SMCI33-2 driver

The driver can be susceptible to a power surge if connected to a charging laptop. To avoid this unplug the laptop while configuring the driver or apply a USB power surge protector.

The firmware can be configured directly in the controller, but can cause unexpected consequences if not careful. If something were to happen where the firmware cannot be reset back to a clean version, then the driver would have to be sent in to Nanotec for a factory reset.

14.9 Parameters unit - SMCI33-2

Stepmode:

The stepmode will be configured with a number signifying the step resolution of the stepmode.

Table 46: Overview of the stepmodes available on the SMCI33-2 driver

Stepmode	Number
Full step	1
Half of a step	2
Quarter of a step	4
Fifth of a step	5
Eight of a step	8
Tenth of a step	10
16th of a step	16
32nd of a step	32
64th of a step	64

Speed:

The max possible speed changes depending on the stepmode the driver is currently in and is measured in Hz. This is given in Hz in the programming manual [42] as a stepper motor moves with steps, this will make it so move the same amounts of steps per second as the current frequency is set to.

Table 47: SMCI33-2 driver max speed per step resolution

Stepmode	Speed in frequency
1/2 step	1 to 32 000 Hz
1/4 step	1 to 64 000 Hz
1/8 step	1 to 128 000 Hz
1/16 step	1 to 256 000 Hz
1/32 step	1 to 512 000 Hz
1/64 step	1 to 1 000 000 Hz

Figure 77 and 80 illustrates that a frequency can be converted to angular rotation with expression (73).

Angular resolution is given by

$$N = \frac{360^\circ}{M}, \quad (72)$$

where M is the number of steps per rotation.

Angular rotation is given by

$$\omega_{deg} = NS\omega_{step}, \quad (73)$$

where N is angular resolution from (72), S is the step resolution of the driver shown in table 47 and ω_{step} is the speed in frequency.

Table 48: SMCI33-2 driver max speed in °/s per step resolution

Stepmode	Speed in °/s
1/2 step	0.45 to 14 400 °/s
1/4 step	0.1125 to 14 400 °/s
1/8 step	0.05625 to 14 400 °/s
1/16 step	0.028125 to 14 400 °/s
1/32 step	0.0140625 to 14 400 °/s
1/64 step	0.00703125 to 14 062.5 °/s

Other parameters:

The other parameters are as follows:

Table 49: Other available SMCI33-2 driver parameter range with units

Parameter	From min to max values
Current	0 to 150%
Distance	0 to 25 600 steps
Acceleration	0.019 to 2988 Hz/ms

The current goes in percentage of the operating current of 2 A. This means it has a current range of 0 A to 3 A.

Acceleration can take in values ranging from 1 to 65 535 as parameter (X). The values in the table is the min and max acceleration allowed after the conversion. The formula to convert it from a parameter to Hz/ms can be found in the programming manual [42], and is:

$$\omega = ((3000.0/\sqrt{X}) - 11.7), \quad (74)$$

Where ω is hz/ms, X is parameter value

14.10 Driver nonfunctional

While testing configurations and communication protocols with the SMCI33-1 it has become unstable and will lose the communication connection after approximately 20 seconds of operating. Because of this, it was decided to acquire a new driver without an

integrated controller which limits the variables that can be controlled but makes it easier to build a functional setup. The final driver chosen was the DM420A driver from long's motor [32].

14.11 Other drivers

Replacement drivers has been researched in case it proved too difficult to control the SMCI33-2 driver. One candidate that will allow to control most parameters through our software was found. However it did not allowed for the control off all parameters so researching replacement drivers will continue. The focus will still be on setting up the SMCI33-2 driver as it currently has the best control over the stepper motor. This is partly because of the time frame for the project and making use of a new driver could cause unexpected issues.

14.11.1 The EM402 - Leadshine

The EM402 driver by Leadshine [28] has capacity to go down to 1/512th microstep resolution if configured with their software, which is higher than the current SMCI33-2 driver. It can supply a low enough current for our stepper motor. On the negative side: few parameters can be controlled by external software which includes the stepmode and current reduction. Acceleration and speed can be controlled through the Arduino mega by changing the delay between each step, and distance can be controlled with amount of steps. The stepmode and current can also be controlled by dip switches on the driver, but these do not give as many options as their software.

After reconsideration this driver was no longer suited as it was decided to go against software dependant drivers due to difficulties during integration with the software to be developed.



Figure 106: EM402 stepper driver by Leadshine

14.12 SMCI33-2 driver conclusion

Because of all the problems to get the SMCI33-2 driver stable and functional it has been replaced with a new stepper motor, the DM420A from Longs-motors [32].

15 Final driver - DM420A

15.1 Document history

Table 50: DM420A driver document history

Version	Date	Author	Description
1.0.0	25.04.2019	JSS	Document created.
1.0.1	17.05.2019	MBC	Proofreading and corrections.
2.0.0	10.05.2019	JSS	Document updated and proofread.
2.0.1	22.05.2019	CPS team	Proofreading and corrections.

15.2 Introduction

This chapter will introduce the stepper driver that was decided to use in the end and the communication protocol to control it through an Arduino Mega 2560.

15.3 Backup driver

The SMCI33-2 driver started behaving erratically where the communication cut off after 20 seconds of operation time. This made it too unstable to use for the project and an alternative driver was needed as a replacement. Since the mechanical interface that has been developed for the project has a strict design since it has been made for the stepper motor borrowed from KDA. So in choosing a backup driver it was important that it could drive this stepper motor without damaging it with a higher voltage and current than it can sustain. With this in mind the DM420A stepper driver from Longs Motor was chosen [32].

15.4 Driver control

The control of the DM420A was easier to implement than the SMCI33-2 with an integrated controller. It does not have full control over all the desired parameters listed in stakeholder requirement 1.21 in table 18. Unlike the SMCI33-2 which had the possibility to control all parameters through code, the DM420 can only control the speed, distance and acceleration through code. The Stepmode and current can be controlled through DIP-switches on the driver itself.

To run the driver an Arduino Mega 2560 is used to give digital signals to activate the motor bridge, choose direction for the motor to run, give a pulse for each step it is to move and a +5 V power supply used for the digital signals. (5 V used alongside on of the other signals to activate a PNP transistor that functions like a switch.)

15.4.1 Stepper motor driver connection diagram

Illustrated in figure 107 is a connection diagram for the DM420A stepper motor driver with the stepper motor and the Arduino Mega 2560.

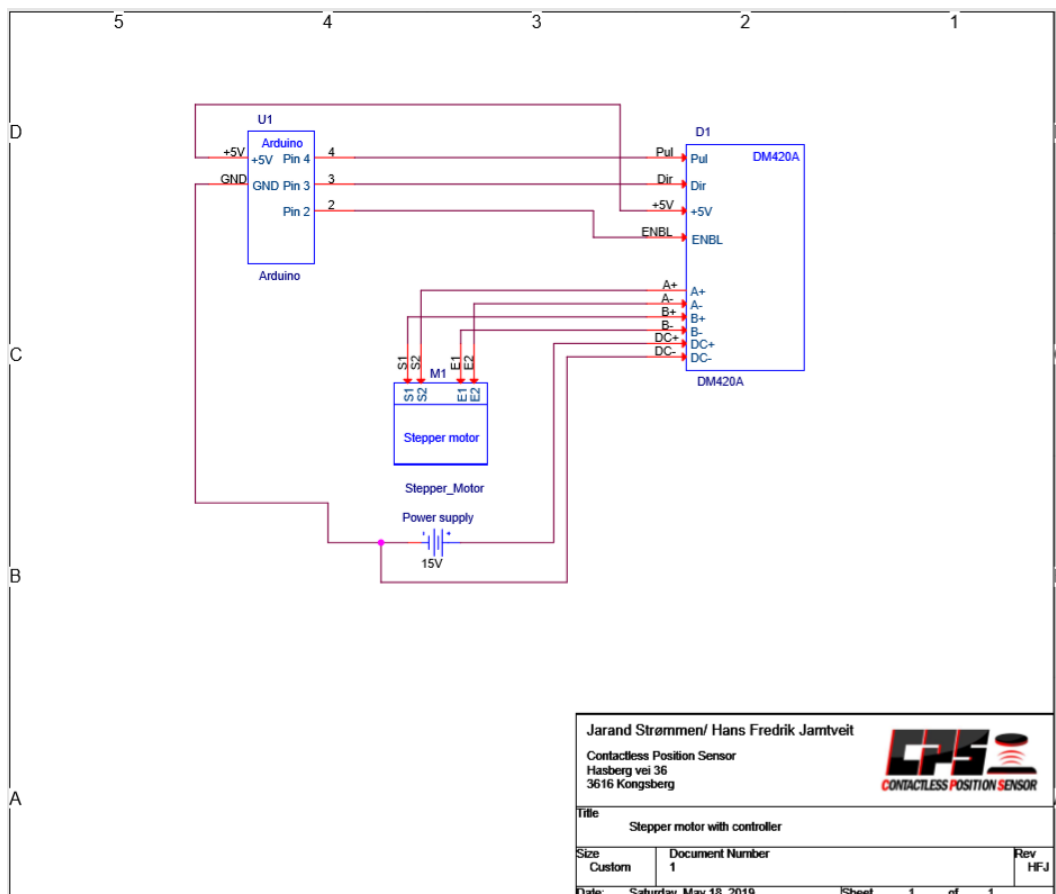


Figure 107: Stepper motor driver connection diagram

15.5 Parameters unit - DM420A

15.5.1 Current

With the DM420A driver the stepmode and current are controlled with dip-switches on the driver itself. The configurations possible are described in the tables below.

Root mean square (RMS)

Table 51: Overview of current settings on the DM420A driver

Peak current	RMS	Switch 1	Switch 2	Switch 3
0.44	0.31	ON	ON	ON
0.62	0.44	OFF	ON	ON
0.74	0.52	ON	OFF	ON
0.86	0.61	OFF	OFF	ON
1.46	1.03	ON	ON	OFF
1.69	1.20	OFF	ON	OFF
2.14	1.51	ON	OFF	OFF
2.83	2.00	OFF	OFF	OFF
Switch 4:	On = Half current Off = Full current			

15.5.2 Stepmode

The stepmode table described here is not identical to the one on the driver itself as a 400 steps stepper motor is used, instead of a 200 steps stepper motor which is what is used for the table on the driver.

Table 52: Overview of stepmodes available on the DM420A driver

Steps in full rotation	Micro step resolution	Switch 5	Switch 6	Switch 7
400	1/1	ON	ON	ON
800	1/2	OFF	ON	ON
1600	1/4	ON	OFF	ON
3200	1/8	OFF	OFF	ON
6400	1/16	ON	ON	OFF
12800	1/32	OFF	ON	OFF
25600	1/64	ON	OFF	OFF
51200	1/128	OFF	OFF	OFF

15.5.3 Speed

The speed depends on the stepmode the driver is currently in. Due to limitations on the stepper motor the speed becomes unstable at values above 180°/s. This happens because the coils inside the stepper motor start switching quicker than the motor can rotate the shaft connected to the motor leaving it to just vibrate at one spot. The minimum speed is restricted due to an integer limitation in the code.

Table 53: Overview of possible speeds with the DM420A driver

Stepmode	Min speed	Max speed
1/1	0.0009°/s (0.001 step/s)	180°/s (200 step/s)
1/2	0.00045°/s (0.001 step/s)	180°/s (400 step/s)
1/4	0.000225°/s (0.001 step/s)	180°/s (800 step/s)
1/8	0.0001125°/s (0.001 step/s)	180°/s (1600 step/s)
1/16	0.00005625°/s (0.001 step/s)	180°/s (3200 step/s)
1/32	0.000028125°/s (0.001 step/s)	180°/s (6400 step/s)
1/64	0.0000140625°/s (0.001 step/s)	180°/s (12800 step/s)
1/128	0.00000703125°/s (0.001 step/s)	180°/s (25600 step/s)

15.5.4 Distance

The max distance is set to 10 full rotations in a single motion. Since the goal of the project is to test the accuracy of the sensor, being able to run further would not be helpful. The least distance it can travel depends on the stepmode it is in as the minimum distance is limited to one step. The step to degree ratio can be seen below in table 54.

Table 54: Distance Overview of possible distances in each stepmode on the DM420A driver

Stepmode	Min distance	Max distance
1/1	0.9°(1 step)	3600°(4000 steps)
1/2	0.45°(1 step)	3600°(8000 steps)
1/4	0.225°(1 step)	3600°(16000 steps)
1/8	0.1125°(1 step)	3600°(32000 steps)
1/16	0.05625°(1 step)	3600°(64000 steps)
1/32	0.028125°(1 step)	3600°(128000 steps)
1/64	0.0140625°(1 step)	3600°(256000 steps)
1/128	0.00703125°(1 step)	3600°(512000 steps)

15.5.5 Direction

There was no requirement to control the direction the stepper motor would turn, but with the new driver this became a function that was not too difficult to add when the communication protocol was set up. Therefore it is possible to change the direction of the driver.

15.5.6 Acceleration

Because of the change in stepper motor driver the acceleration has become difficult to adjust. Even though the CPS can set up a decreasing delay between each step to add an acceleration. With a finer understanding of how slow the test station is in need of moving to accurately gather the position, the control of acceleration would be very limited or not have an impact for the test.

15.6 Conversions

To give the user some choices in the speed and the distance of the sensor there was a need to convert the values given by the GUI to the ones that the Arduino could use.

15.6.1 Distance

Distance can be given as either steps or degrees. For the Arduino this value needs to be specified as steps, so if it is given as degrees in the GUI this will need to be converted to steps. This conversion is given by

$$\ell = \frac{\Delta\theta_{deg}}{\left(\frac{N}{S^{-1}}\right)}, \quad (75)$$

where $\Delta\theta_{deg}$ is the angular rotation, N is the angular resolution from (72), S is the step resolution of the driver shown in table 52.

Therefore

$$\ell = \text{round}\left(\frac{\Delta\theta_{deg}S^{-1}}{N}\right), \quad (76)$$

The equation needs to be rounded as the steps can only be given as an integer.

15.6.2 Speed

The rotational speed can be given as either step/s or degree/s. For the Arduino the speed needs to be converted to a delay that will be used for each step it takes. Because of this both step/s and degree/s needs to be converted to a delay.

To convert from step/s to delay this expression is used

$$\tau = \left(\frac{1}{(\omega_{step}k)}\right)0.5, \quad (77)$$

where ω_{step} is the speed, k is the constant 10^{-6} .

To convert from °/s to delay this expression is used

$$\tau = \left(\frac{(N/S^{-1})}{(\omega_{deg}k)}\right)0.5, \quad (78)$$

where ω_{deg} is the speed, k is the constant 10^{-6} , N is angular resolution from (72), S is the step resolution of the driver shown in table 52.

At times being able to convert back from the Arduino values could be necessary for

debugging or if it is desirable to view both speed units at the same time.
To convert from delay values to step/s this expression is used

$$\omega_{step} = \left(\frac{1}{\tau k} \right) 0.5, \quad (79)$$

where τ is the delay, k is the constant 10^{-6}
To convert from delay values to °/s this expression is used

$$\omega_{deg} = \left(\frac{(N/S^{-1})}{\tau k} \right) 0.5, \quad (80)$$

Where τ is the delay, k is the constant 10^{-6} , N is angular resolution from (72), S is the step resolution.

15.7 Communication protocol

The communication to run the DM420A goes through the Arduino Mega 2560 to generate digital signals that the driver requires. To transfer type of parameter as well as the value for said parameter a communication protocol was set up so that all the information could be transferred in a single transfer. This protocol is set up to take the parameter type and value and use ":" as a break character to divide the values. If multiple parameters are transferred at the same time they will be divided with "&" character.

15.7.1 Parameter types

These are the parameters that are sent to the Arduino. The user will be able to choose other units to set parameters of speed and distance which will be converted to match these parameters before it is sent to the Arduino. Some parameters are restricted to a max value that is below what the system can process as any higher values would not be reasonable to conduct testing with.

Table 55: Parameters to the Arduino Mega 2560

Parameter	Description	Value range
1	Enable signal to turn on the motor bridge	0 for off, 1 for on
2	The direction the motor turns	0 for clockwise, 1 for counterclockwise
3	Speed of the motor in delay between steps	1 μ s to 10+ days
4	Distance to run in steps and starts a run	0 steps to 510 200 steps
5	Start calibration sequence	A numerical value will start it
6	Runs the motor continuously	0 for off, 1 for on

15.7.2 Example parameter string

Table 56: An example parameter string message to the Arduino Mega 2560

1:1&2:0&3:50&4:51200

As mentioned above, each type of parameter is split with "&", the table shows each sections of the string after this split has been made.

Table 57: Explanation of the parameter string given in 56

String section	Explanation
1:1	Enable signal, set high
2:0	Direction signal, set low for clockwise
3:50	Speed delay, set to 50 μ s. It will use 100 μ s per step. Two delays are used.
4:51200	Distance, set to 51200 steps

16 Unified Modelling Language

16.1 Document history

Table 58: Unified Modelling Language document history

Version	Date	Author	Description
1.0.0	19.03.2019	MBC	Document created. Added sections 2 and 3.
1.2.0	20.03.2019	MBC	Added section 5 and 6.
1.2.1	23.03.2019	JSS	Proofread.
1.2.2	26.03.2019	MBC & HS	Proofreading and corrections.
1.3.0	13.05.2019	MBC	Added explanation of extend and include dependencies.
1.3.1	17.05.2019	MBC	Proofreading and corrections.

16.2 Introduction

This section will explain what Unified Modelling Language (UML) is and explain two simplified examples of UML diagrams. It will give a light introduction to UML to give the reader insight in what it is and how it benefits our project. The section will not cover every aspect of UML but instead give a simplified top view of it. The reasoning behind this is that it will prepare any reader who does not have any previous knowledge about UML and the CPS teams approach to UML.

16.3 What is Unified Modelling Language

“The unified Modelling Language is a visual language that provides a way for people who analyse and design object-oriented systems to visualise, construct and document the artefacts of software systems and to model the business organisations that use such systems” [5].

This means that it can be used to develop software systems, but unlike a programming language UML is a high-level specification language used to describe the functionality of the system. It consists of a set of elements and rules that describe how to use it and is useful to plan and develop software systems.

UML can provide a graphical representation of how the system works in multiple levels of complexity. By using the different elements that UML offers, one can describe nearly every subsystem of the product. This can be particularly effective in explaining software related artefacts to a customer who may not have a software related background.

16.4 Use cases

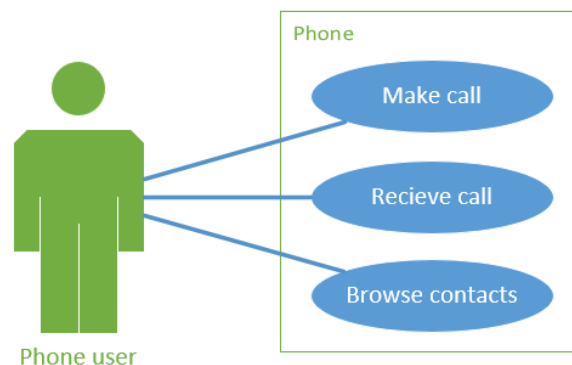


Figure 108: Use case example

Figure 108 represents a simplified example of a use case diagram of a phone. The phone user is called an actor and is either a person or another system that interacts with use cases. The blue bubbles are use cases and in this example, refer to upper level functions of the system. The green square that encapsulates the use cases is the subsystem. In this case the subsystem is the phone and it aids in visualising the functionality of the phone and

how a user may interact with it.

The arrow with the label "Include" is called a dependency and describes the bubbles relations to each other. In this case an include tells us that the bubble pointed to is a direct consequence of the parent bubble. If the parent bubble action is done, then the included bubble will immediately follow and be done. Similarly, the arrow labelled "Extend" describes a similar dependency, but where the include dependency is absolute, the extend dependency is optional. This means that the bubble that is extended is not something that has to be done, but optional.

16.5 Sequence diagrams

Another useful element of UML are sequence diagrams. They are used to visualise the interaction between different classes/objects in the software.

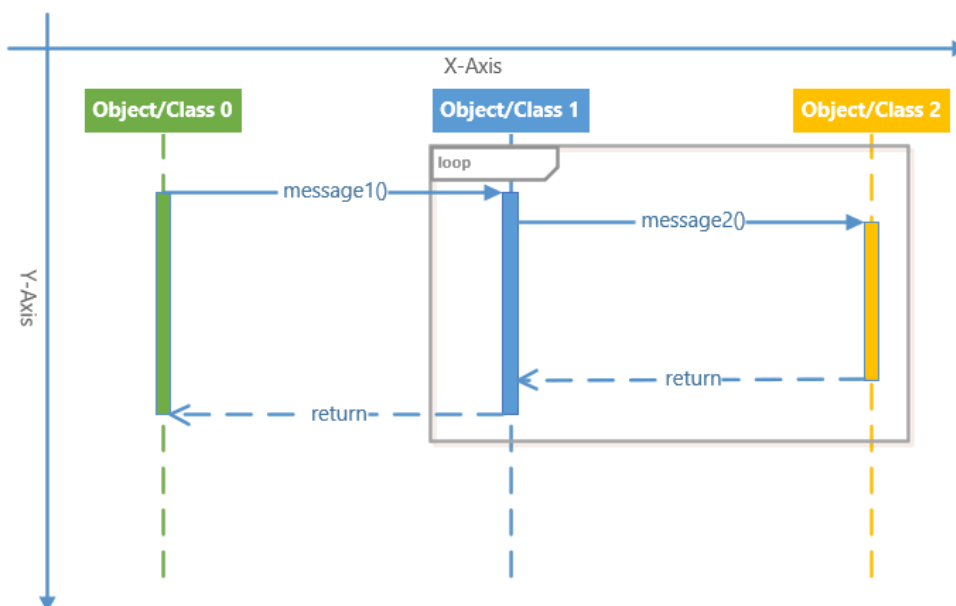


Figure 109: Sequence diagram example

In this diagram the Y-Axis represents time elapsed, and the x-Axis holds the different lifelines. The boxes colour coded by green blue and yellow are lifelines in this diagram. They represent the participation of said class/object in this interaction.

The boxes labelled "Object/Class" with dotted lines, are called lifeline notation. They represent the participation of said object or class in an interaction. Having the dotted lines connected to the lifeline shows that the class or object is already created and will continue to be so after any interaction is done.

The solid boxes covering the dotted lines represent events. If a message is sent to object 1 an event occurs. Object 1 starts processing and performing tasks and when done notifies

that it is done.

The grey box is called a frame. The frame specifies events or actions that happen inside this frame. In figure 109 there exists a loop frame. This means that the interactions inside this frame are redone until the loop is done.

16.6 Further reading

This has been a light introduction to what UML, use cases and sequence Diagrams are. For a further and more in depth explanation of UML a recommendation is the Object Management Groups website [45].

17 CPS software planning

17.1 Document history

Table 59: CPS Software planning document history

Version	Date	Author	Description
1.0.0	22.03.2019	MBC	Document created, and added subsection 1 to 4.
1.1.0	24.03.2019	MBC	Added subsection 5 to 8.
1.1.1	25.03.2019	JSS	Proofread.
1.1.2	26.03.2019	HS & MBC	Proofreading and corrections.
1.2.0	13.05.2019	MBC	Updating UML diagrams and their description.
1.2.1	14.05.2019	MBC	Finalised UML diagram descriptions.
1.2.4	17.05.2019	MBC	Proofreading and corrections.

17.2 Introduction

To model the software system of the CPS bachelor project, planning is necessary. It was given a set of requirements and they were analysed, in order to plan the software system that was to be developed. This section will explain the initial concept as well as explore the current software concept that was chosen to be further developed.

17.3 Initial challenge

Initially there was a challenge in regard to how to model the system for KDA. There were no requirements specifying software functionality, but instead a task summary which included a somewhat broad task. This task stated that a test station with a convenient user interface to perform functional testing and calibration of the sensor, was to be developed. Different ideas were discussed about how to model the software for the test station.

17.4 First software concept

Different ideas were explored. First we considered having the test station software stationary on a local computer such as a Raspberry Pi3, shown in figure 110. This concept would eliminate the need for any user to bring a laptop to perform functional testing and calibrating of the sensor.

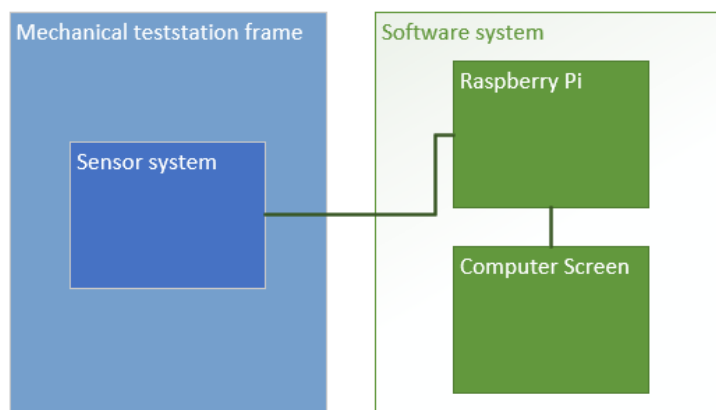


Figure 110: First software concept

This concept was discarded as the complete system concept evolved. To be able to test in different locations the test station needed to be portable and having a stationary screen would contradict this.

17.5 Second software concept

Iterating from the first concept the second concept evolved. The idea behind this concept was to make it as easy as possible for users to use the test station. It would be built upon a server client architecture, and the graphical user interface would be accessed by a website. This would eliminate the need to install software or transport any test log from a

test station to personal work laptop. The possibilities of an app were discussed as it would eliminate the need for any laptop or desktop to perform a functional test or calibration from the users phone.

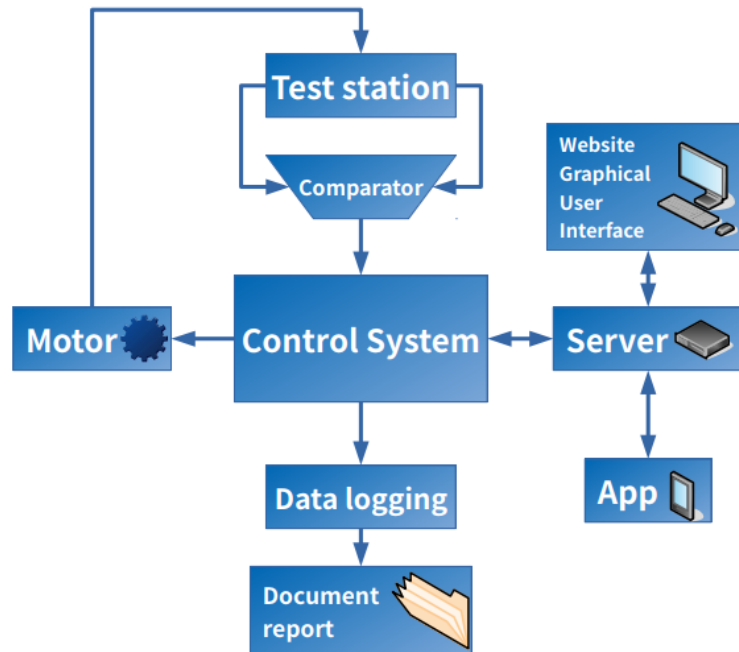


Figure 111: second software concept

This concept was also discarded after conversation with KDA as there were multiple problems with this concept. KDA has strict security policies and to prevent any potential IT security breaches, none of KDA's test equipment are connected to the internet. Bluetooth technologies are also regulated due to the strict security and this resulted in the CPS having to exclude the app as well.

17.6 Third software concept

After iterating throughout the previous concepts, the third concept was developed. It is intended to run on a user's work laptop thus does not violate any security precautions as well as having great portability. After the second software concept iteration the team received updated requirements and software requirements had to be implemented. The software team could start to plan the test station software.

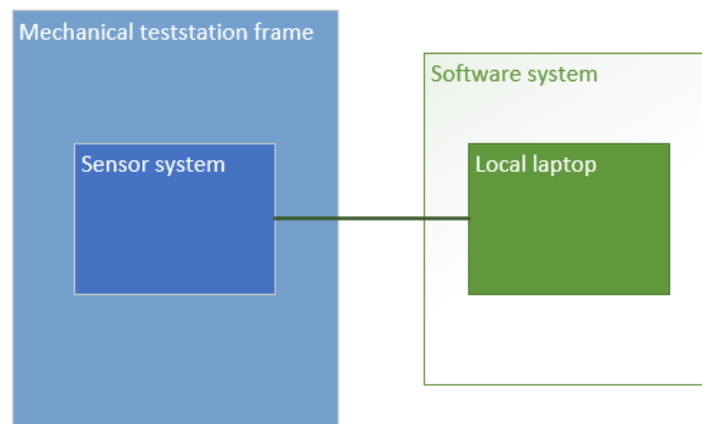


Figure 112: Third software concept

The concept is based around a layered software architecture, as shown in figure 118, and is intended to run on a local laptop, preferably the users work laptop. The users laptop is already cleared for usage in KDA, thus the only remaining security issue is to clear the software once it is developed. The stakeholder requirements 1.19 to 1.23 in table 18 state that the test station shall be able to display data, run with different parameters, log data which enables for later extraction, as well as include "run" and "motor enable" features. The following Use Case diagrams as shown in the next section, was created.

17.6.1 Use Cases

As mentioned in in section 16, the usage of Use Cases and other UML diagrams, has the purpose of describing the interaction between users, systems, and subsystems. The following Use Case diagrams was developed to understand how the software system would behave and to use it as a template. The Use Cases was designed with the stakeholder requirements in mind. The requirements 1.19 to 1.23 from 18 describe functionality of the test station software. The base Use Case diagram illustrated in figure 113 was then created.

This Use Case features functionality that is explained general enough to not allow for concept discussions on how to implement it, but at the same time specific enough to understand what functionality is modelled.

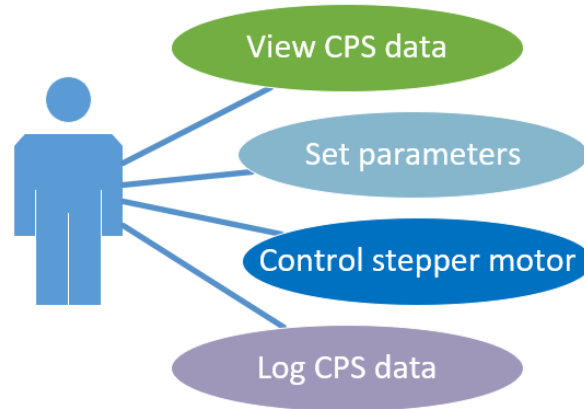


Figure 113: CPS software base Use Case

The green Use Case states "View CPS data". This Use Case stems from stakeholder requirements 1.10 and 1.19, which state that the position shall be based on change in frequency and provide an accurate and analogue position value, as well as that the test station shall through a graphical user interface be able to show change in degree and speed. The next Use Case states "Set parameters" and derived from stakeholder requirement 1.21. This requirement described different parameters that the test station was to include. The ability to change these parameters allows for an extensive test station and variety in testing. Through the test station the user also needed the ability to control the rotation of the sensor inside the test station. It was decided in a conversation with the stakeholder that the test station was to utilise a stepper motor with a high degree of accuracy and repeatability. Therefore the dark blue Use Case states "Control Stepper motor". The last Use Case states "Log CPS data" and derives from stakeholder requirement 1.21. This requirement describes that the test station needs to log test data with all of its parameters in a format which enables for extraction at a later time. As stated above these descriptions are general enough to understand what system function needed to be performed, and specific enough to not confuse it with other functions. After creating the base Use Case from figure 113, a decision to further describe the different Use Case in detail and add more specific functions, was made. The following Use Case diagrams illustrates the different Use Cases explained above with more specific functions and dependencies.

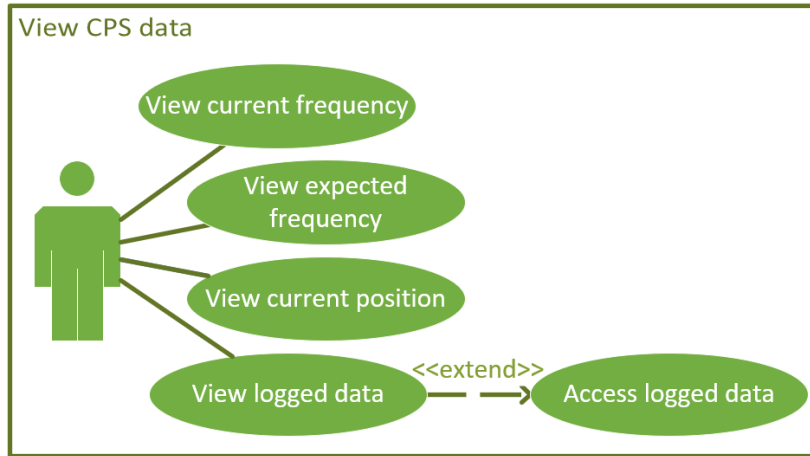


Figure 114: View CPS data Use Case

The first one is the "View CPS data" Use Case. As seen in figure 114 The user can view the actual frequency value from the CPS and also see the expected frequency. In addition to this the user can also view the current position of the rotor. Lastly the user can also view previously logged data, and access the data if desired.

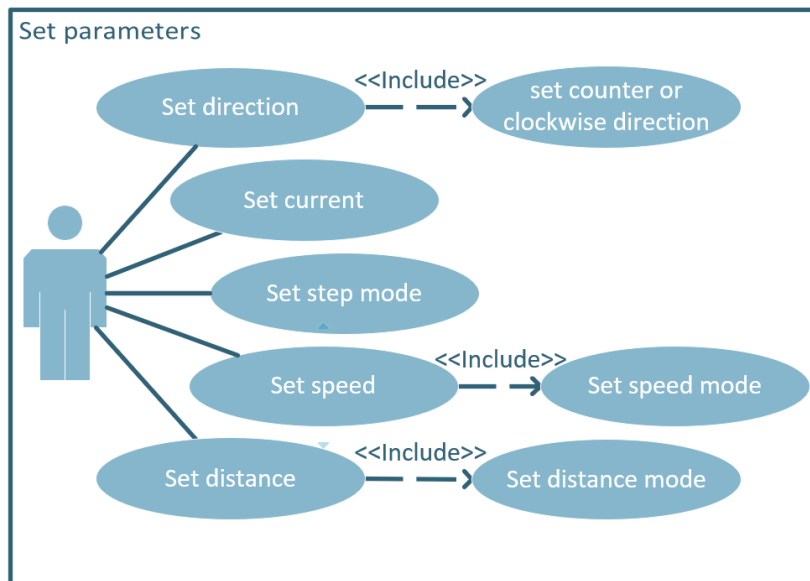


Figure 115: Set parameters Use Case

The next Use Case "Set parameters" is illustrated in figure 115. This one is more complex because of the different parameters that needed to be implemented due to the requirements. The user is able to set the direction of the stepper of motor. This is described as an include as it is something that has to happen if the user sets the direction. The user is also able to both set the current parameter and the step mode parameter. The last two parameters; speed and distance both have includes to describe that the mode in which the user sets speed and distance has to be specified. When a user changes the speed

of the stepper motor, it is necessary to specify if the user wants to use steps per seconds or degrees per second. This applies to the distance parameter as well, as it is possible to both decide to move in number of steps or degrees.

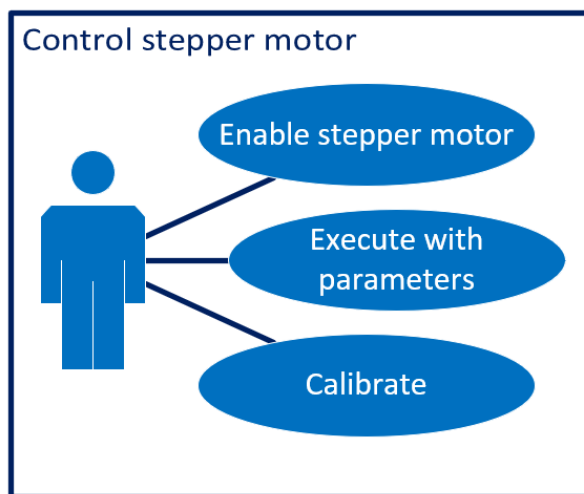


Figure 116: Control stepper motor Use Case

The next Use Case is illustrated in figure 116 and it is not as complex as earlier Use Case. Here the user can enable the stepper motor bridge to allow for movement. The user is also able to execute with the selected parameters. Lastly the user is able to perform calibration of the CPS. This is necessary as calibration sets the blueprint for the degrees and its values in frequency. The comparison of actual frequency and expected frequency is between the real-time data and the data collected during the calibration.

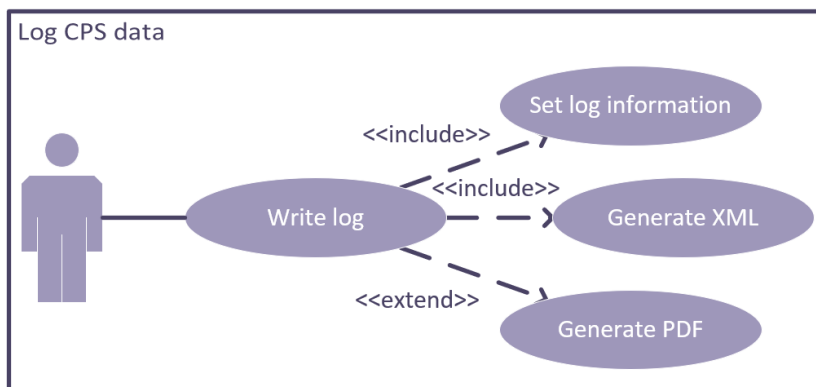


Figure 117: Log CPS data Use Case

The last Use Case is illustrated in figure 117. This one is special as there are two includes, and one extend dependencies. When a log of CPS data is written the user has to set log data, such as different parameters, and different project information such as project name, test ID, etc. When an log is written a XML file is generated with the data available. Lastly the user has the opportunity to specify if generating a PDF test report is desirable, but this is as mentioned optional hence the extend dependency.

17.7 Third software concept architecture

17.7.1 Why CPS test station utilises software architecture

Developing software systems without a certain structure or architecture can often result in an unpredictable situation. As the source code increases in complexity, it can prove difficult to change parts of the code as one change might effect more of the system than intended. Results of this can be time consuming debugging that could have been avoided. Another reason to utilise software architecture is the handover to the customer. The customer may want to maintain or further develop the system and by having an unorganised code, this can be complicated.

This can be prevented by structuring the software by what is called software architecture. There are many types of software architectures that are widely used in software development, and choosing the wrong architecture for the system can complicate the development instead of helping to structure it. An example of a software architecture is the server client type which the second concept was based on (see section 17.5).

17.7.2 Choosing the right architecture (Software architecture)

The architecture chosen was the layered architecture, and it is based upon splitting components into horizontal layers as shown in figure 118. Each layer only communicates with neighbour layers. An example of this is how the presentation layer illustrated cannot communicate directly with the functional layer, but instead communicates trough the control layer. Each layer has a specific role, or responsibilities in the software system. An example is that the presentation layer does not need to know how to control the driver, but instead only has to present the user the option to control it.

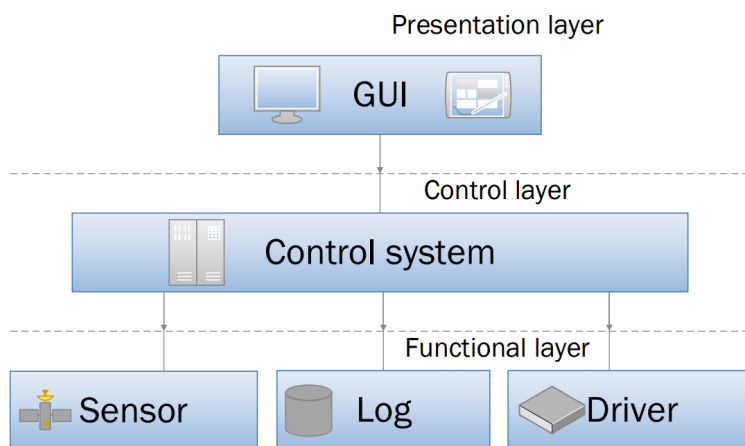


Figure 118: software concept architecture

The presentation layer only handles the graphical display of data, or sends instructions to the control layer. This functionality is derived from STRQ 1.19 in table 18.

The control layer is responsible for handling the instructions from the presentation layer

and initiates other components such as the logging of data, or driving the motor with different parameters, as stated in the STRQ 1.21 in table 18.

The functional layer is where the system functions are coupled with the corresponding hardware/software. Here the circuit that makes up the contactless position sensor is interfaced with the software enabling data reading from the sensor. Other data about parameter, angles, speed etc is processed and logged in the desired format (See STRQ 1.20 in table 18), and the motor to drive the rotation of the rotor is interfaced with the software to control different parameters (See STRQ 1.21 in table 18).

17.7.3 Objects/classes interaction (Sequence diagram)

As previously mentioned in section 16, sequence diagrams illustrate the interactions between the classes or objects in the system. It was decided to incorporate this type of diagram to further develop an understanding of how to model the software and to enforce the Use Case diagrams. There are in total four sequence diagrams showcasing the interactions between hardware and software in different scenarios. The usage of the layered architecture described in section 17.7.2 is further reinforced by the following figures by having every class or object not skipping its parent or child layer to exchange data. Everything from the graphical user interface goes through the control system, and the same applies the other way around. Everything going to the user interface goes through the control system first.

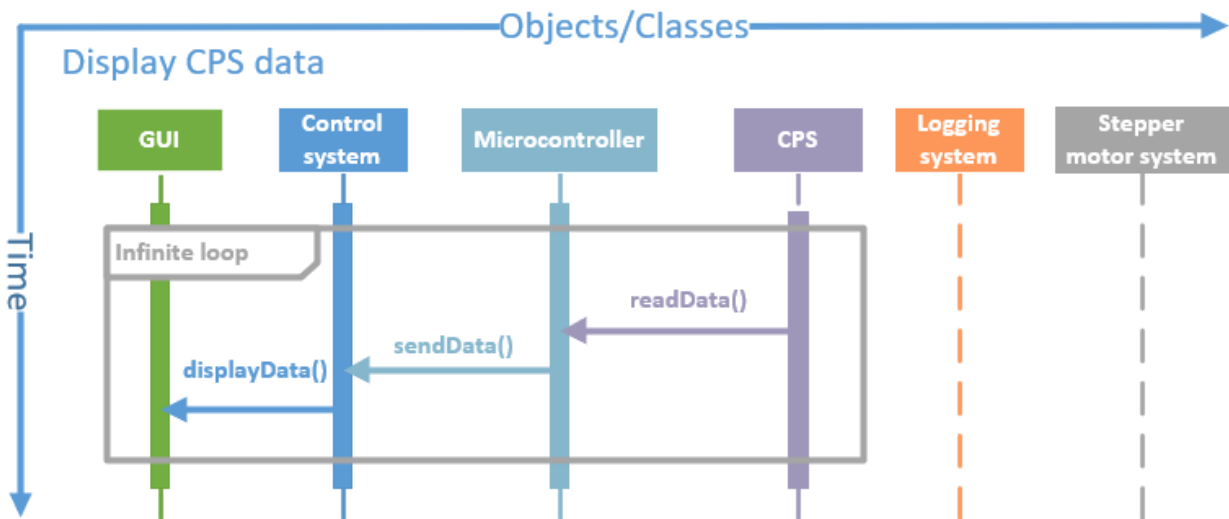


Figure 119: Display CPS data sequence diagram

The first sequence diagram illustrated in figure 119 showcases the sequence of interactions while displaying CPS data on the GUI. This sequence starts at the CPS itself, which is always pouring out analogue data to the microcontroller. Following the arrival of new sensor data, the microcontroller packets the data in an appropriate manner and sends it to the software via the control system. Here the data packet is disassembled and dissected and displayed on the GUI. All of this will happen continuously, unless a command to do

something else has been issued, thus this sequence is placed inside a infinite loop. Although being labelled infinite loop, it can be broken as mentioned.

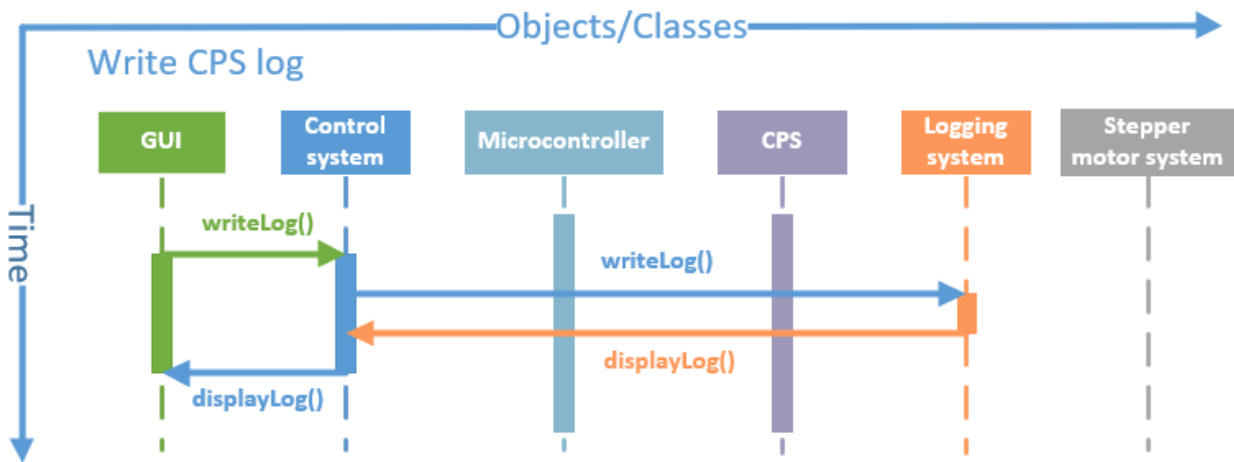


Figure 120: Write log sequence diagram

The second sequence diagram is illustrated in figure 120, and the number of interactions is somewhat smaller due to the nature of the interaction. Writing a log is something that happens after the user initiates it or after an calibration run. This sequence starts by the control system receiving an initiate message. The control system then invokes logging system and provides it with the necessary information to write a log. When writing the log is finished it is displayed in the GUI in its appropriate folder, described in the folder structure section 21.

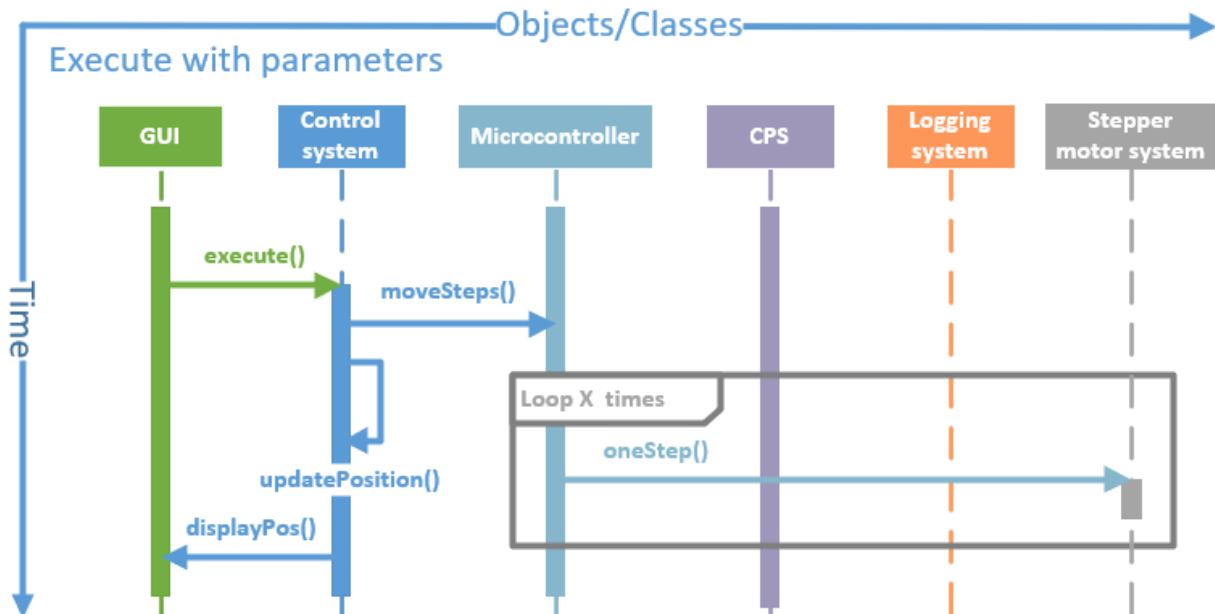


Figure 121: Execute with parameters sequence diagram

The third sequence is illustrated in figure 121 and showcases the sequence interactions after the user presses the run button. This sequence starts by having the user pressing the run/start button. The control system then pieces together a data packet containing the necessary information such as how many steps, and at what speed. This is then sent to the microcontroller which executes this by looping and performing the operations needed. However, after the control system sends the data packet it will update the position data and then display it to the GUI. This is information that is controlled through the control system thus not having to pass it to the microcontroller and back.

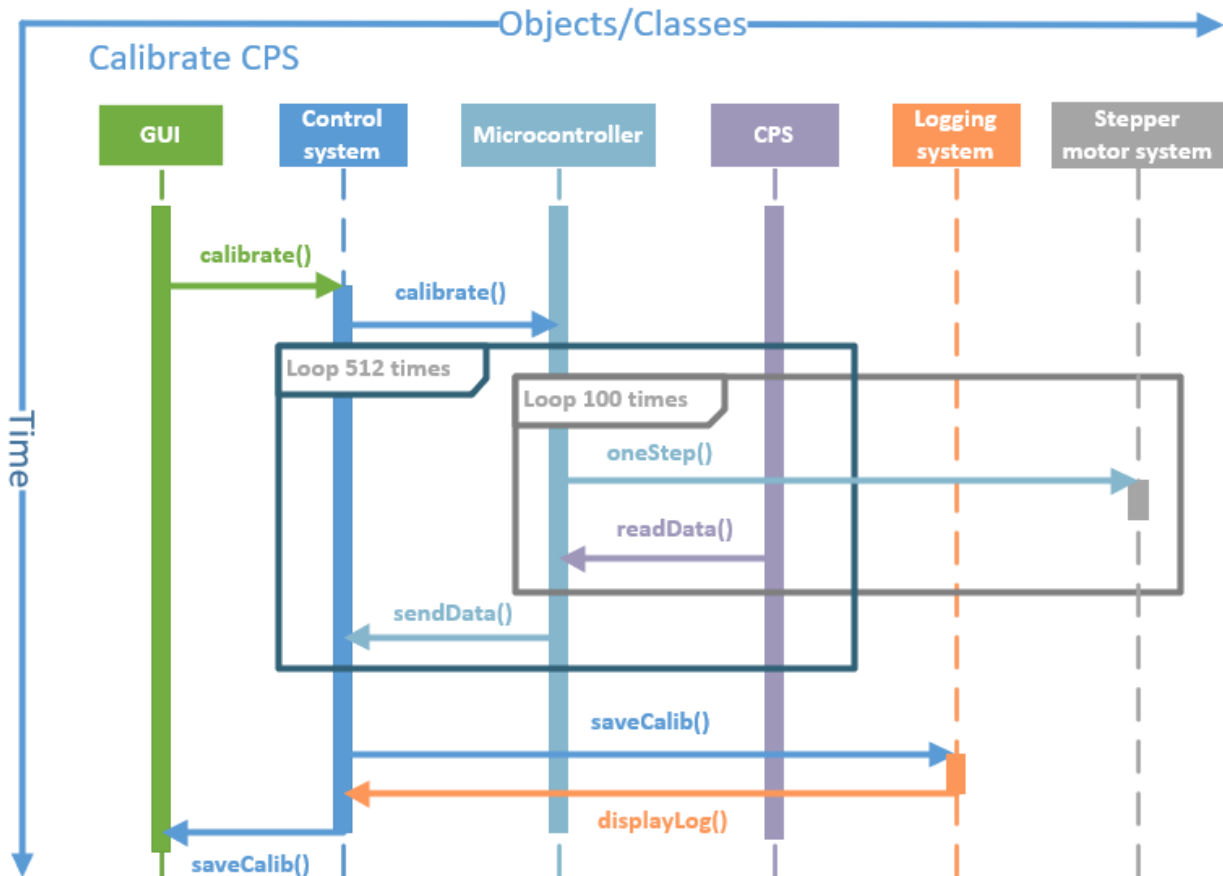


Figure 122: Calibrate sequence diagram

The last sequence is the calibrate sequence, and this one incorporates the writing log and the execute with parameters sequence. This is the sequence that lays the blueprint for further testing of the CPS and is therefore important. The previous sequences such as displaying data and writing logs, are dependent on the calibration in order to provide max value information. Viewing the analogue data produced by the CPS, has higher value if its possible to compare it to an expected value. This sequence starts by having the user pressing a calibrate button. The control system then receives this information and initiates the calibration process. Here there are two loops. One that contains the microcontroller and stepper motor system, and one which contains the control system and microcontroller. The inner loop takes one step and stores the data of the CPS corresponding to each step.

This is done 100 times and this data set is then sent to the control system. Getting 100 analogue data sets is also looped 512 times. This is because there are a total of 51200 steps at the highest resolution of the stepper motor, and the calibration needs to measure the analogue data at every step. When the calibration is done it is logged and then made accessible to the GUI.

18 Detecting frequency

18.1 Document history

Table 60: Detecting frequency document history

Version	Date	Author	Description
1.0.0	06.05.2019	MBC	Documented created. Added subsection 1 to 3.
1.1.0	08.05.2019	MBC	Documented created. Subsided introduction, and other sections.
1.2.0	12.05.2019	MBC	Added circuit diagram.
1.3.0	14.05.2019	MBC	Added interface diagrams and reference to test.
1.3.1	17.05.2019	MBC	Proofreading and corrections.

18.2 Introduction

Detecting the frequencies in the different coils is an important aspect of the CPS project. Stakeholder requirement 1.10 from 18 state: ” *The CPS shall based on change in frequency provide an accurate and analogue position value.* This means that a solution to the problem of how to detect a frequency, had to be discovered. This subsection will explain the approach to detect the different frequencies in the different coils from the CPS sensor.

18.3 Frequency detection initial thoughts

Initially the team wanted to recreate the analogue signal in the eventual software, to be able to get precise measurements. As mentioned in the introduction above, the it was concluded that frequencies of 6 digits were needed to be detected. To recreate these signals an ADC (analogue to digital converter) was needed. This ADC would have to have a set of specifications to accurately represent the analogue signal as a digital signal.

By using the prototype received by KDA, it was discovered that frequencies over 1 MHz gave an unstable and electric noise-filled signal, while frequencies under 1 MHz gave a signal with a higher tolerance for electric noise 24. Thus, the team needed to detect frequencies which consisted of 6 digits, for example a frequency of 850 KHz.

According to the Nyquist criterion, any sampling rate less than twice the highest frequency of the analogue signal, can result in a false representation of the analogue signal [7]. This means that it was needed to sample with at least twice and some more of the highest frequency that the analogue signals would have. It was then concluded that having a sampling rate of at least 2 MHz would be sufficient. To recreate the analogue signal with the desired precision, the CPS then wanted a resolution of 12 bits. This way the team could represent 4096 voltage values and reach the desired level of precision.

Based on these criteria, concepts that could perform the desired task at the desired level of precision was researched. As the different concepts where researched it was found that recreating the analogue signal, was not the only solution.

18.3.1 Frequency detection concept - DAQ

This concept explores the idea of using already built/created equipment, to allocate time to other aspects of the project. The team researched the possibilities of buying a dedicated data acquisition device, that could be interfaced with a laptop and the CPS software as illustrated in figure 123. This concept would allow for focusing on other tasks, as most of the groundwork such as converting the analogue signal and sending it to a laptop, would already exist. Searching such devices was harder than anticipated and led to trouble finding a device that could be used. Either the sampling rate would be insufficient for an accurate recreation of the analogue signal, or the resolution would be too small to represent the binary value of the frequency. Price of the DAQ devices was also a challenge as most of the DAQ's that the team found would cost about 60% of the allocated money given for this

project, which was needed to cover the cost of milling the test station frame. After discussions with the external advisor, a less complicated concept was developed.



Figure 123: Frequency detection DAQ concept

18.3.2 Frequency detection concept - USB oscilloscope

After discussing other concepts to detect frequency, one member of another project group (Martin Larsen) tipped about the USB oscilloscopes from Diligent. The oscilloscopes are the model called Analog Discovery, and they come with accompanying software.

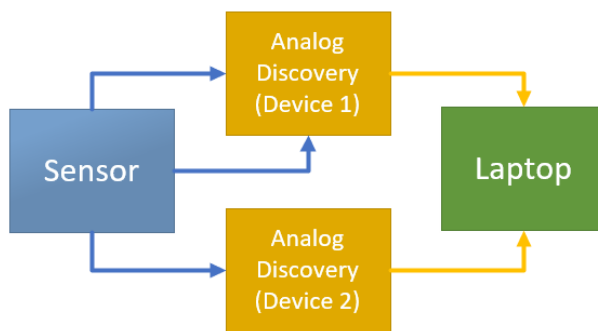


Figure 124: Frequency detection USB oscilloscope concept

This software allows you to create scripts in different programming languages, as well as having an accessible API. This API made it possible to add library and access its members. This concept is illustrated in figure 124.

In listing 2, it is

illustrated how to connected a Analog Discovery device in C++ by utilising its software API.

```

HDWF hdwf;
STS sts;
double* rgdSamples;
int cSamples;
int cChannel;
char szError[512] = {0};

std::cout << "Open automatically the first available device\n";
if (!FDwfDeviceOpen(-1, &hdwf)) {
    FDwfGetLastErrorMsg(szError);
    std::cout << "Device open failed" << szError;
    //return 0;
}
  
```

```
std::cout.flush();
```

Listing 2: Accessing Digilent Waveform API in C++

Although receiving this tip, there were several problems with the concept. First, the oscilloscope has two channels, while the CPS has three coils that need to be read. This results in not having to utilise two devices. Having two devices compromises the desire to make the test station as portable and free of devices to plug in, as possible. Another problem was the possibility of recreating the desired signal. Although having sufficient sampling rate at up to 100 MSPS, 14 bits is not as high as preferred [13]. After discussing possibilities another concept was discovered.

18.3.3 Frequency detection concept - FPGA

One of the earlier concepts to detect frequencies was based around an FPGA. An FPGA is a logic device which consists of programmable switches and generic logic cells [8]. With many logic cells and fast internal clock speeds, they can be programmed to perform a diverse number of tasks. These tasks can range from creating simple logic circuits to create fairly complicated CPUs.

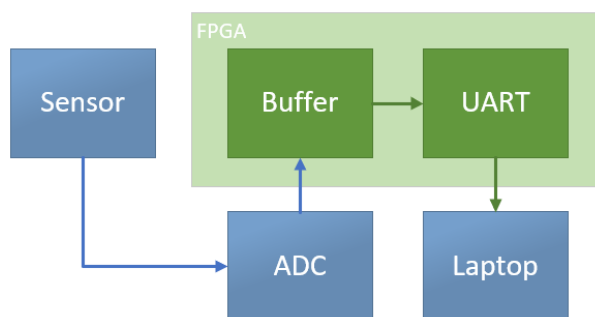


Figure 125: Frequency detection FPGA concept

This concept included several tasks the FPGA would have to perform to be able to be utilised as a viable concept. First, the FPGA would need to store the analogue data sets for recreating the analogue signal to digital. Then it would need to be able to transfer the data sets to software for processing. The team discussed the possibility of using an external Analogue digital converter connected to the FGPA and then saving the analogue data in a buffer. This buffer would then be sent to the software through a USB UART bridge. A UART is a universal asynchronous receiver transmitter device that enables for serial communication. This concept is illustrated in figure 125.

Using the BASYS 3 FPGA board was considered viable because of its specifications and availability. With internal clock speeds exceeding 450MHz and 1,800 Kbits of fast block ram [14], the BASYS 3 FPGA would suit the CPS test station well. Depending on which ADC that would be utilised with this concept, this board could be used with the CPS as the frequencies in the CPS coils would not exceed 300MHz.

This would be a viable option, but the CPS project has a time constraint of 5 months, and

this concept required that the buffer, UART, and the interfaces needed would be built from scratch. At the time working with the detection of the frequencies started, there was not sufficient time left to implement the FPGA solution from scratch. During development several time-consuming problems was encountered. Problems that took a longer time to solve, than first anticipated. This combined with the remaining time left, led to the choice of another concept.

18.3.4 Frequency detection concept - Counter Integrated Circuit

As mentioned earlier in subsection 18.3, and subsection 18.3.1, a lightweight and simplified concept was developed after talking to the external adviser. This concept is illustrated in figure 126

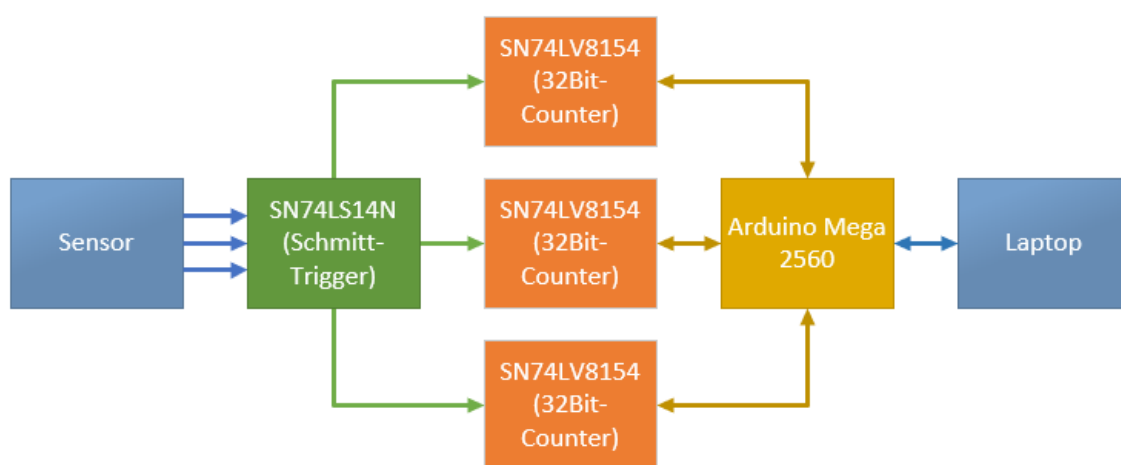


Figure 126: Counter integrated circuit concept

This concept is different from the other concepts whereas it does not revolve around sampling and quantifying the signal, but instead counting the number of rising edges over a period of time. Given that the period of time spent counting could be repeated with marginal errors, this could be used to represent a value close to the original sine wave. With this concept the device requirements shift from sample rate and quantization resolution, to how fast the counter can count and how high. With an ideal circuit, counting the rising edges of the sine wave over 1 second would result in the number counted representing the actual frequency.

To start and stop the counter in question an Arduino Mega 2560 microcontroller was utilised. In this concept communication speed is not prioritised, thus enabling the use of the arduino. The counter works independently with a minor exception of getting start and stop signal, which can be done through the arduino. The value extracted from the counters is then sent over serial communication to the software on the users laptop. With a clock speed of 16MHz it was concluded that it was sufficient.

Figure 127 illustrates how this concept is interfaced with the software. This diagram

illustrates viewing of CPS data and then moving to a new position in order to read new sensor data.

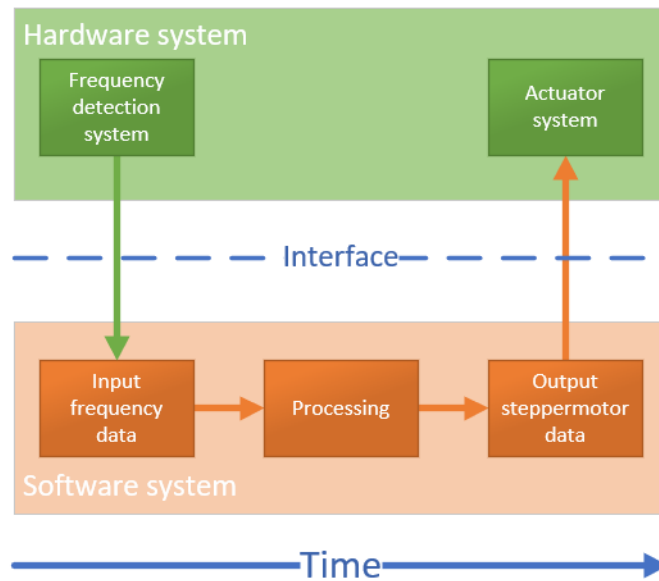


Figure 127: Interface diagram

Building the counter on an FPGA was also considered but found that an integrated circuit (IC) chip could serve as a viable option. The counter could be controlled through an arduino in which had already been utilised in earlier work, see section 29. After researching possible counter circuits, a possible IC was found. This was the SN74LV8154 IC with dual 16-Bit binary counters [54], illustrated in 128.

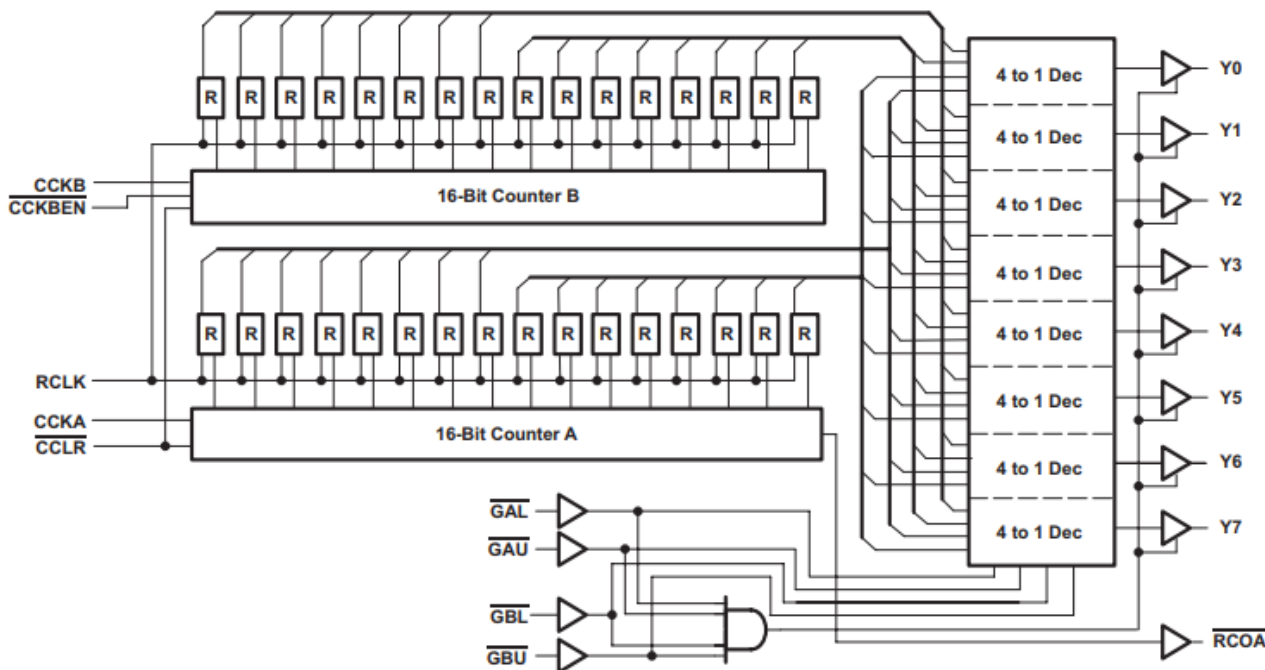


Figure 128: Counter integrated circuit [54]

This IC consists of two 16 bits counters that can be connected to a 32 bit counter. This is done by connecting the RCOA pin to the CCKBEN pin. It also features an internal storage register to save the output of the counters at a desired time. With 8 outputs the counted data has to be extracted by cycling through the high and lower bytes of both the counters. This is done by driving logic high and low to the GAL, GAU, GBL and GBU. The counters also feature a synchronous clear, which can be used to reset the counter before each counting. Due to its availability and price the decision to order this counter and continue testing it was made. However there were some concerns using this counter circuit, which were confirmed when testing the IC. When counting a square wave analogue signal the counter worked perfectly, but once a sine wave was utilised, the rising edge was not steep enough to trigger the counting consistently. This problem resulted in utilising a Schmitt-Trigger as illustrated in figure 126. The green box with the label *SN7LS14N* (*Schmitt-Trigger*) is an IC with Schmitt-Triggers. This is an active circuit which holds the current state of the signal until a certain threshold has been reached. This means if an analogue high is changed to analogue low, the Schmitt-Trigger does not change to low until its voltage threshold is reached. The same goes for the other way around.

Figure 129 illustrates how the counters are connected to the arduino and how they share the same control signals.

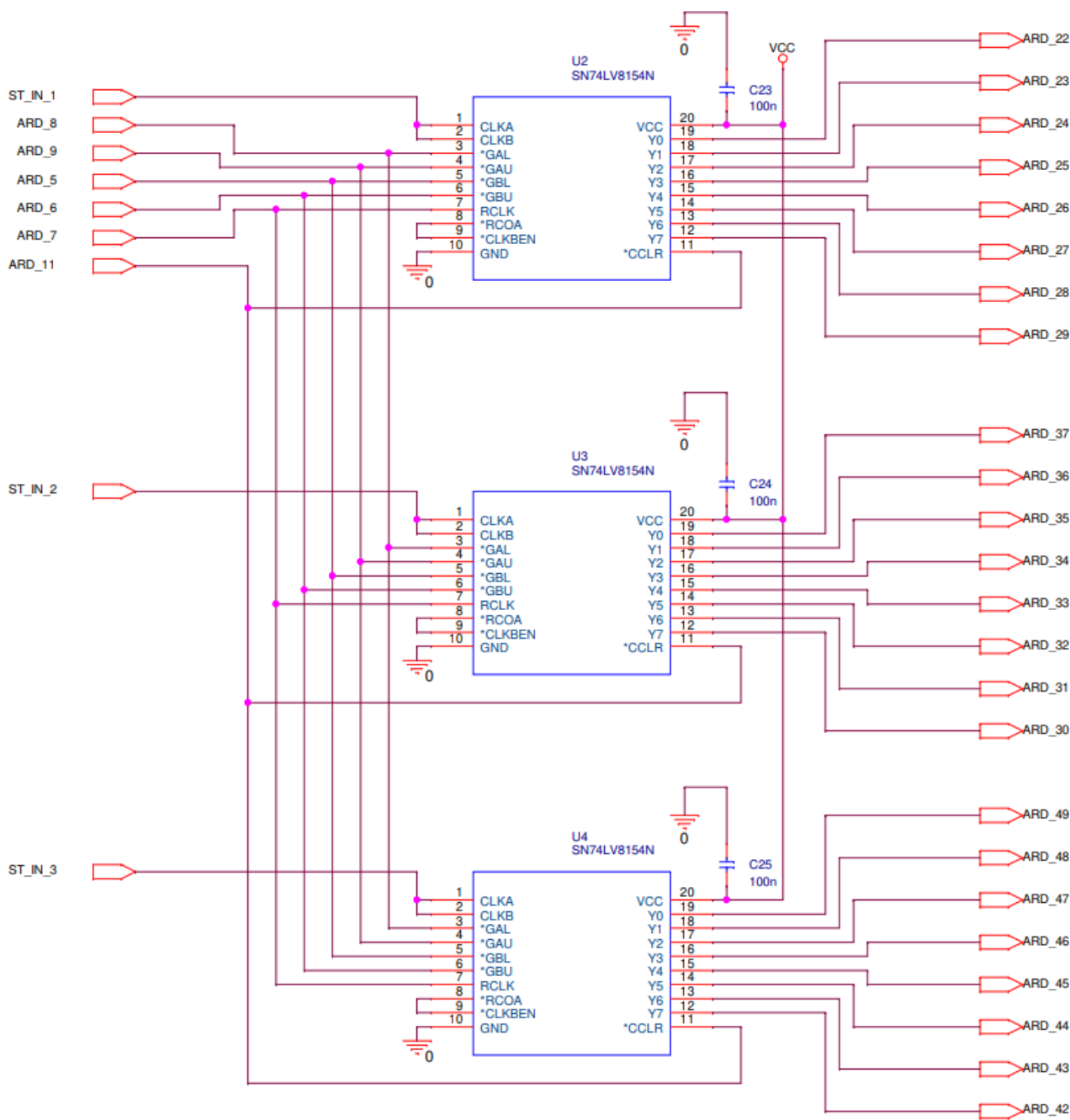


Figure 129: Circuit diagram integrated circuit diagram

Through testing (24.4), it was discovered that the SN74LV8154 was precise and could be utilised in the test station. This test concluded that using an arduino would be possible although not providing an ideal environment for data acquisition, it would be enough to provide the proof of concept needed. This reasoning combined with the limited time left on the project led to the team continued developing this concept with said components.

19 Frequency calculation

19.1 Document history

Table 61: Frequency calculation document history

Version	Date	Author	Description
1.0.0	11.05.2019	AR	Created document.
1.1.0	17.05.2019	AR	Changed name of subsection and added Self resonance.
1.1.1	17.05.2019	MBC	Proofreading and corrections.
1.1.2	22.05.2019	CPS team	Proofreading and corrections.

19.2 Introduction

This section is going to discuss the choice of frequency and frequency span. This section will also describe how the rotor affects the circuit and output frequency.

19.3 Explanation of frequency span

The CPS team tested how the rotor affected the inductance in the coils in test 24.3, and how precise the counter was in the test 24.4. With this information, the frequency span needed to detect change in degrees could be calculated. The frequency span is the difference between the highest and lowest frequency that the oscillator produces. The oscillator will produce a higher frequency when the copper plate is covering the coil, see section 73. With a bigger frequency span, the measurement on the counter circuit can be more precise, hence it is desirable with a high frequency span.

The inductance in the coil is 110 μH without any interaction from the rotor. When the coil is fully covered by the rotor the inductance is approximate 55 μH . The expression for frequency is given by (54).

From table 54 in section 15.5.2, the highest resolution of the motor is 0.00703125° per step. With this it is possible to calculate how much the frequency is changing with per step. One coil on the stator is covering 90 degrees out of 360 degrees. Hence

$$\frac{51200 \text{ steps}}{360^\circ} \cdot 90^\circ = 12800 \text{ steps.} \quad (81)$$

With 51200 steps on the motor, 12800 steps are required to go from covering 0% to 100% for one coil.

The counter can detect a change of approximate 5 Hz. Taking account for some variation and margin of error it was assumed to reliably count every 10 Hz. To get an accuracy of 0.01 degrees it was required to get 10 Hz difference for 1 step. Hence the required frequency span is,

$$F_{span} = (12800 \text{ steps})(10 \text{ Hz}) = 128000 \text{ Hz.} \quad (82)$$

A calibration is a full 360° rotation where the main goal is to look at every position and see a change for 4 out of 5 steps to achieve the desired requirement. See table 62.

Table 62: Speed table

Step	Angle
1	0.00703125°
2	0.0140625°
3	0.02109375°
4	0.028125°
5	0.03515625°
6	0.0421875°
7	0.04921875°
8	0.05625°
9	0.06328125°
10	0.0703125°
11	0.07734375°

It is possible to use 1 second to count the rising edges in the signal, which would result in an approximation of the frequency. Then the rotor needs to move for 20 ms to rotate 1 step, and 20 ms to stabilise at this position. Totalling the time spent,

$$(1 \text{ s} + 20 \text{ ms} + 20 \text{ ms})51200 = 53248 \text{ s}, \quad (83)$$

and convert seconds to hours,

$$\frac{53248 \text{ s}}{60(s/m)60(m/h)} = 14.79 \text{ Hours} \approx 15 \text{ Hours}, \quad (84)$$

it is going to take approximate 15 hours for one calibration.

This would be acceptable for a one time calibration run, but to properly test the sensor it must be calibrated in various conditions. Therefore this is not desirable.

To reduce the calibration time, it is possible to sample faster than every second. The problem with this is that the counters will count less then what the real frequency is. The counters also cannot count decimal points. That disqualifies the option to just multiply the sum to get the real frequency. Lowering the count time also lowers amount of data. A solution to this is to increase the frequency span to account for the loss in data.

With an frequency span of 200k it is possible use 0.65 s for instead for 1 s on each step.

$$(200\text{k Hz})(0.65 \text{ s}) = 130\text{kHz}, \quad (85)$$

where 130 kHz is the relative frequency span when counting at 0.65 seconds. This give use the possibility to calibrate using 0.65 s and will reduce the calibration time,

$$(0.65 \text{ s} + 20 \text{ ms} + 20 \text{ ms})51200 = 35328 \text{ s}, \quad (86)$$

and convert seconds to hours.

$$\frac{35328 \text{ s}}{60(s/m)60(m/h)} = 9.813 \text{ Hours} \approx 10 \text{ Hours} \quad (87)$$

A concern with the formula for frequency in a Clapp oscillator is not linear. So from step 0 to 1, there is significant less change in frequency than from step 12799 to step 12800. This can lead to a situation where the sensor cannot read the values before the rotor has reached a certain threshold.

To prevent this, it is desirable to get a greater frequency span than 128000 Hz. These calculations are in an ideal scenario and can vary in a physical model. The frequency for each step is given by,

$$F = \frac{1}{2\pi\sqrt{L_{covered}C}}, \quad (88)$$

where x is the number of steps and $L_{covered}$ is how much the inductance get reduced when the coil is fully covered,

$$L_{covered} = L - (55 \cdot 10^{-6}) \left(\frac{x}{12800Step} \right). \quad (89)$$

An example on this is,

$$L_{covered} = (115 \cdot 10^{-9}) \text{ H} - (55 \cdot 10^{-6}) \text{ H} \left(\frac{1}{12800Step} \right), \quad (90)$$

where x now are the first step and L is the inductance in the coil with no covered. Putting this back into (91) and set C equal to 1.5 μH gives,

$$F = \frac{1}{2\pi\sqrt{((115 \cdot 10^{-6})\text{H} - ((55 \cdot 10^{-6})\text{H}(\frac{1}{12800\text{Step}}))) (1.5 \cdot 10^{-9})\text{F}}} = 380435 \text{ Hz}. \quad (91)$$

If step are changed from 1 to 2 and every other value is the same, then this gives a frequency of 380442 Hz.

The frequency change between step 1 to step 2 is,

$$380442 \text{ Hz} - 380435 \text{ Hz} = 7 \text{ Hz}. \quad (92)$$

When calculating frequency span it is important to find how many steps that needs to be covered before the counter circuit can see change. However, the other coils are going to see a change. One coil is 100 % covered while the others are 59/90 degrees covered and 1/90. This means that the change in the last coil can accommodate for lack of change in the other coil. This happens because the rotor is 210 degrees. See figure 130, where the light green part of the rotor is copper. This gives a visual representation of how the rotor always covers two coils.

To keep the precision of the sensor, it is important to make sure that coil A changes in Hz when coil C is at the point with minor changes. This goes the other way around for all of the coils.

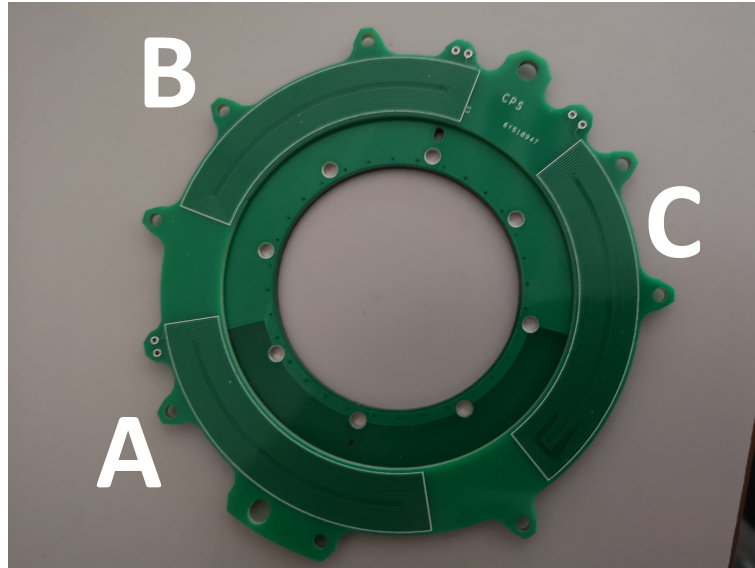


Figure 130: Stator on rotor

When the rotor cover 1 step of coil A, then coil C is covered 8534 steps out of 12800. Putting 8534 in (91) gives 464311 Hz and step 8535 gives 464324. This give an frequency change of 13 Hz. Therefor a calibration of 1 second per step is possible and a calibration of 0.65 second is possible if the margin of error is ignored.

Figure 131 shows the estimate on how the frequency is going to change with steps covered by the rotor.

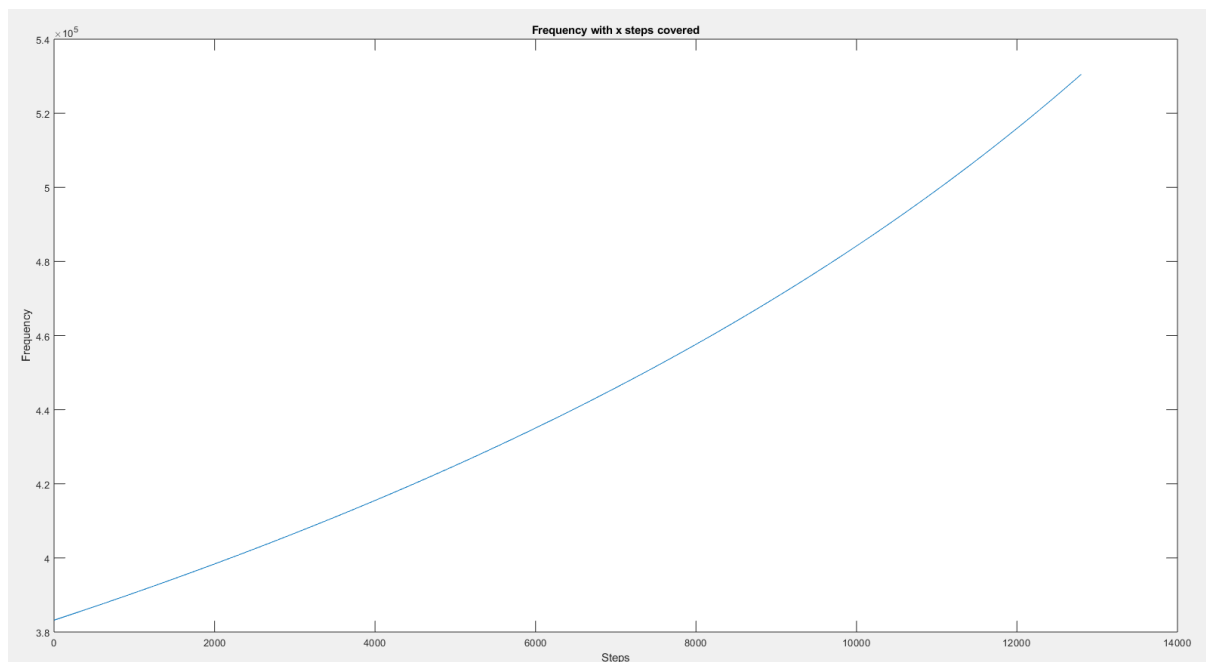


Figure 131: Frequency depending on rotor position in steps

20 Qt Framework

20.1 Document history

Table 63: Qt Framework document history

Version	Date	Author	Description
1.0.0	20.03.2019	MBC	Document created and started subsection 2.
1.2.0	21.03.2019	MBC	Finished subsection 2.
1.3.0	23.03.2019	MBC	Added subsection 3 to 6.
1.3.1	25.03.2019	JSS	Proofread.
1.3.2	26.03.2019	MBC & HS	Proofreading and corrections.
1.3.2	15.05.2019	MBC	Proofreading and corrections.
1.3.3	22.05.2019	CPS team	Proofreading and corrections.

20.2 Introduction

This section will explain what it is and how to use the Qt framework. It will explore some core concepts needed to understand the final source code for the CPS project.

20.3 Qt framework

Qt is a framework for creating applications that support a vast selection of platforms. Examples of popular ones are Windows, OS X, Linux, Android and iOS [60]. Qt as a framework means that it is not an independent programming language but instead adds functionality to already existing and supported programming languages. This means that it can be viewed as an external library that can be included and used. Qt is written in C++ and uses a preprocessor to extend C++ with different features. The way the preprocessor handles Qt extended C++ files makes it possible for Qt applications/classes/projects to be compiled by other C++ compilers like Microsoft Visual Studio(MSVC), Clang, MinGW etc [60]. This is relevant as it shows how well it can be integrated with "regular" C++ code.

20.4 Qt Integrated Development Environment

Qt also comes with its own Integrated Development Environment (IDE) called Qt creator. This IDE runs on the three major platforms, Windows, OS X and Linux. Qt Creator is a rich IDE with syntax highlighting, debugger, intelligent code completion etc. It can also be added to MSVC via an add-in [60]. Navigating through the IDE is simple and effective. The IDE offers the user the option to choose different modes. By choosing an appropriate mode a user can browse sample code or tutorials or access the project menu, or design GUI using the designer mode. The compiler also offers traditional directory view and output windows. Other standard IDE elements are present like the directory view, menu bar and output window.

20.5 Qt Designer

The Qt framework offers to both write GUI elements directly in C++ or instead utilise a design tool called Qt Designer. Qt Designer is a drag and drop environment where one can create and customise the graphical user interface to the users liking.

Adding a widget through the creator mode does not add code to the main project C++ files, but instead adds XML nodes to a UI file which at compile time gets translated to C++ code that can be compiled. The XML nodes in the UI file is illustrated in 3.

```
<widget class="QWidget" name="centralWidget">
  <widget class="QPushButton" name="pushButton">
    <property name="geometry">
      <rect>
        <x>60</x>
        <y>40</y>
        <width>250</width>
        <height>250</height>
      </rect>
    </property>
    <property name="text">
      <string>PushButton</string>
    </property>
  </widget>
</widget>
```

Listing 3: Qt Designer XML example

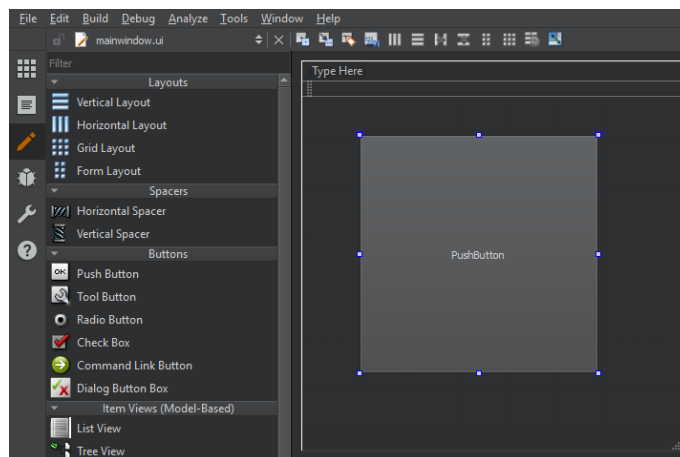


Figure 132: Qt Designer example

In figure 132, different application layout properties are represented in the menu under "Layouts". These and the "Spacers" properties are used to edit the position of different widgets. The "Buttons" section represent different buttons that can be dragged and dropped into the application window. This widgets properties can then be edited by accessing the properties window.

Styling widgets trough Qt is quick and effective as one can use Cascade Style Sheets bases. Here its possible to change name, dimensions, texts, and further edit the widgets properties.

20.6 Not using Qt Designer

With Qt using the Qt Designer tool is optional. It is possible to add widgets by hard-coding as shown in figure 4. In this example a button object is created and has geometry set.

```
button = new QPushButton("My Button", this);  
button->setGeometry(QRect(QPoint(60, 50), QSize(250, 250)));  
connect(button, SIGNAL (released()), this, SLOT (handleButton()));
```

Listing 4: Qt button example

The connect function is Qt's alternative to callbacks. A callbacks are when executable code is sent as parameters. In relation to GUI development, when a button is changed or a radio button is checked, something may be required to happen. In listing 4, when connect is called it connects the button with the slot/function. In this example this slot/function will resize the button and sets the text to the string specified. Thus each time this button is called, this function is called.

21 Log files and directory structure

21.1 Document history

Table 64: Log files and directory structure document history

Version	Date	Author	Description
1.0.0	15.03.2019	JSS	Document created.
1.1.0	18.03.2019	JSS	Added introduction, the directory structure, the solution, file name format, directory structure diagram.
1.1.1	26.03.2019	HS & MBC	Proofreading and corrections.
2.0.0	12.05.2019	JSS	Adding PDF generation and updating XML generation.
2.0.1	17.05.2019	MBC & AR	Proofreading and corrections.
2.0.2	22.05.2019	CPS team	Proofreading and corrections.

21.2 Introduction

This chapter will introduce the directory structure that is auto-generated when a new log file is made and what the log files contains.

21.3 Stakeholder requirement

Stakeholder requirement 1.20 in table 18 states: "The test station shall log the results of a test in a format which enables for extraction at a later time (all parameters and settings shall be included in the log file)".

21.4 The directory structure

As the CPS system will generate a log file after each test run, it would be helpful to have these files sorted in a way that allows for easier extraction when the user requires it. That is why it was decided to set up a directory structure where the user specifies the name of the project, the component ID that is to be tested and the test ID, from this the directory structure will be auto generated with new folders if needed. It will not allow earlier log files be overwritten if they share the same name. The exclusion from this is the calibration file that has a need to be updated after each calibration run. The name for the calibration file needs to be static for the program to find it.

21.4.1 Directory structure diagram

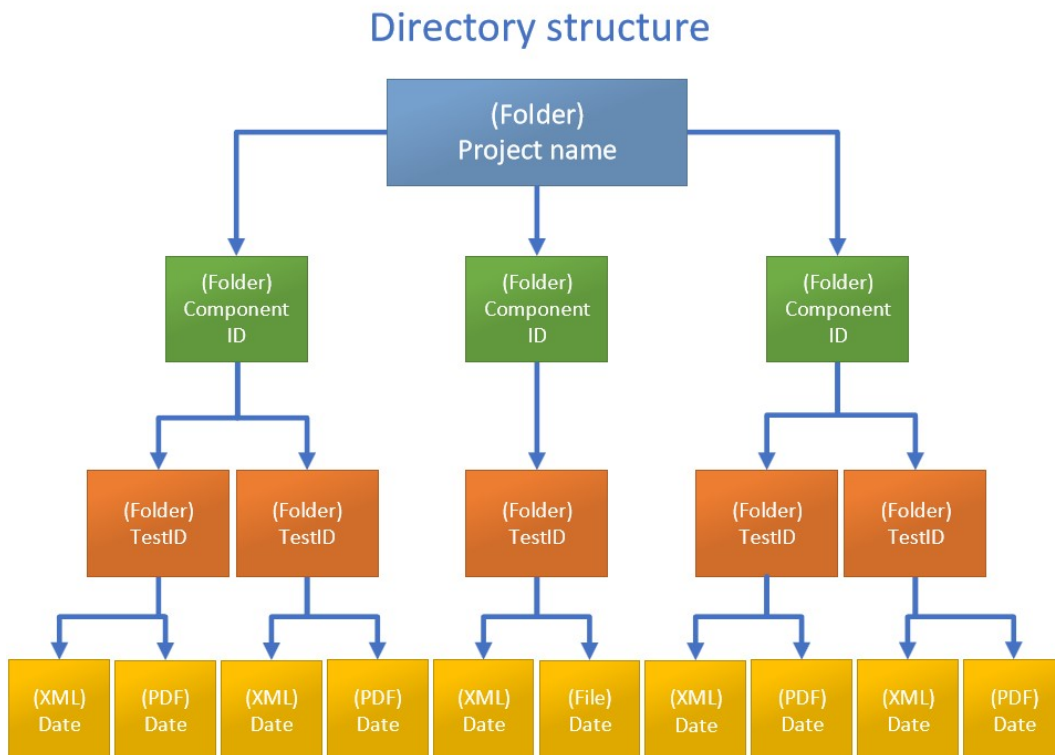


Figure 133: Diagram of the directory structure

One project can have multiple components tested.
Each component can be tested in multiple different tests.
Each test can be run multiple times generating multiple files. Both XML and PDF.

21.5 Log and report file

There are two types of files that can be generated.

21.5.1 XML log file

Extensible markup language (XML) is designed to store and transport information in self descriptive nodes. It has been useful to store and extract information from custom nodes that describes the information they contain. Since the tests for accuracy requires it to be compared to a baseline in ideal conditions, which is the conditions the calibration is ran in. And the tests for linearity requires a list of the frequency values in a rotation, which is essentially what the calibration does.

The parameters and information that is to be saved in the XML log file is described here in table 65. The difference in colour signifies a change in the parent node. an example file can be seen in the section 21.5.2

Table 65: Explanation of each node in an XML log file

Node	Description	Parent node
TestID	The ID of the test that is to be conducted. This is the main node	None
Date	The date the test is conducted	TestID
Test personnel	Te person conducting the test	TestID
Project name	The name of the project	TestID
Component ID	The ID of the component to be tested	TestID
Test ID	The ID of the test that is to be conducted, this is a node with the same name and information as the main node	TestID
Notes	Special notes considering the test	TestID
Parameters	A list of parameters	TestID
Stepmode	The stepmode of the driver	Parameters
Current	The current setting on the driver	Parameters
Distance	The distance it is to change	Parameters
Speed	The speed of the driver	Parameters
Gear ratio	If gears are present to increase the step resolution, this the the gear ratio	Parameters
Results	A list of results, if multiple tests are ran there will be one result node per run	TestID
Result	A list of saved values from the test	Results
CoilA	The measured values of CoilA	Result
CoilA Expected	The expected values of CoilA gathered from the calibration file	Result
CoilB	The measured values of CoilB	Result
CoilB Expected	The expected values of CoilB gathered from the calibration file	Result
CoilC	The measured values of CoilC	Result
CoilC Expected	The expected values of CoilC gathered from the calibration file	Result
Degree	The location of the sensor in degrees in relation to the calibration file	Result
Step locations	The location of the sensor in steps in relation to the calibration file	Result

PugiXML [25] is used to build up, edit and save the XML file. PugiXML is an XML processing library for C++ which can be compiled together with the project without the need for library files.

21.5.2 Example XML file output

```

1  <?xml version="1.0"?>
2  <testid name="T-C0-1.0.0">
3    <date>14.05.2019 09:54</date>
4    <personnel>Jarand Strommen</personnel>
5    <projectname>CPS2019</projectname>
6    <componentid>DM420A</componentid>
7    <testid>T-C0-1.0.0</testid>
8    <notes></notes>
9    <parameters>
10     <stepmode>1/128</stepmode>
11     <current>0.31 RMS Current</current>
12     <distance>51200 Step(s)</distance>
13     <speed>5000 Step(s)/s</speed>
14     <gearratio>1/1</gearratio>
15   </parameters>
16   <results>
17     <Result ID="0">
18       <coilA>14029</coilA>
19       <coilAexpected>99999</coilAexpected>
20       <coilB>14029</coilB>
21       <coilBexpected>61230</coilBexpected>
22       <coilC>14029</coilC>
23       <coilCexpected>62323</coilCexpected>
24       <degree>210.937500</degree>
25       <Step location>30000</Step location>
26     </Result>
27   </results>
28 </testid>

```

Listing 5: Directory structure - Example XML

21.5.3 PDF report

The other file that can be generated is a complete PDF report with information from the current run compared with the calibration file. This is an extra function not needed by the stakeholder requirement but allows users to save time by auto generating this report ready for export without the user having to set it up themselves. These reports can also be generated for the calibration run if needed.

Libraries used for PDF report:

To generate the PDF report the external library LibHaru [16] was used. This library, unlike pugiXML, was not easy to integrate into our project. The library had two libraries that it depended on. All libraries can be seen in the following table:

Table 66: Overview of libraries used to set up PDF report generation

Library	Description	Dependencies
LibHaru [16]	Used to generate PDF documents, not capable of reading or editing existing PDF document	Libz & LibPNG
Libz [33]	Used for data compression and decompression	None
LibPNG [46]	Used for PNG image manipulation	Libz



Challenges along the way:

The developers of LibHaru has set up the library so it can function with as many compilers as possible, and as such there is no premade library files that can be included in a project. These need to be generated from their GitHub project [15]. This can be done multiple ways depending on the compiler use. Cmake compiler was used to generate a MSVC 2017 compatible project. This had to be done for the two other libraries beforehand as it is required to link the lib files from Libz and LibPNG to successfully make the project. After the MSVC project was successfully generated it had to be build and compiled, this made sure that the .lib files could be implemented into our project. The lack of prior knowledge around these kind of libraries made this very difficult to complete even with the installation instructions as there were a certain level of familiarity with compilers that seemed to be needed.

After the .lib files were successfully made, linking them to the project was troublesome as there were other files that needed to be added as well since the .lib files were dependant on them. These included the path to the header files for all three libraries, the path to the .lib file for each of the libraries and the name of each .lib file needed to be specified for the compiler to find them. After all this LibHaru was up and running.

Making these libraries for the native compiler in Qt creato was also triedr, but this proved too troublesome and instead the compiled used for Qt was changed to MSVC 2017.

Formatting the PDF report:

Formatting a PDF with the use of Libharu was challenging as every element that was to be placed needed to have a starting point. The default anchor point, that is where the measurement started from, was in the bottom left of the new page. All measurement was in pixels so the grid to draw on was small.

To make the page setup easier to alter, constant variables for calculating distances was used. This was to simplify the calculations needed to get a default page format. This did take a lot of experimenting to complete, but was worth it in the end.

It was planned to have the chance to do multiple tests and have them documented in the same report, but because of troubles this was not accomplished. Even so it was taken into consideration when the PDF generation was set up so the box containing the test results is dynamic and will expand to new pages if the amount of tests run exceeds the pages. This has been tested up to 1024 pages of results. This was made use of when generating a PDF report on a calibration run. The text box for the notes is also dynamic just in case there is a lot to take into consideration for a test, but the notes text box is limited to only one additional page.

21.5.4 Log and report file name format

The log file and report file generated will have the current date as its name and will have the format:

YYYY-MM-DD-HH-MM-SS.xml

This means that it will be dated by year, month, day, hour, minute and seconds.

21.6 Last step

One problem that occurred was that if the program were to shut down after a calibration, the reference used to see the location of the rotor on the sensor would be reset and make the calibration invalid.

As a step to keep the calibration file valid the CPS set up an external file that would be generated and updated every time the stepper motor moved the rotor. This would keep track of the last position of the rotor and keep the recent calibration valid. The filetype used was an XML file as team already had a way to access, change and save these kinds of files through the PugiXML library mentioned in section 21.5.

There was a second approach considered to solve this problem which was to save a variable that would keep track of the location in the flash memory on the Arduino. That is the memory that is retained even when the Arduino is shut down. Since this value was needed on the control side of the system, that is on the PC, this would need to be sent over by the Arduino and caught by the computer which was proven to be difficult. Accessing the flash memory would not have been easy to accomplish either.

21.7 Calibration

Calibration is crucial in our project as it is desired to test the stakeholder requirement 1.3 in table 18. For this the software was in need of a calibration file which had all frequency values logged for each step in a full rotation. The values would be from when the sensor was in ideal conditions to set a base line for testing. A comparison between these calibrated values and new values read from the CPS under other conditions would give an error margin which will be used to see if the accuracy and linearity of the sensor are high enough to pass the requirement.

21.8 Doxygen documentation and code

The doxygen documentation for the directory structure can be found in section 32. The full Visual studio solution and clean text version can be found in the attachments under the folder:Final Test Station Software.

22 CPS graphical user interface

22.1 Document history

Table 67: CPS graphical user interface document history

Version	Date	Author	Description
1.0.0	23.03.2019	MBC	Document created, and added subsection 1 to 3.
1.0.1	25.03.2019	JSS	Proofread.
1.0.2	26.03.2019	HS & MBC	Proofreading and corrections.
2.0.0	14.05.2019	MBC	Started on final GUI section.
2.1.0	15.05.2019	MBC	Finished description of the GUI.
2.1.1	17.05.2019	MBC & AR	Proofreading and corrections.
2.1.2	20.05.2019	MBC	Proofreading and corrections.
2.1.3	22.05.2019	CPS team	Proofreading and corrections.

22.2 Introduction

Communication with primary stakeholders during the development of the project is important, as it gives the stakeholder the possibility to understand and influence the end product. The CPS team has designed GUI concepts. This chapter will explain the different GUI mockups that have been created and explain which one the CPS team and KDA chose to iterate further and base the CPS test station GUI on.

22.3 GUI Concepts

These GUI concepts were drawn in Microsoft Visio as it allowed for quick and easy editing. They do not represent the final product, but are intended to act as iterations/inspiration for the final product. Note that the display of data is prone to change and does not need to be identical. This means some of the graphs may end up having other means of representing data such as digital numbers or other widgets.

22.3.1 GUI concept 1

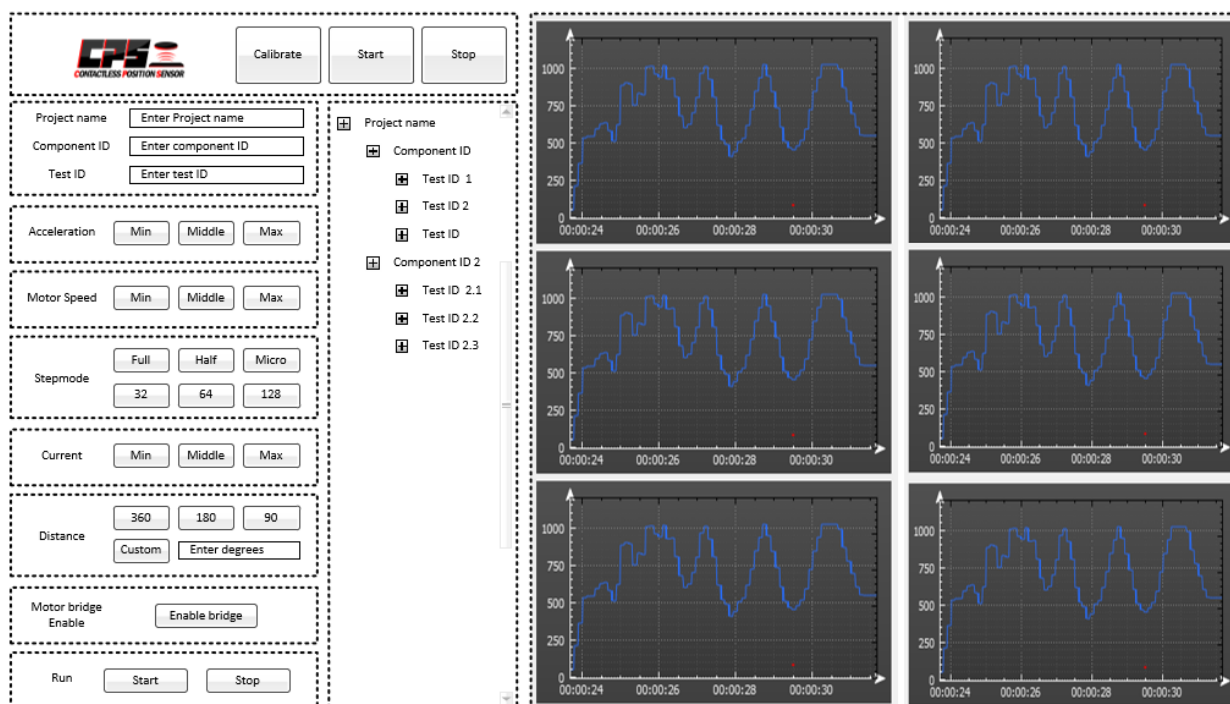


Figure 134: GUI concept 1

GUI concept 1 illustrated in figure 134, is based on push buttons and text fields to enter custom data. The idea is that the user selects parameters by pressing the appropriate push buttons, and once a parameter is pushed the button gets highlighted. STQR 1.19 from table 18 states that the test station shall be able to show change in degree and speed and that is accomplished by having said data projected on the graphs. STQR 1.20 to 1.23 from table 18 are represented as buttons and the file directory is shown in the middle of the GUI.

22.3.2 GUI concept 2

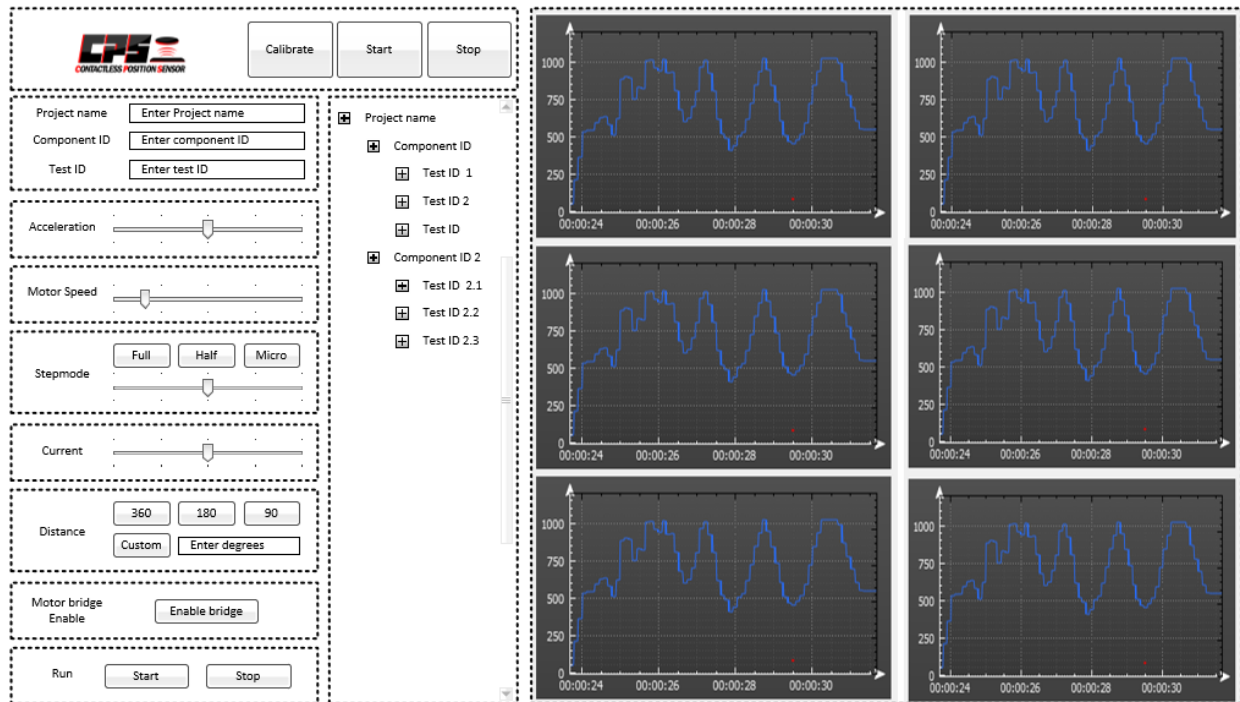


Figure 135: GUI Concept 2

GUI concept 2 illustrated in figure 135 is an alternative version of GUI concept 1, except instead of having buttons to set parameters slider bars are used. This would give the user the possibility to have more options inside each parameter. An example of this would be the difference in motor speed options from figure 134 and figure 135. In figure 134 the user has 3 choices of motor speed, "min", "middle", and "max" while in figure 135 the slider has 5 options. Having better options inside each parameter would greatly increase the value of the test station as it can offer a more diverse set of environments.

GUI concept 3 illustrated in figure 136 is the same as GUI concept 2, except that the file directory is moved to the left of the GUI. This concept is inspired by the Windows Explorer where the directory view is moved to the left. Having the directory as this is frequently used in software that require a directory view and can be what users might find comfortable.

22.3.3 GUI concept 3

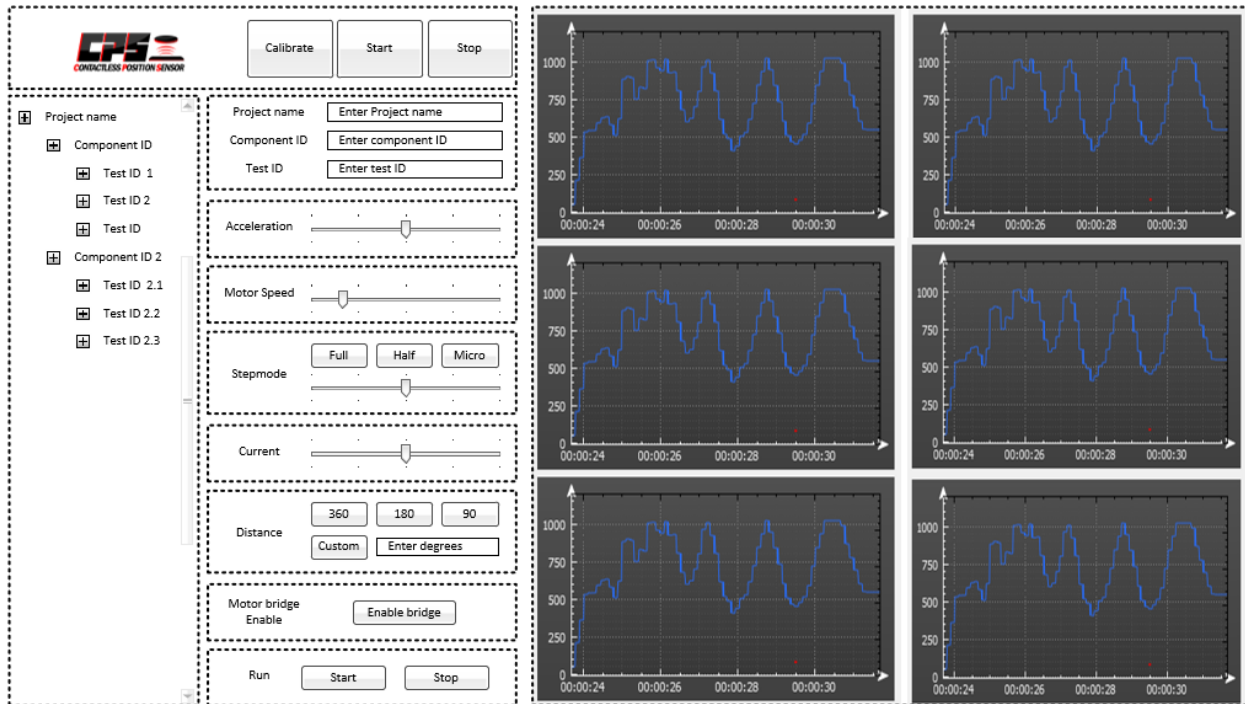


Figure 136: GUI Concept 3

GUI concept 4 illustrated in figure 137, is a variation off the other concepts, where the file directory is cut out from the GUI. It would still be present in form of a drop-down menu bar located at the top of the GUI. This solution could potentially free up window space for other functions or graphical representations illustrated by the graphs surrounding the middle.

22.3.4 GUI concept 4

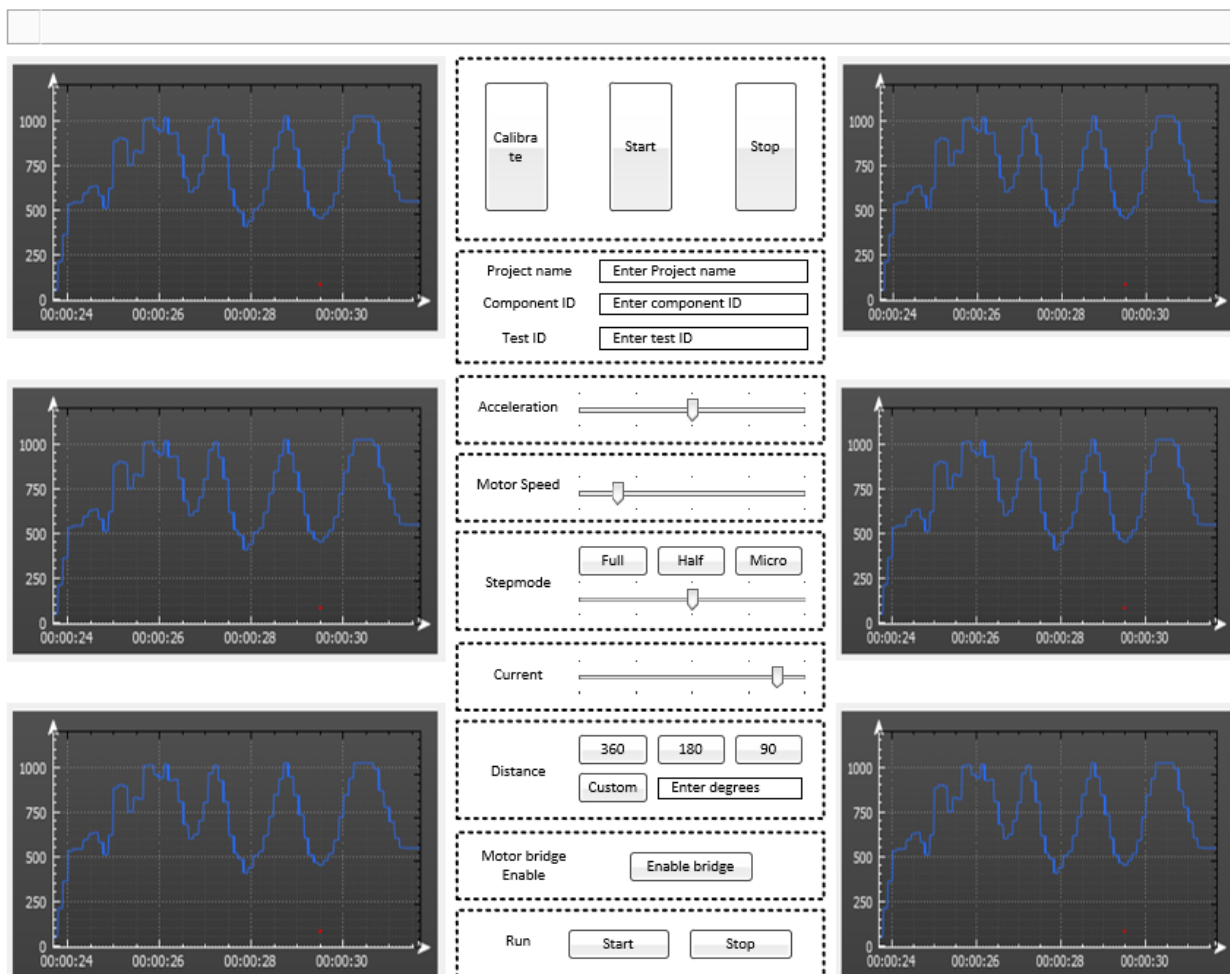


Figure 137: GUI Concept 4

22.3.5 Choice of GUI concept

After conversation with KDA about GUI concepts the group and KDA agreed to further develop GUI concept 3. KDA found the sliders the most fitting although they wanted to further iterate and add an input field to let the users add custom values to the parameters. The idea is to further improve this concept and optimise it as to give the best user experience as possible.

22.4 Final graphical user interface

This section will explain how the final graphical user interface for the test station works and how it was developed.

As mentioned in section 22 the CPS team decided to further develop concept 3. The final

graphical user interface is illustrated in figure 144. During the development of the GUI, the team focused on meeting the stakeholder requirements 1.19 to 1.23 from table 18.

22.4.1 File directory

The GUI can be split into 3 categories which are directly connected to the Use Cases and the stakeholder requirements. The leftmost category is the section where saved logs and the calibration file appears. This is an extensive directory view which allows the user to navigate through different projects, components and tests. The file directory is further explained in section 21. Following this the user can access the files from the rightmost directory window. Once the program starts running, two objects of the Qt class `QFileSystemModel` is instantiated. This class provides data models for the local filesystem [61].

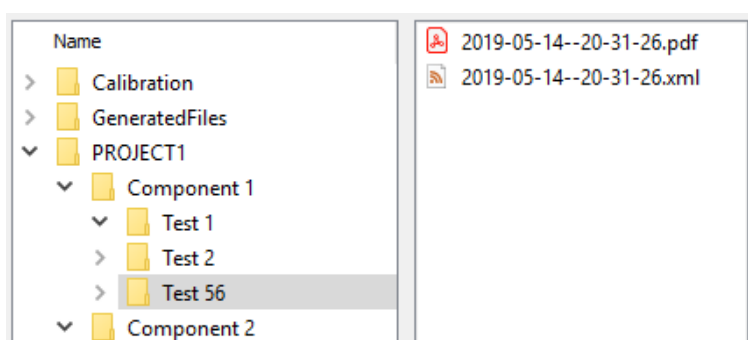


Figure 138: Directory view from final GUI

The objects gets different root paths, as to display the correct path. The leftmost model gets set to an upper level root and the rightmost model gets the same root initially. After styling the objects they are then connected to the corresponding user interface widgets. In this case they are coupled to a listView widget, and this enables for the usage of predefined slots. As mentioned in section 20 slots are called once its connected signals are emitted. In relation the directory view once a user clicks on a folder or a file, it's root path is extracted and set as a new root for that file. This way the users can both open and close files. In the rightmost section the files, inside the selected folder, are displayed. As with the tree view once the user selects a file, its root path gets extracted and with that path as a parameter a function from `QDesktopServices` class is called. This function accesses the file at the given root path. This functionality is illustrated from the Use Case diagram in figure 114.

22.4.2 Control section

The middle section of the GUI is where the user can set parameters and control the test station. This section can again be categorised in 3 sections; control, parameters, and project information. The control category is where the user initiates an action or a sequence of actions. The project-information category is where the user inputs useful information about the project, and lastly the parameters category is where the parameters for testing are set. As illustrated in figure 139 there are in total eight buttons available for the user. These buttons initiates different segments of code that perform different tasks.

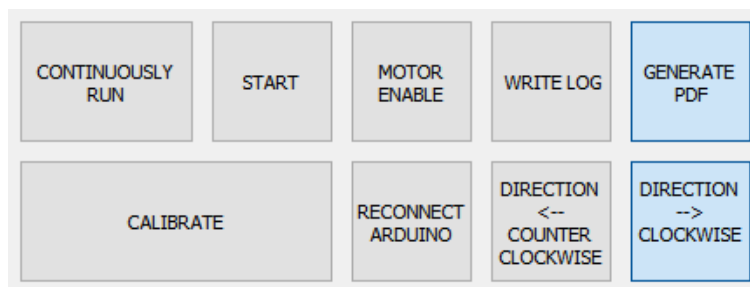


Figure 139: Control section part 1

"GENERATE PDF" is a checkable button and when its highlighted, as shown in figure 139, it enables the generating of a PDF version of the log. The "WRITE LOG" button is not checkable and when clicked, takes all of the current data, parameters and associated data from the calibration file and generates XML document. As mentioned earlier, if the "GENERATE PDF" button is highlighted, a PDF format of the report will also be generated. The buttons labelled "DIRECTION COUNTER-CLOCKWISE" and "DIRECTION CLOCKWISE" are checkable buttons and when one is selected, the other one is deselected. These button set the direction of the rotation by the stepper motor. The last four buttons are labelled "START", "CALIBRATE", "CONTINUOUSLY RUN" and "RECONNECT ARDUINO". These buttons all have a common denominator that they all initiate segments of code that communicates with an Arduino. Figure 127 illustrates how the software and hardware is interfaced and depicts a simplified communication run. On start up of the program an QSerialPort object is created and given the appropriate configuration to setup a serial connection with the arduino. If, however the arduino gets disconnected while the program is connected, the reconnect button will close the previous connection and open a new one. This way the software does not have to be exited and opened again. The start button takes all of the set parameters and builds a QByteArray consisting of different information about the speed of the motor and how many steps to move. A QByteArray is the Qt version byte array. This bytearray is sent to the arduino to perform the given motion. The last button is the calibrate button. Upon clicking this button a pre-constructed instruction is sent to the arduino, and the calibration process is started. Due to memory constraints on the Arduino Mega 2560 only 100 samples of the three coils can be transmitted by the arduino at one time. Therefore the arduino transmits the data set of 100 values then starts recording new values. This is looped 512 times and is due to the stepper motors max resolution of 51200 steps. This way the calibration will provide a detailed description of each possible degree when testing.

PROJECT NAME	Projectname not selected
NAME OF USER	User not selected
COMPONENT ID	Component ID not selected
TEST ID	Test ID not selected
SPECIAL NOTES	Special notes not selected

Figure 140: Control section part 2

Figure 140 illustrates the information the user can enter. There are 5 text input fields where the user can specify the name of the project, name of the user, the component being tested, and special notes if there are any special circumstances that need to be considered. The text input fields have placeholder text that disappears once the user starts typing. Inputting information in these text fields assures that folders are created and the tests are not all saved in the same folder.

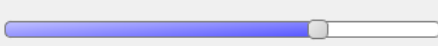

GEAR RATIO	Gear ratio not selected	
CURRENT	0.31 - Remember to change DIP switch ▾	
STEPMODE	1/128 - Remember to change DIP switch ▾	
SPEED	Degrees per secor ▾	66
	 66	
DISTANCE	Steps ▾	243892
	 243892	

Figure 141: Control section part 3

The last category is the parameter section, and consists of the tools the user needs to be able to set different parameters for testing. The first three parameters; gear ration, current, and step mode are all parameters the user needs to manually change on the different hardware devices. Changing them on the GUI only ensures that the values are changed on the data log. While gear ratio is a text input field where the user decides what to write, the current and speed parameter are both drop down menus. The speed and distance parameter can be set by either using the slider or entering the value directly in the spinbox. In addition to this, the user can decide to set these parameters in either degrees/s

or steps/s. This is done by using the combobox next to the "SPEED/DISTANCE" label. The value selected is displayed on the LCD widget.

22.4.3 Display CPS data section

The rightmost category of the GUI consists of the displaying of realtime data and expected data from the calibration. Here the realtime data from the counter circuit is displayed on lcd widgets as well as being plotted to a graph in realtime. The graph is made using a custom Qt library called QCustomPlot.

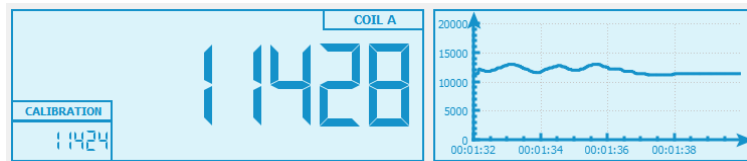


Figure 142: GUI realtime CPS data

Depicted in figure 142, the data from the counter is projected on both the LCD display and the graph. In the bottom left corner, it is possible to see the expected value (from the calibration) at the given degree or step position. This allows for quick comparison and not having to write a test-log and then compare the values.

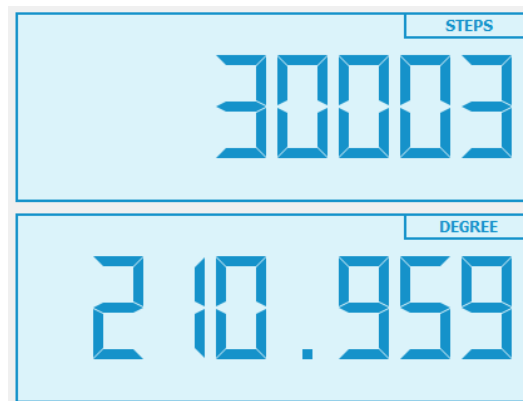


Figure 143: GUI current position

The last data that is displayed on the graphical user interface is the position data. Here the user is displayed what the current degree is in and what the equivalent in steps. Every time the position changes the new position is stored in a XML file that allows for the GUI to "remember the last position" if the application is closed.

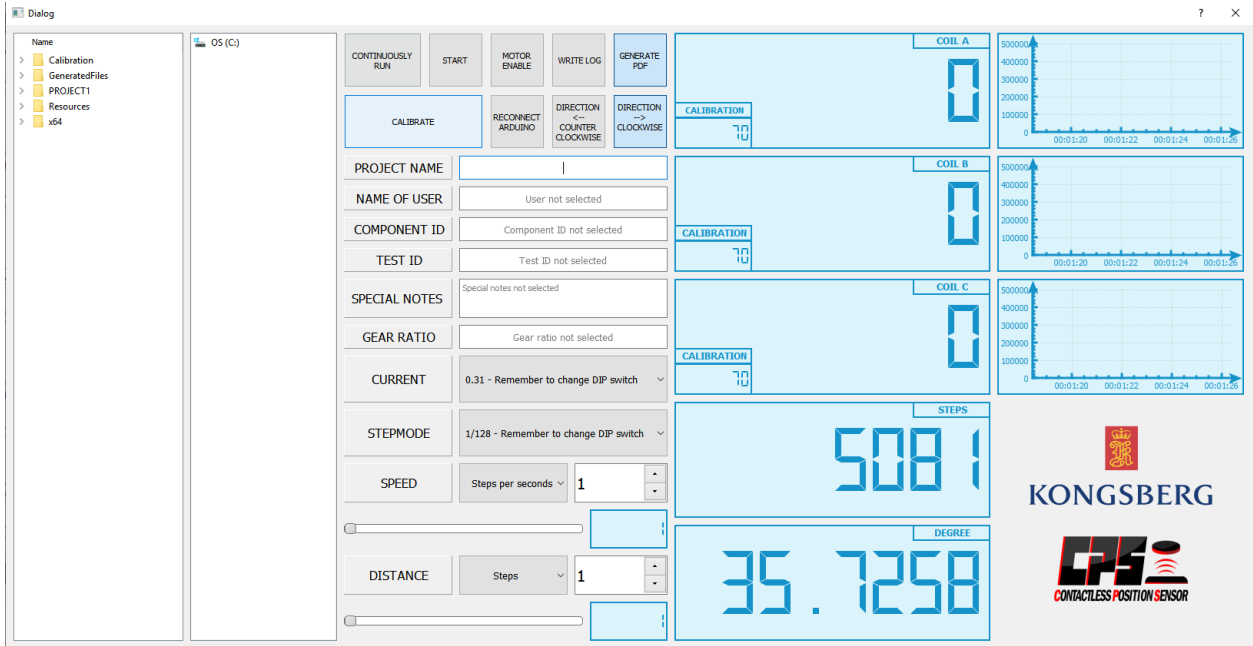


Figure 144: Final graphical user interface

23 Source code

23.1 Document history

Table 68: Source code document history

Version	Date	Author	Description
1.0.0	15.05.2019	MBC	Document created. Added hardware configuration.
1.2.0	16.05.2019	MBC	Added class diagram and information.
1.2.1	17.05.2019	MBC & AR	Proofreading and corrections.
1.2.2	20.05.2019	MBC	Proofreading and corrections.
1.2.3	22.05.2019	MBC	Proofreading and corrections.

23.2 Introduction

This section will show the different classes in the project. An in-depth explanation of every member used in this project, can be read in the doxygen generated report, which is attached as appendix 32.

23.3 Hardware configuration for developing and testing

The CPS test station software has been developed and tested on the following hardware configuration.

Table 69: Hardware configuration

Hardware configuration	
Manufacturer	Asus
Model	Zenbook 14
CPU	Intel(R) Core(TM) i7-8565U CPU 1.99 GHz
RAM	16.0 GB (15.8 Usable)
GPU 1	Nvidia GeForce MX150
GPU 2	Intel(R) UHD Graphics 620

23.4 Class diagram

The class diagram illustrated in figure 145 illustrates the different classes in the final source code. This diagram depicts the classes in the test station software as well as the relationship between them. The external library classes are not documented in the final doxygen-report, but it is possible to read more about them on the official documentation for PugiXML [25], LibHaru [16] and QCustomPlot [17].

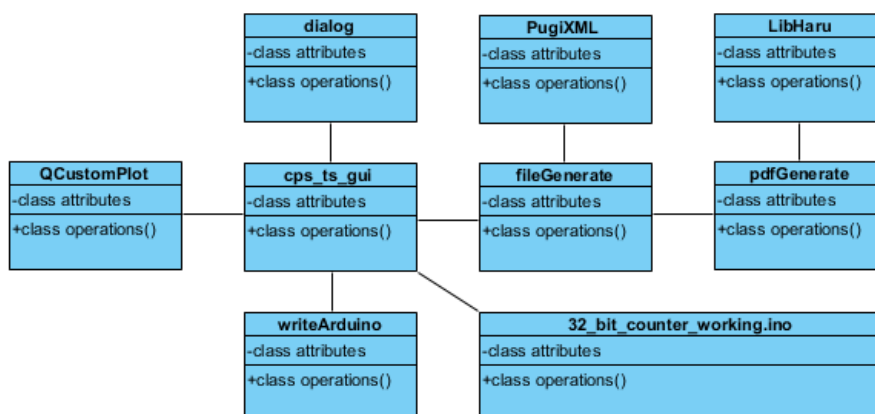


Figure 145: Class diagram final software

23.5 Include graphs

Illustrated in figure 146, the include graph generated by MSVC2017. The files created and/or added manually by the CPS team are grouped in the large box. External libraries such as Libharu and Qt are present but placed inside the "others" box.

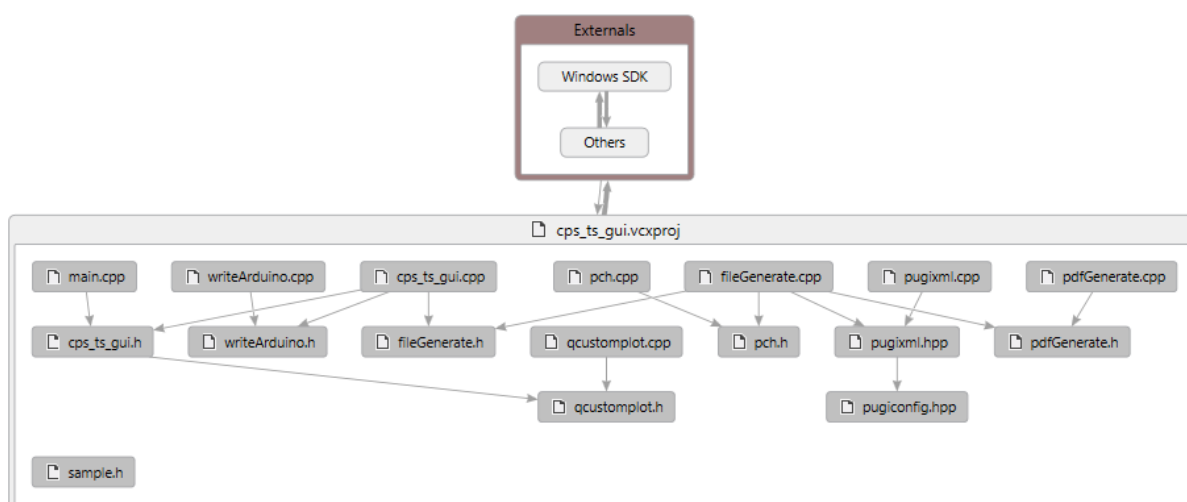


Figure 146: Include graph final software part 1

23.6 Arduino source code flowchart

The flowcharts illustrated in figures 147, 148, 149 and 150 shows how the arduino source code works. This code is responsible for reading and sending CPS data to the GUI, as well as controlling the rotor on the test station. The first flowchart (figure 147) showcases the loop that is run continuously on the microcontroller. First the microcontroller checks for incoming messages over serial connection. If there is incoming data, the data is stored in a variable and then a sub-process is initiated with the received data. If the serial connection is clear and a calibration variable is set to false, the microcontroller then clears the 32bit counters 18. After clearing a delay is initiated and after a certain amount of time the content on the counters is saved to its internal storage register. Then it is extracted from this register and sent over serial connection to the CPS test station GUI. Lastly the serial message is flushed, which means that the microcontroller waits for the outgoing message to be sent before doing anything new. The loop starts from the top again.

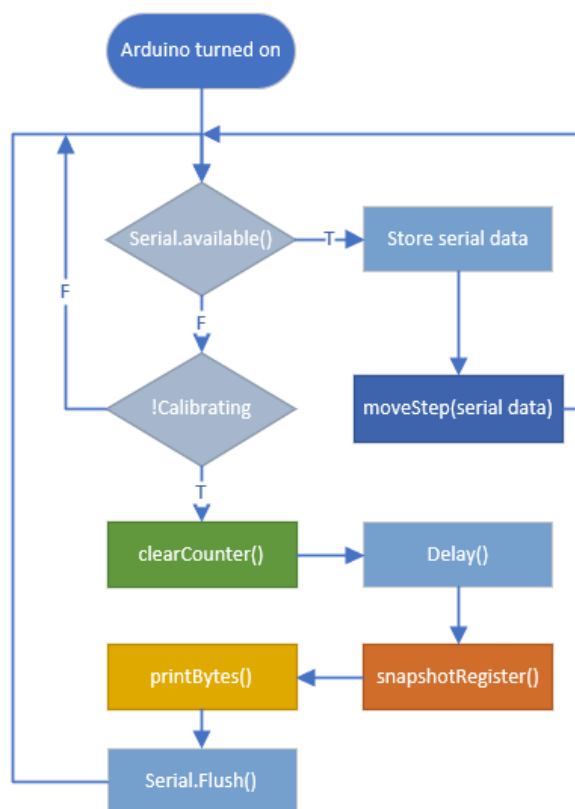


Figure 147: Arduino flowchart general

The microcontroller clears the counter by driving the appropriate control pins high and low as stated in the SN74LV8154 [54] datasheet. This sub process is illustrated in figure 148.

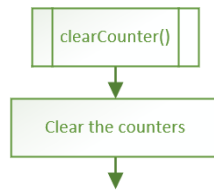


Figure 148: Arduino flowchart clear counter

Figure 149 illustrates the sub process of storing the value of the counters at a given point in time. This is done by triggering the RCLK pin on the counters. Doing this stores the value of the counters on the internal storage register on the IC and enables it for extraction. The values are then extracted and stored by cycling through the four bytes of the 32 bit counter.

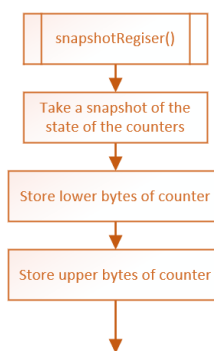


Figure 149: Arduino flowchart snapshot register

The next sub process is illustrated in figure 150. Here the stored bytes from the counters are put together in a byte array in order to get their correct complete representation of the number counted. The microcontroller then puts the values of the three counters together in a message string and sends it over serial communication for the CPS test station GUI.

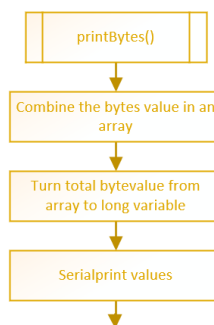


Figure 150: Arduino flowchart print bytes

The last sub process is the control. The moveStep() functions takes in the byte array that has been read through the serial communication decipher it. An explanation of the byte array values can be found in the section 15.7 with an example string that can be found in section 15.7.2.

The number 0 is added to the end of the byte array which will be used to exit the function when done. What the function does, is looking through the bytearray for the first & sign and move everything before that sign into another variable. For explanation reasons, the bytearray will be called "command". This will be the type of parameter and the value that belongs to it. The second thing the function does is looking through this new variable, "2command", for the sign and moves everything before that into another new variable called "paraId". So "paraId" will now be the ID of the parameter that is to be changed, and remaining in the "command" variable will be the value of that parameter. Renaming the "command" variable to "paraVal" too keep it descriptive.

Now that the parameter ID is saved in "paraId" and the value for that parameter is saved in "paraVal" we can use it to control the stepper motor driver. After the last split there is a switch case which looks at the "paraId" variable to see which parameter is to be changed and depending on the parameter the "paraVal" variable will be used to change settings for the stepper motor driver.

This was just for one of the parameters in the first bytearray so this will loop and look for the next & and repeat the process. When the program has gone through the entire bytearray it will encounter the number 0 which signals the end of the command line and is used to exit the function.

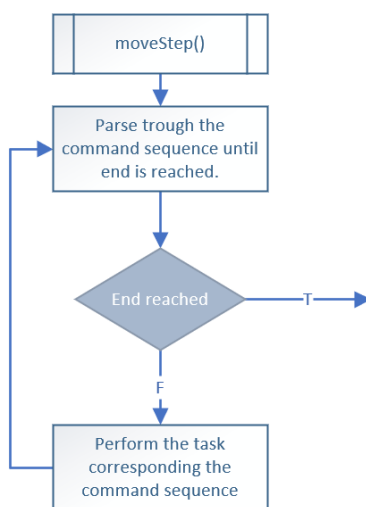


Figure 151: Arduino flowchart move steps

23.7 Signals and slots in Qt

When the different buttons and widgets are interacted with by the user, a signal is emitted and a slot is executed. Many of the widgets utilised in the software, already has built in connection between these signals and slots. An example of this is when the user clicks a button. A signal is then emitted and the slot (defined by the software) is run. Two examples of this practice is explained bellow, but a further explanation can be found in the doxygen report for the final software 32.

When data is sent from the arduino to the software, the QSerialPort detects this and emits a signal. This signal is called readyRead() [62], and is emitted once new data is available on the serial port. This signal is connected to a slot which results in that particular slot executing every time the signal is emitted. This is depicted in figure 152.



Figure 152: Qt signal and slots part 1

The process of plotting CPS data, runs in a similar matter. A QTimer object emits a signal with a given interval which triggers the execution of a slot. In this case the slot is realtimeDataSlot() and is responsible to plot the new content on the graph while removing the old content. This is depicted in figure 153.

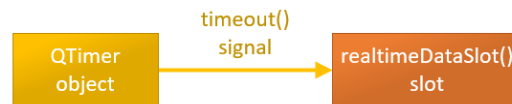


Figure 153: Qt signal and slots part 2

24 Conducted tests

24.1 Introduction

This section will contain the tests performed by the CPS team.

24.2 Test, KDA sensor T-C0-1.0.0

Table 70: Test, KDA sensor T-C0-1.0.0

Test ID	T-C0-1.0.0
Requirement ID	1.10.
Name	Hans Fredrik Jamtveit / Anders Rønning
Date and location	Elektrolabb 1, USN, Kongsberg
Goal	Find inductance of coils in prototype form KDA.
Hypothesis	Inductance of the coil will be found
Pass criteria	Inductance is found
Equipment	<ul style="list-style-type: none"> • Prototype form KDA. The prototype consisted of a 3 cm wide circular circuit board with three non circular coils. Wires were attached to all of the three coils. In addition there was a half circular shaped copper plate with the possibility to mount a thin metal rod in the middle. Prototype shown in(Figure 154) • 2 multimeter • 1 signal generator • 1 resistor 47Ω • Wires
Safety precautions	Turn off signal generator when handling components
Execution	<p>1. Calculating/measuring inductance Connecting a coil in series with a resistor creates a Low-Pass filter [23]. The cutoff frequency ω_c [rad/s] of the RL filter is</p> $\omega_c = \frac{R}{L}. \quad (93)$

Table 71: Test, KDA sensor T-C0-1.0.0

Execution	<p>Where R is the resistance [Ω] of the resistor and L is the inductance [Henry] of the inductor. The cutoff frequency is stated in Hz and has to be in rad/s</p> $\omega_c = 2\pi f, \tag{94}$ <p>The inductance of an inductor can be calculated by measuring the voltage drops across the resistor and the inductor while increasing the frequency of the input. When the voltage drop across the inductor and the resistor is equal the filter is at the cutoff frequency. The cutoff frequency is noted. At the cutoff frequency the reactance of the inductor</p> $X_L = 2\pi fL, \tag{95}$ <p>is equal to the resistance of the resistor given by</p> $R = \omega_c L. \tag{96}$ <p>At ω_c</p> $X_L = R. \tag{97}$ <p>Substituting X_L with R</p> $R = 2\pi fL, \tag{98}$ <p>Where R is the resistance of the resistor and f is the input frequency where the voltage drop across the inductor and the resistor is equal. Rearranging the formula with regards to L gives the following equation</p> $L = \frac{R}{2\pi f}. \tag{99}$ <p>2. Testing prototype</p> <p>While the sensor was attached to the signal generator and the resistor the CPS team tested how the copper plate affected the output of the sensor. The findings were that hovering the copper plate at different distances over the coil changed the amplitude and phase of the output. The frequency of the output did not change. Calculations and measurements of the induction showed a decrease in inductance while the copper plate was in proximity of the inductor(1,08mm measured with caliper).</p>
-----------	--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Table 72: Test, KDA sensor T-C0-1.0.0

<p>Result</p>	<p>1. Calculating/measuring inductance With a resistor at 47Ω and a cutoff frequency of 388,60 KHz this gives an inductance of:</p> $L = \frac{47 \Omega}{2\pi \times 388,66 \text{ KHz}} = 19,249 \mu\text{H} , \quad (100)$ <p>measuring the inductance of the inductor with the electronic explorer board(EE Board) and the WaveForms software inductance meter results in the same inductance of 19 μH.</p> <p>2. Testing prototype By using (8), the inductance was calculate to be</p> $L = \frac{47 \Omega}{2\pi 438,96 \text{ KHz}} = 17,04 \mu\text{H} , \quad (101)$ <p>this shows a decrease of 2,2 μH.</p>
<p>Observations</p>	<p>No significant external factors</p>
<p>Analysis</p>	<p>Finding the inductance makes design of the oscillator which is to be connected to the sensor easier. No change of frequency was detected in the test. This was as expected. The change in inductance proves that the concept is possible.</p>
<p>Conclusion</p>	<p>Results from the test of inductance shows that the presence of the rotor over the coils of the sensor decreases the inductance in the coils. There is no change in frequency while the rotor is over the coil of the stator when using a signal generator as input on the coil. There is a change in phase and amplitude of the output. Using the inductor of the sensor as a part of an LC oscillator and then exposing the inductor to the copper of the rotor will cause a frequency change of the output. This is because the frequency of oscillation in a LC oscillator [19] is determined by</p> $f = \frac{1}{2\pi\sqrt{LC}}, \quad (102)$ <p>and a decrease in the coil inductance from exposure to the copper plate confirms this.</p>

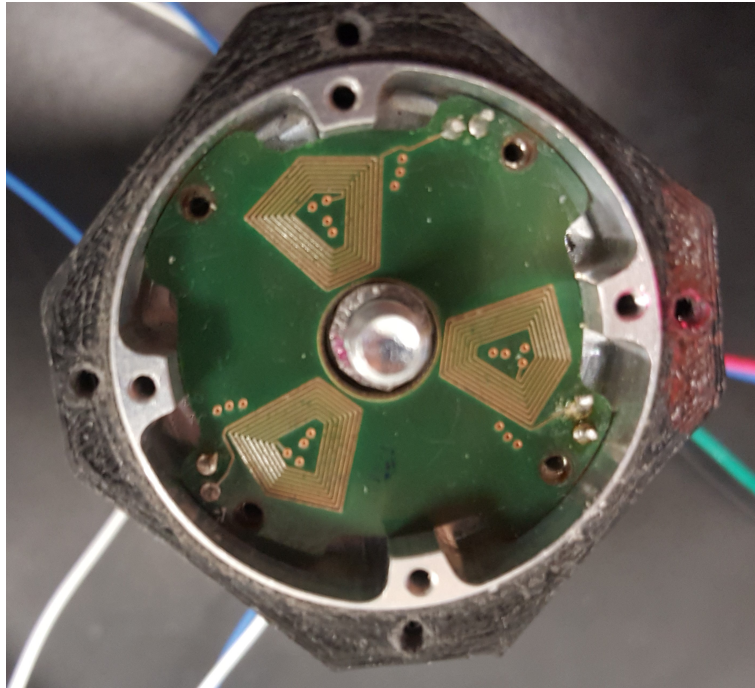


Figure 154: Prototype from KDA

24.3 Subtest, inductance final concept Sub-T-2.0.0

Table 73: Subtest, inductance final concept Sub-T-2.0.0

Test ID	Sub-T-2.0.0
Requirement ID	-
Name	Anders Rønning & Hans Fredrik Jamtevit.
Date and location	13.05.2019
Goal	To find the inductance of the coils in the stator.
Hypothesis	It should be around 50 μH to 400 μH .
Pass criteria	Connection from power to ground. Approximate the same inductance for all three coils on one stator.
Equipment	Waveform software, Electronics Explorer, oscilloscope, 1000 Ω resistor, 3 shims 0.60mm, stator and rotor.
Safety precautions	Made sure everything was properly connected to ground before connecting to power.

Table 74: Subtest, inductance final concept Sub-T-2.0.0

<p>Execution</p>	<p>Setting up the circuit like in figure 155. In the Waveforms software select "Impedance". Inside the new tab set the option to inductance. In setting the resistor needs to match the one in the circuit diagram. It is important to notice the value of the voltage amplitude.</p> <p>The physical setup was done in the 3D printed mount. The way to get the rotor in the same position every time was to push it as far into the wall of the mount every time. The use of shims on top of the stator to achieve the same height every time is also necessary. For each position of the rotor the position was measured against a marked paper with angles. See figure 156 and 157. In this way the measurement was more or less consistent each time.</p>
<p>Observations</p>	<p>This was done before the final test station was finished and therefore the result are approximations of what the real values are.</p> <p>There should also be a test to see how much the height of the rotor affects the inductance.</p>
<p>Analysis</p>	<p>-</p>
<p>Results</p>	<p>From the results in table 76, it shows that inductance is almost constant within the frequency range that is desired.</p>
<p>Conclusion</p>	<p>With this method it is not possible to test the accuracy that is necessary to determine an exact value for the inductance.</p> <p>There should also be a test to see how much the height of the rotor affects the inductance.</p> <p>The values were within the acceptable range and therefore the effect from rotor on stator is approved and a successes.</p> <p>This test should be repeated with the test station to verify that this measurement is approximately correct.</p>

Table 75: inductance 2V

Covered of coil	400 kHz	800 kHz
No rotor	96μH	94μH
0°	95μH	93μH
25°	80μH	83μH
50°	72μH	74μH
75°	63μH	63μH
100°	55μH	55μH

Table 76: inductance 5V

Covered of coil	400 kHz	800 kHz
No rotor	113μH	115μH
0°	112μH	114μH
25°	97μH	99μH
50°	81μH	82μH
75°	72μH	73μH
100°	63μH	63μH

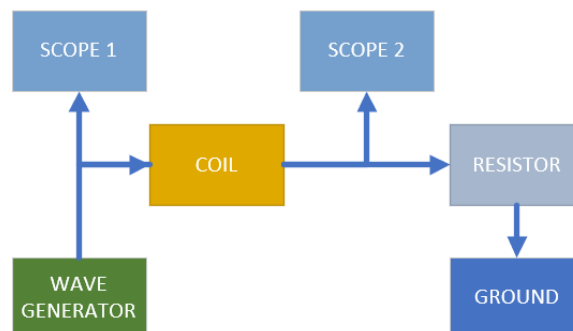


Figure 155: Circuit for setup

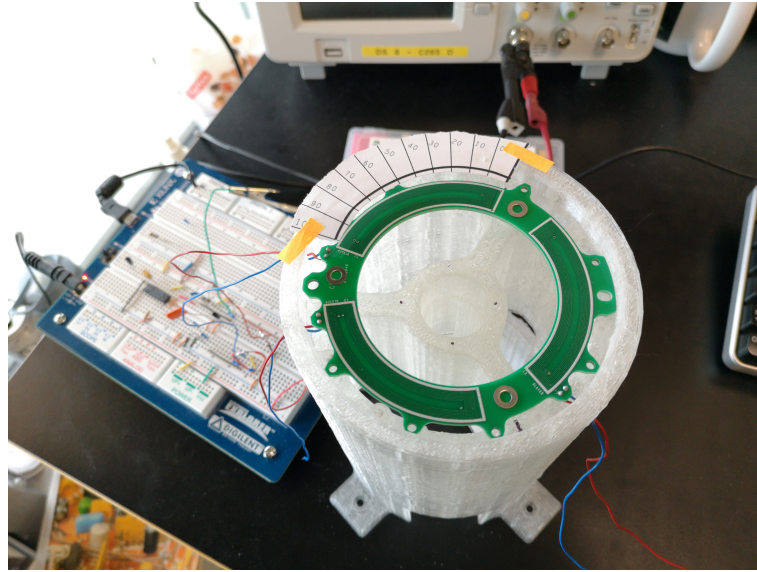


Figure 156: Stator in 3D printed test station

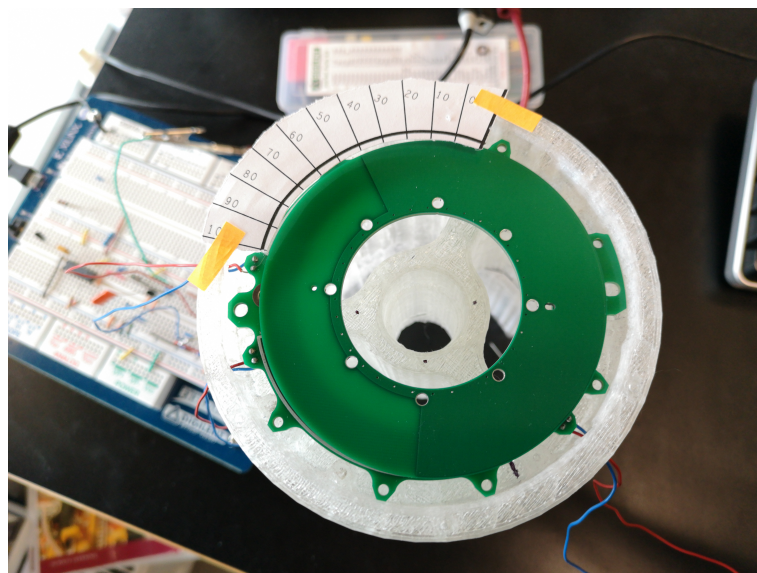


Figure 157: Rotor in 3D printed test station

24.4 Subtest, counter IC concept Sub-T-3.0.0

Table 77: Subtest, counter IC concept Sub-T-3.0.0

Test ID	Sub-T-3.0.0
Requirement ID	-
Name	Magnus Berntsen Caro, Anders Rønning.
Date and location	26.04.2019
Goal	Test to see if the counters can be utilised to count frequency and how accurate they can perform
Hypothesis	The SN74LV815 counter has sufficient technical specifications to be able to count to the desired level of accuracy, therefore it is suitable to be used in the CPS test station. Desired level of accuracy is down to 3Hz
Pass criteria	The SN74LV815 counter is able to count to the desired level of accuracy with an error margin of $\pm 5\text{Hz}$.
Equipment	<ul style="list-style-type: none"> • GW INSTEK AAFG-2005 Arbitrary Function Generator • SN74LV8154 Dual 16-Bit Binary Counters With 3-State Output Registers • Arduino Mega 2560 Rev 3 • Capacitor 100nF
Safety precautions	Made sure everything was grounded and had correct amount of voltage, and read datasheet.

Table 78: Subtest, counter IC concept Sub-T-3.0.0

Execution	<p>The SN74LV8154 counter was connected to the Arduino Mega and a signal from the signal generator was sent to the IC. The arduino program was set up to take 100 samples to be able to find and net value and the margin of error. The CPS team tested the following conditions;</p> <ol style="list-style-type: none">1. 100KHz input, 999ms counting time, 100 samples2. 800KHz input, 999ms counting time, 100 samples3. 100KHz input, 250ms counting time, 100 samples4. 800KHz input, 250ms counting time, 100 samples
-----------	-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Table 79: Subtest, counter IC concept Sub-T-3.0.0

Result	<p>Condition 1:</p> <ul style="list-style-type: none">• Net value: 100035.55• Max value: 100036• Min value: 100035• Difference between max and min value: 1 <p>Condition 2:</p> <ul style="list-style-type: none">• Net value: 800284.19• Max value: 800288• Min value: 800281• Difference between max and min value: 7 <p>Condition 3:</p> <ul style="list-style-type: none">• Net value: 25034.97• Max value: 25036• Min value: 25034• Difference between max and min value: 2 <p>Condition 4:</p> <ul style="list-style-type: none">• Net value: 200279.91• Max value: 200286• Min value: 200277• Difference between max and min value: 9
--------	------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Table 80: Subtest, counter IC concept Sub-T-3.0.0

Conclusion	<p>The CPS team was able to get precise and accurate counting from the SN74LV8154 integrated circuit while counting low frequencies such as 100KHz. On the other hand when turning the frequency up to 800KHz the accuracy suffered. From having a error margin of ± 1 on 100KHz, the CPS team experienced that the counter had an error margin of ± 3.5 on higher frequencies such as 800KHz. This is probably due to inconsistencies produced by the control signals from the Arduino. Nevertheless this test has passed the pass criteria as the CPS team was able to get precise readings well within the desired level.</p> <p>Moving forward the CPS team will continue to utilise an arduino seeing that there is limited time left of the project. The team will however consider other microcontroller as an alternative improvement should the project be continued.</p>
------------	------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

24.5 Subtest, height of rotor impact on inductance Sub-T-4.0.0

Table 81: Height of rotor impact on inductance Sub-T-4.0.0

Test ID	Sub-T-4.0.0
Requirement ID	-
Name	Anders Rønning & Jarand Solberg Strømmen.
Date and location	18.05.2019
Goal	Look at the change of inductance when changing the height of the rotor over the stator
Hypothesis	The rotor affects the stator less the greater the distance between them
Pass criteria	Measurements done at the distances 0.6 mm and 1.4 mm
Equipment	Waveform software, Electronics Explorer, oscilloscope, 1000 Ω resistor, 9 shims 0.60mm, stator, rotor and 3D printed mount.
Safety precautions	

Table 82: Height of rotor impact on inductance Sub-T-4.0.0

Execution	Take a measurement on each of the distances 0.6 mm and 1.4 mm while the coil tested is completely covered by the rotor. Then compare the values of each of the three distances while the coil is not covered. The way to get the rotor approximately in the same position each time was to force it into the same section of the wall of the mount each time.
Observations	The test was performed under an open container and was affected by external noise from electrical equipment nearby.
Analysis	-
Results	<p>The frequency is changing less whit bigger height difference between rotor and stator. This is because the induction in the coil is affected less the future away the rotor is.</p> <p>Figure 5 shows the frequency when at 115 μH, with the induction shown in figure 6. This applies for every set of figures. Table 83 shows that the delta frequency drops with 100 kHz</p>
Conclusion	<p>The hypothesis was correct. The inductance is affected less with a bigger gap between rotor and stator. This is important information in further use of the test station.</p> <p>The ideal test would be to change the difference in position for every 0.1 mm to understand the relationship between height of rotor and induction in the coil.</p>

Table 83: Frequency depending on height of rotor

	0% coverage	100% coverage
0.6 mm	497.805 kHz	709.879 kHz
1.4 mm	497.972 kHz	608.787 kHz

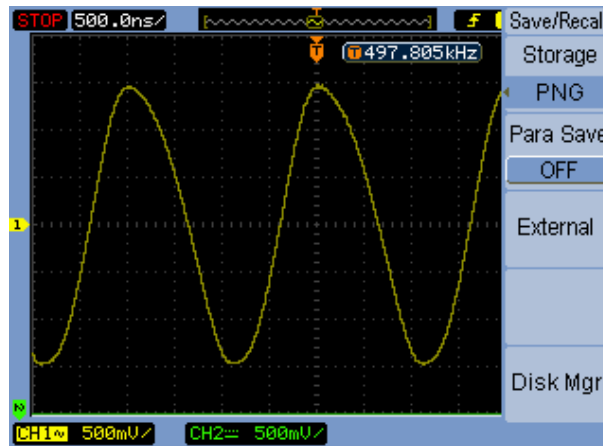


Figure 158: Frequency at 0% covered, 0.6 mm distance

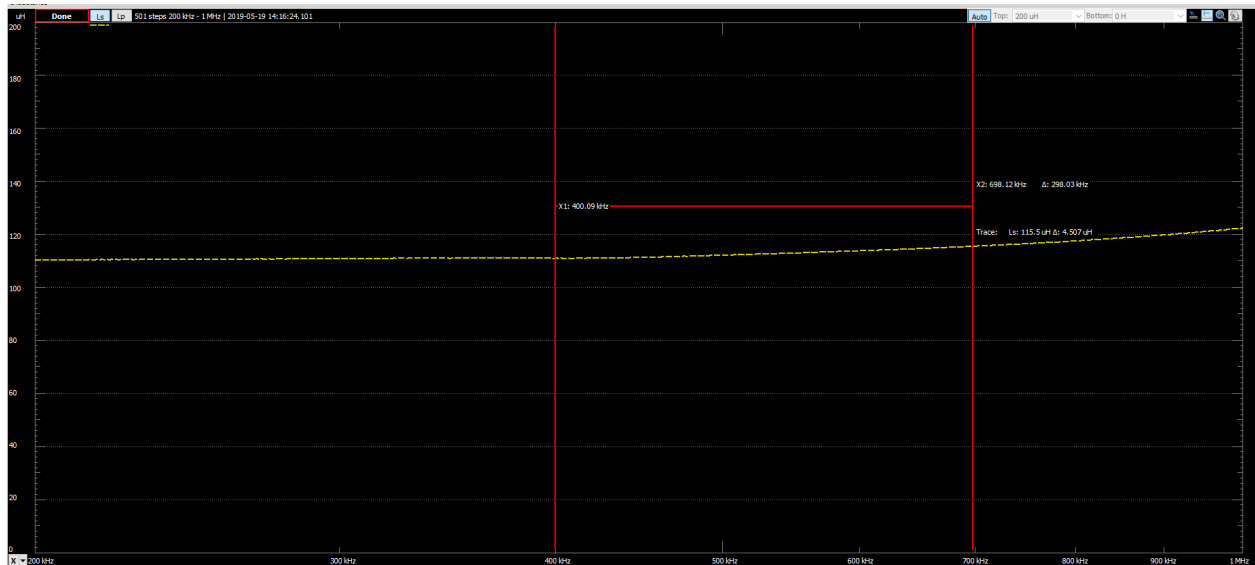


Figure 159: Inductance with 0% covered, 0.6 mm distance between rotor and stator

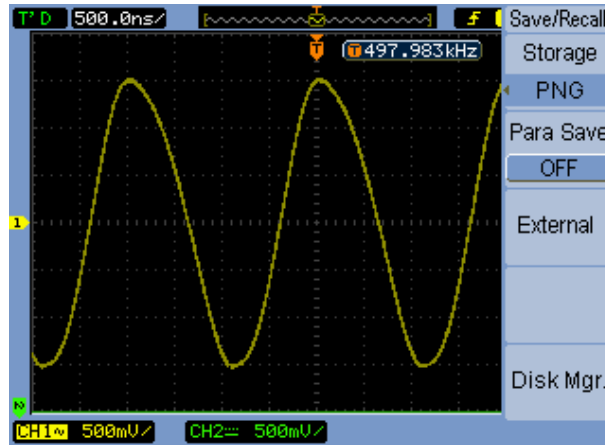


Figure 160: Frequency at 0% covered, 1.4 mm distance

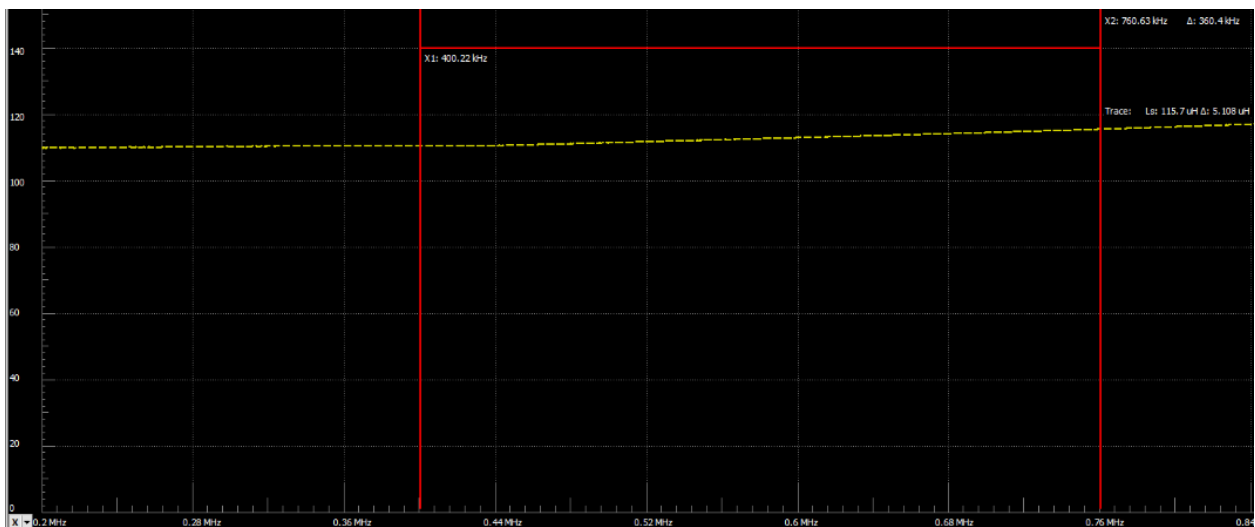


Figure 161: Inductance with 0% covered, 1.4 mm distance between rotor and stator

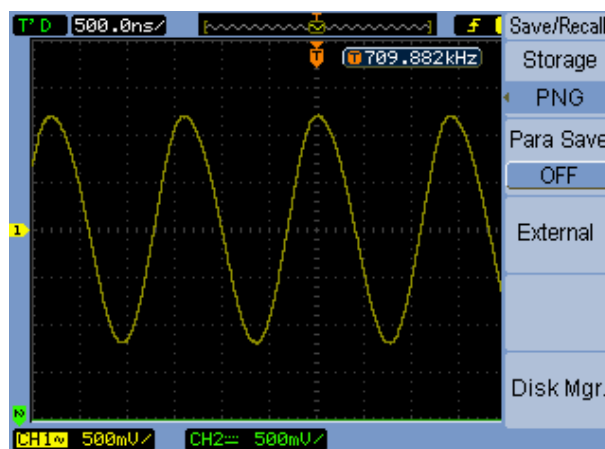


Figure 162: Frequency at 100% covered, 0.6 mm distance

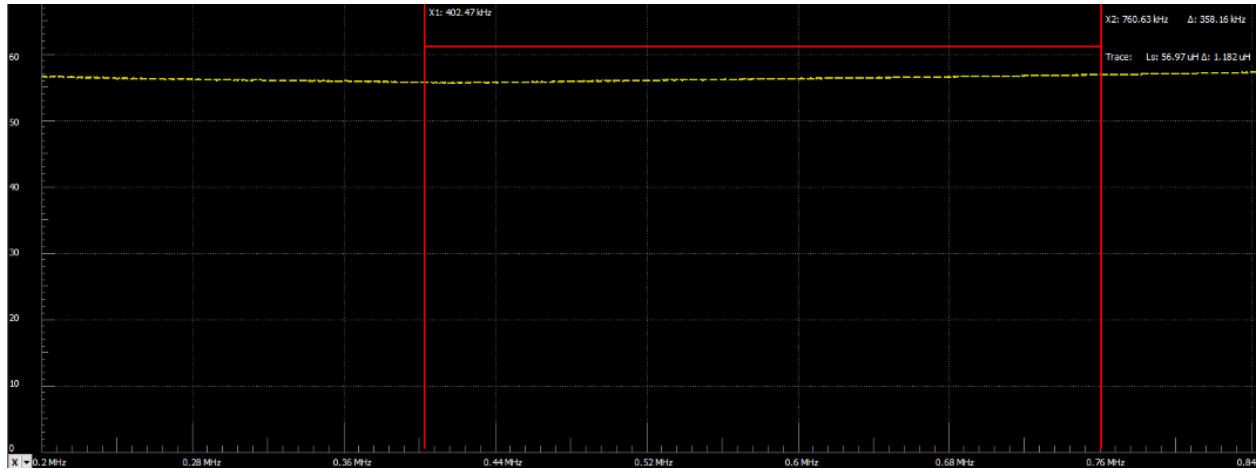


Figure 163: Inductance with 100% covered, 0.6 mm distance between rotor and stator

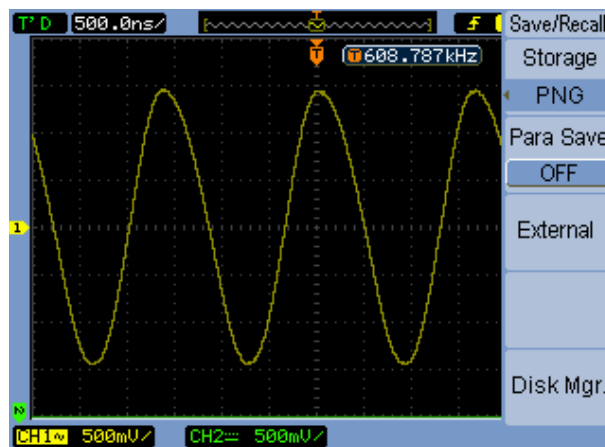


Figure 164: Frequency at 100% covered, 1.4 mm distance

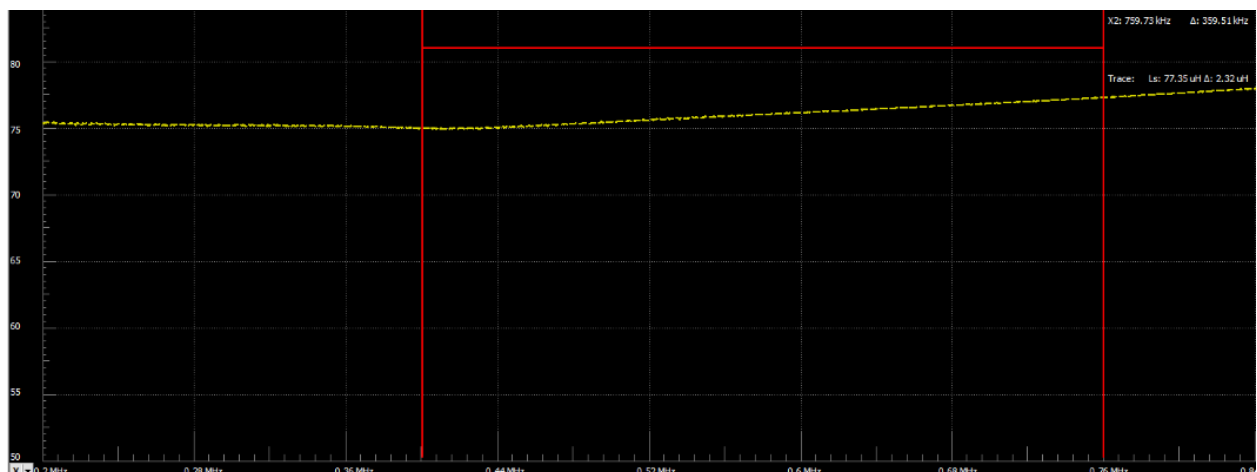


Figure 165: Inductance with 100% covered, 1.4 mm distance between rotor and stator

25 Improvements

25.1 Document history

Table 84: Improvements document history

Version	Date	Author	Description
1.0.0	22.05.2019	CPS team	Created document

25.2 Introduction

This section will explain known issues and possible solutions.

25.3 General system improvements

When testing the whole system, it was not working as intended. The problem under the first test was that the stepper motor would not rotate with less than 40 steps in distance. Thus the requirement of 0.01° cannot be tested at this point. Testing distance of 50 steps can be completed and gives a change in frequency. This indicates that the system works and a change in frequency can be detected.

When the stator and rotor were mounted in the test station the height difference was 0.52 mm. This spacing made the oscillator circuit not function. On higher frequency the feedback was too small and the signal would stop oscillating. This height difference was adjusted to 0.6 mm and it worked. When the oscillator did not function, the shims increased the height with 0.35 mm. This made the frequency span be 100 kHz. Under further testing an improvement of 0.1 mm shims shall be utilised.

25.4 Counter circuit improvements

While the counter circuit has the required specifications to count accurately [54], it is dependent on control signals. These signals are currently emitted from an Arduino Mega 2560 and as discovered in testing 24.4 there is an error margin when counting.

The main potential sources for this problem is the microcontroller. This is likely due to time inconsistency in driving pins from high to low and the other way around.

This problem could be solved by administering signals from a microcontroller or FPGA with a higher degree of consistency and accuracy. Utilising a Basys 3 FPGA board could potentially offer a solution to this problem. The Arduino DUE [4] could also offer improvements as it utilises a better processor.

25.5 Software improvements

Log files can currently only log a single position at a time or a whole calibration round. There is no in-between which could be useful to have to log multiple positions on a single test report.

Calibration can currently only be done in the highest resolution. It would be useful for this to be dynamic and work in all step resolutions, but this was not implemented. This goes for the calibration file as well, as it currently only can create a proper file if the resolution is $1/128$.

All of the user interface elements are shown in one window. The main goal with this was to provide the user with all the necessary information in one place. With many information

elements present, the graphs are a bit small. This could be solved by splitting different parts of the GUI into different windows.

Another improvement would be queuing commands. currently the user can select parameters and send that built command to the arduino. Implementing queuing of commands would enable for the user to queue multiple test runs and press run. This combined with the ability to generate a log report with multiple test data present would be beneficial.

25.6 Stepper motor improvement

When running 1 - 40 steps in 1/128 resolution the stepper motor is not able to move. It will drain more current the further it is asked to go and if it asked to go far enough, somewhere between 40 - 80 steps, it will jump to that location skipping all steps in-between. Exactly why this is happening is unknown but it is theorised that when it has such a short distance to move the change in the stepper motor's coils is too low move the stepper motor to counter the friction in the test station.

25.7 Oscillator improvements

The main improvement to the oscillator is to make it on an PCB. The reason is that a breadboard is far from perfect and has noise that is affecting the circuit. This has been a problem under development. Under development the capacitor have been a problem due to leakage current. The leakage current made the oscillator not stabilise at the exact frequency.

Shielding of the stator and rotor would also reduce the noise in the signal, this is can be done with better grounding and no loose wires from the test station. Testing with the cap of the test station attached would reduce noise from other electrical equipment nearby.

Smaller improvements would be to build the circuit with resistors with high tolerance. This to avoid errors under testing of temperature requirement 1.7 from table 18.

26 Conclusion

into the contactless position sensor

Research conducted in this report has shown that it is possible to create a contactless position sensor with an accuracy of 0.01° . This report has also shown that developing a test station with continuously rotating position is possible through design choices. The developed test station meets the requirements needed to conduct all tests regarding position. However due to the changes that had to be made during manufacturing, a spring needs to be implemented for testing in various temperature conditions.

Analysis has shown that an LC oscillator has the capability to generate a sufficient frequency span. With this frequency span it is possible to measure position accurate. Considering research, calculations and testing, the CPS team considers the sensor to be promising, but further testing of accuracy and space environment remains. The CPS team considers that the sensor has potential to be used in other position tracking systems.

27 References

- [1] Advanced-Surface-Technologies-Inc. Passivation. <https://www.astfinishing.com/wp-content/uploads/2015/07/Passivation.pdf>. Accessed 18.05.2019.
- [2] Lars Hammare ake Björkman. Grundkurs, Ritningsregler/Ritteteknik. Björkman utbildningsmaterial AB Lars Hammare utbildning AB, 1st edition, 01.01.2013.
- [3] Ambersil. Dry molybdenum disulphide lubricant. <https://docs-emea.rs-online.com/webdocs/1517/0900766b81517ef8.pdf>. Accessed 16.05.2019.
- [4] Arduino. Arduino-due. <https://store.arduino.cc/due>. Accessed 22.05.2019.
- [5] Simon Bennet. Schaum's outline of UML. McGraw-Hill, 2nd edition, 2004.
- [6] Ronald Berberg. Passivation of stainless steel. MortenHojem. Accessed 18.05.2019.
- [7] William Bolton. Mechatronics Electronic control systems in mechanical and electrical engineering. Pearson Education, 6th edition, 2015.
- [8] Pong P. Chu. FPGA prototyping by VHDL examples : Xilinx Spartan-3 version. Wiley, spartan-3 edition, 2008.
- [9] Circuitstoday. Colpitt oscillator. <http://www.circuitstoday.com/colpitts-oscillator>. Published 13.08.2018.
- [10] Mike Cohn. User stories. <https://www.mountangoatsoftware.com/agile/user-stories>. Accessed 23.01.2019.
- [11] Fairchild Semiconductor Corporat. 2n3904 npn general-purpose amplifier. <http://www.mouser.com/ds/2/149/2N3904-82270.pdf>. Published 10.2014.
- [12] KEMET Electronics Corporation. Goldmax, 300 series, conformally coated, x7r dielectric, 25 – 250 vdc (commercial grade). https://www.elfadistrelec.no/Web/Downloads/_t/ds/Goldman_X7R_eng_tds.pdf. Published 01.10.2017.
- [13] Digilent. Analog discovery technical reference manual. https://reference.digilentinc.com/_media/analog_discovery%3Aanalog_discovery_rm.pdf. Published 18.03.2015.
- [14] Digilent. Basys 3 fpga board reference manual. https://reference.digilentinc.com/_media/basys3:basys3_rm.pdf. Published 08.04.2016.
- [15] Antony Dovgal. Libharu github repository. <https://github.com/libharu>. Published 28.06.2015.
- [16] Antony Dovgal. Libharu homepage. www.libharu.org. Accessed 08.04.2019.

- [17] Emanuel Eichhammer. Introduction.
<https://www.qcustomplot.com/index.php/introduction>. Published 18.03.2015.
- [18] Electronics-tutorials. Lc oscillator basics. <https://www.electronics-tutorials.ws/oscillator/oscillators.html>. Accessed 24.03.2019.
- [19] Thomas L. Floyd. Electronic Devices. Pearson Education, 9th edition, 2014.
- [20] Richard Gratton and Dave West. Scrum reboot this time with the values™.
https://scrumorg-website-prod.s3.amazonaws.com/drupal/2017-12/Case-Study_Intarlinks-Reboot_July2017v3.pdf. Accessed 17.01.2019.
- [21] Jeremy Hill. Gdt 101: An introduction to geometric dimensioning and tolerancing.
<https://www.fictiv.com/hwg/fabricate/gdt-101-an-introduction-to-geometric-dimensioning-and-tolerancing>. Published 14.02.2017.
- [22] itemis.com. Requirements engineering with scrum. <https://www.itemis.com/en/agile/scrumpact/requirements-engineering-with-scrum/stage-2-of-the-product-backlog>. Accessed 23.01.2019.
- [23] Susan Riedel James Nilsson. Electric Circuits. Pearson Education, 10th edition, 2015.
- [24] Ron Jeffries. Essential xp: Card, conversation, confirmation.
https://ronjeffries.com/xprog/articles/expcardconversationconfirmation/?fbclid=IwAR1CeQi_L0xOIn23V8bR5sHOy7saUVqUJXJnN9DK2J7rp4pMoONWQMYYOAg. Published 30.08.2001.
- [25] Arseny Kapoulkine. Pugixml. <https://pugixml.org/>. Published 04.04.2018.
- [26] Kongsberg-Defence-&-Aerospace. Space surveillance. <https://www.kongsberg.com/en/kds/products/spacetechnologyandsystems/#>. Accessed 21.01.2019.
- [27] Kongsberg-Defence-&-Aerospace. Systems, kongsberg defence & aerospace.
<https://www.kongsberg.com/en/kds/>. Accessed 21.01.2019.
- [28] Leadshine. Em402 step motor driver.
<http://www.leadshine.com/productDetail.aspx?type=products&category=stepper-products&producttype=stepper-drives&subtype=general-stepper-drives&series=em&model=EM402>. Accessed 13.03.2019.
- [29] Lesjöfors. 4293: Ds 22,5x11,2x0,6.
<https://catalog.lesjoforsab.com/ds-22-5x11-2x0-6>. Accessed 15.05.2019.

- [30] Lesjöfors. 5028: Dsl 23,7x14,3x0,4.
<https://catalog.lesjoforsab.com/dsl-23-7x14-3x0-4>. Accessed 15.05.2019.
- [31] Kyrre Lohne. Kongsberg awarded crows contracts valued 805 mnok. <https://www.kongsberg.com/en/kds/news/2018/desember/kongsberg%20awarded%20crows%20contracts%20valued%20805%20mnok/>. Published 31.12.2018.
- [32] Longs-Motor. Longs motor dm420a stepper driver.
<http://www.longs-motor.com/stepper-motor-driver-dm542a.html>. Accessed 25.04.2019.
- [33] Jean loup Gailly Mark Adler. Libharu homepage. <https://zlib.net/>. Published 15.01.2017.
- [34] Sandra May. What is a satellite?
<https://www.nasa.gov/audience/forstudents/5-8/features/nasa-knows/what-is-a-satellite-58.html>. Published 07.08.2017.
- [35] Ian Mitchell. The definition of done pattern. <https://www.scrum.org/resources/blog/walking-through-definition-done>. Published 31.05.2017.
- [36] Ian Mitchell. The scrum guide™.
<https://www.scrum.org/resources/blog/typical-sprint-play-play>. Published 24.02.2017.
- [37] Ian Mitchell. Walking through a definition of done. <https://www.scrum.org/resources/blog/walking-through-definition-done>. Published 31.05.2017.
- [38] Mollificio-Modenese. Stainless steel wire en 10270-3. <http://tiny.cc/3ltx6y>. Accessed 15.05.2019.
- [39] Vannevar Morgan. Qt-temperature-sensor/ds18b20_qt/dialog.cpp.
https://github.com/vannevar-morgan/Qt-Temperature-Sensor/blob/master/DS18B20_Qt/dialog.cpp. Published 18.03.2015.
- [40] NanoTec. Smci33-2 driver datasheet.
https://en.nanotec.com/fileadmin/files/Baureihenuuebersichten/Motorsteuerungen/Product_Overview_SMCI33.pdf. Accessed 06.02.2019.
- [41] NanoTec. Smci33-2 driver product page.
<https://en.nanotec.com/products/1037-smci33-2/>. Accessed 06.02.2019.
- [42] NanoTec. Smci33-2 driver programming manual.
https://en.nanotec.com/fileadmin/files/Handbuecher/Programmierung/Programming_Manual_V2.7.pdf. Published 25.06.2013.

- [43] NanoTec. Smci33-2 driver technical manual.
https://en.nanotec.com/fileadmin/files/Handbuecher/Motorsteuerungen/SMCI33_Technical-Manual_V2.2.pdf. Published 11.08.2010.
- [44] NHBB-Inc. Grease dry film lubricants. <https://nhbb.com/reference/rod-ends-bearings/grease-dry-film-lubricants.aspx>. Accessed 16.05.2019.
- [45] Object-Management-Group. Object management group website.
<https://www.omg.org/index.htm>. Accessed 20.03.2019.
- [46] Guy Eric Schalnat Andreas Dilger other contributors. libpng homepage.
<http://www.libpng.org/pub/png/libpng.html>. Published 21.04.2019.
- [47] Scrum.org. Scrum value poster.
<https://www.scrum.org/resources/scrum-values-poster>. Accessed 23.01.2019.
- [48] Scrum.org. What is a sprint retrospective? <https://www.scrum.org/resources/what-is-a-sprint-retrospective>. Accessed 17.01.2019.
- [49] Scrum.org. What is a sprint review?
<https://www.scrum.org/resources/what-is-a-sprint-review>. Accessed 17.01.2019.
- [50] SKF-Group. 6000-2rshs.
<https://www.skf.com/group/products/bearings-units-housings/ball-bearings/deep-groove-ball-bearings/deep-groove-ball-bearings/index.html?designation=6000-2RSH>. Accessed 09.05.2019.
- [51] SKF-group. Bearing preload. https://www.skf.com/binary/21-299896/0901d1968065f1f4-Bearing-preload_tcm_12-299896.pdf. Accessed 09.02.2019.
- [52] SKF-group. Pole position, bearing self study guide.
<https://www.skf.com/binary/79-69177/457640.pdf>. Accessed 10.03.2019.
- [53] SMB-Bearings. Ball bearing lubricants. <https://www.smbbearings.com/technical/bearing-lubrication.html>. Accessed 16.05.2019.
- [54] Texas-Instruments. Sn74lv8154 dual 16-bit binary counters with 3-state output registers. <http://www.ti.com/lit/ds/symlink/sn74lv8154.pdf>. Published 10.2015.
- [55] Texas-Instruments. Snx414 and snx4ls14 hex schmitt-trigger inverters.
<http://www.ti.com/lit/ds/symlink/sn74ls14.pdf>. Published 11.2016.

- [56] Texas-Instruments. Snx4hc14 hex schmitt-trigger inverters. <http://www.ti.com/lit/ds/symlink/sn74hc14.pdf>. Published 12.1982, REVISED 10.2016.
- [57] The-Engineering-ToolBox. Coefficients of linear thermal expansion. https://www.engineeringtoolbox.com/linear-expansion-coefficients-d_95.html. Accessed 17.05.2019.
- [58] The-Engineering-ToolBox. Linear thermal expansion. https://www.engineeringtoolbox.com/linear-thermal-expansion-d_1379.html. Accessed 18.05.2019.
- [59] The-Nickel-Development-Institute. Design guidelines for the selection and use of stainless steel. https://www.nickelinstitute.org/media/1667/designguidelinesfortheselectionanduseofstainlesssteels_9014_.pdf. Accessed 17.05.2019.
- [60] The-Qt-company. About qt. https://wiki.qt.io/About_Qt. Published 23.05.2018.
- [61] The-Qt-Company. Qfilesystemmodel class. <https://doc.qt.io/qt-5/qfilesystemmodel.html>. Accessed 15.05.2019.
- [62] The-Qt-Company. Qiodevice class. <https://doc.qt.io/qt-5/qiodevice.html#readyRead>. Accessed 22.05.2019.
- [63] Gunther Verheyen. There's value in the scrum values. <https://guntherverheyen.com/2013/05/03/theres-value-in-the-scrum-values/>. Published 03.05.2013.
- [64] Bill Wake. Invest in good stories, and smart tasks. https://xp123.com/articles/invest-in-good-stories-and-smart-tasks/?fbclid=IwAR3XinjY8FkZ5oYr04HCZEmFo_KgeaUSO_BzfyVxdVzPpP9x7HZBZSqysFE. Published 17.08.2003.

28 Appendix A - All iterations of risk analysis

28.1 Introduction

This appendix contain all iterations, including the current, of the risk analysis.

28.2 Risk analysis - 25.01.2019

28.2.1 Hardware risks:

Table 85: Hardware risks page 1 - iteration 1

ID	Risk	Description	Root cause	First mitigating	Probability	Severity	Risk level
R1	Hardware faults				2.8	2.2	4.9
R1.1	Sensor failure				3.0	1.5	4.5
R1.1.1	External magnetic field	If the sensor starts to show incorrect data due to external magnetic field	Other electronic components near the sensor could affect the readings of the sensor	Take EMC into consideration under the construction	5.0	2.0	10.0
R1.1.2	Cracked or bent PCB	If the PCB becomes cracked or bent leaving it unusable	Structural damage due to the intensive testing as well as mishandling it	Have several backup PCBs	1.0	1.0	1.0
R1.2	Sensor limitation				2.3	2.3	4.7
R1.2.1	Weight capacity	If the weight of the sensor becomes problematic	Under testing the weight of the components for the sensor can be damaging if it is too heavy. The more weight added, the more stress for the connections	Research on material to use material with low density	2.0	2.0	4.0
R1.2.2	Continuous rotation	Problems that could arise from the required continuous rotation of the sensor	The wires to and from the sensor could tangle if handled improperly as well as the overall precision of the sensor could degrade	Run tests on the prototype borrowed from KDA	1.0	3.0	3.0
R1.2.3	Scale sensor to mechanical interface	Problems that could arise when scaling the sensor to fit the standard mechanical interface KDA use today	The sensor may or may not become more unstable with a wider area of effect	Analyse the sensor after scaling to make sure it still is a viable	4.0	2.0	8.0

Table 86: Hardware risks page 2 - iteration 1

R1.3	Test environment				3.0	2.7	5.7
R1.3.1	Extreme temperature				3.0	2.7	5.7
R1.3.1.1	Low temperatures	Problems that could arise from testing the sensor under cold temperatures	Under very low temperatures the electronics may become unreliable	Research into available material that can withstand low	4.0	3.0	12.0
R1.3.1.2	High temperatures	Problems that could arise from testing the sensor under high temperatures	Under very high temperatures the electronics may become unreliable	Research into available material that can withstand high	3.0	3.0	9.0
R1.3.2	Vibration	Problems that could arise from testing the sensor under severe vibrations	Under vibration testing the different layers in the sensor could start hitting each other	Fasten components firmly and research into material that can withstand the vibration	2.0	2.0	4.0

28.2.2 Human risks:

Table 87: Human risks page 1 - iteration 1

R2	Human risks				2.3	3.5	5.8
R2.1	Absence				3.7	1.3	5.0
R2.1.1	Absence within the CPS team	If a member of the CPS team is absent	The members can fall ill or have other events happening in their life that require them to be elsewhere	Early notification to the rest of the team so precautions can be made for their	5.0	1.0	5.0
R2.1.2	Absent internal censor	If the internal censor becomes unavailable to attend an activity	The internal censor has other responsibilities that may take priority	Mutiple possible dates for the activity to better fit their	1.0	2.0	2.0
R2.1.3	Absent external censor	If the external censor becomes unavailable to attend an activity	The external censor has other responsibilities that may take priority	Mutiple possible dates for the activity to better fit their	5.0	1.0	5.0
R2.2	Data unavailable	If some data is not available to the whole team at crucial points	Some may have data local on their own computer when it needs to be integrated to the main document	All documentatio n is to be available online even when it's	1.0	5.0	5.0

Table 88: Human risks page 2 - iteration 1

R2.3	Damaged equipment				1.0	4.0	5.0
R2.3.1	Borrowed equipment	If accidents happens where some of the borrowed equipment is damaged	Accidents can happend where the borrowed equipment is damaged due to improper handling or carelessness	Do not test anything before the proper procedures are known and can be	1.0	4.0	4.0
R2.3.2	CPS teams equipment	If accidents happens where some of the CPS teams equipment is damaged	Accidents can happend where the personal equipment is damaged due to improper handling or carelessness	Do not test anything before the proper procedures are known and can be	1.0	4.0	4.0
R2.4	New risks	If new risks are discovered that previously was unplanned for	There are many points of views to consider when discovering risks and it is as good as impossible to count for all aspects of a project	Ask the team about it. Make it a point in the next sprint or add to current one if it is vital	5.0	2.0	10.0
R2.5	Plagiarism	If a member of the CPS team copies others work and claims it as their own	If the CPS team don't use refrences in a proper way so some of the documentation becomes plagiarism	Ask internal adviser if the refrences is proper and all CPS team member can question content if it seems off	1.0	5.0	5.0

28.2.3 Management risks:

Table 89: Management risks - iteration 1

R3	Management risks				3.0	3.5	6.5
R3.1	Time management	If the CPS team does not meet time limits	Events can happend that takes time to solve properly	Follow scrum and do the highest priority task	1.0	5.0	5.0
R3.2	Cost	If unexpected costs appear	If the CPS team needs to buy in additional equipment or components to be able to continue the project	Overestimate what equipment cost	3.0	4.0	12.0
R3.3	Delivery	If deliveries of components take longer than anticipated	Deliveries that requires shipping can be uncertain if it will arrive in time	Order as soon as possible and look for backup parts	3.0	3.0	9.0
R3.4	New requirements	If the stakeholder arrives with new requirements	New requirements can be introduced all along the project timeline and require rework of the design	Make it a point in the next sprint or add to current one if vital	5.0	2.0	10.0

28.3 Risk analysis - 27.02.2019

28.3.1 Hardware risks:

Table 90: Hardware risks page 1 - iteration 2

ID	Risk	Description	Root cause	Mitigating actions done	Mitigating action	Probability	Severity	Risk level
R1	Hardware faults					2,6	2,1	5,4
R1.1	Sensor failure					3,0	1,5	4,5
R1.1.1	External magnetic field	If the sensor starts to show incorrect data due to external magnetic field	Other electronic components near the sensor could affect the readings of the sensor	Taken action to properly ground the PCB	Take EMC into consideration under the construction phase	5,0	2,0	10,0
R1.1.2	Cracked or bent PCB	If the PCB becomes cracked or bent under transportation or assembly leaving it unusable	Structural damage due to the intensive testing as well as mishandling it		Have several backup PCBs. Correctly assemble the PCB to the test station	1,0	1,0	1,0
R1.2	Sensor limitation					2,3	2,3	5,4
R1.2.1	Weight capacity	If the weight of the sensor becomes problematic	Under testing the weight of the components for the sensor can be damaging if it is too heavy. The more weight added, the more stress for the connections		Research on material to use material with low density	2,0	2,0	4,0
R1.2.2	Continuous rotation	Problems that could arise from the required continuous rotation of the sensor	The wires to and from the sensor could tangle if handled improperly as well as the overall precision of the sensor could degrade	This will not happen due to choices in design	Run tests on the prototype borrowed from KDA	1,0	3,0	3,0
R1.2.3	Scale sensor to mechanical interface	Problems that could arise when scaling the sensor to fit the standard mechanical interface KDA use today	The sensor may become more unstable due to noise		Analyse the sensor after scaling to make sure it still is viable. Choose frequency range less affected by external noise	4,0	2,0	8,0

Table 91: Hardware risks page 2 - iteration 2

R1.3	Test environment					2,5	2,3	5,8
R1.3.1	Extreme temperature					3,0	2,7	8,0
R1.3.1.1	Low temperatures	Problems that could arise from testing the sensor under cold temperatures	Under very low temperatures the electronics may become unreliable		Research into available material that can withstand low temperatures	4,0	3,0	12,0
R1.3.1.2	High temperatures	Problems that could arise from testing the sensor under high temperatures	Under very high temperatures the electronics may become unreliable		Research into available material that can withstand high temperatures	3,0	3,0	9,0
R1.3.2	Vibration	Problems that could arise from testing the sensor under severe vibrations	Under vibration testing the different layers in the sensor could start hitting each other		Fasten components firmly and research into material that can withstand the vibration	2,0	2,0	4,0

28.3.2 Human risks:

Table 92: Human risks page 1 - iteration 2

R2	Human risks					2,1	3,7	7,7
R2.1	Absence					2,3	2,7	6,2
R2.1.1	Absence within the CPS team	If a member of the CPS team is absent	The members can fall ill or have other events happening in their life that require them to be elsewhere		Early notification to the rest of the team so precautions can be made for their absence	5,0	1,0	5,0
R2.1.2	Absent internal censor	If the internal censor becomes unavailable to attend an activity	The internal censor has other responsibilities that may take priority		Mutiple possible dates for the activity to better fit their schedule	1,0	2,0	2,0
R2.1.3	Absent external censor	If the external censor becomes unavailable to attend an activity	The external censor has other responsibilities that may take priority		Mutiple possible dates for the activity to better fit their schedule	1,0	5,0	5,0
R2.2	Data unavailable	If some data is not available to the whole team at crucial points	Some may have data local on their own computer when it needs to be integrated to the main document	The CPS team has frequently uploaded data to a cloud server. Backup downloaded weekly.	All documentation is to be available online even when it's being written	1,0	5,0	5,0
R2.3	Damaged equipment					1,0	4,0	4,0
R2.3.1	Borrowed equipment	If accidents happens where some of the borrowed equipment is damaged	Accidents can happend where the borrowed equipment is damaged due to improper handling or carelessness		Do not test anything before the proper procedures are known and can be used	1,0	4,0	4,0
R2.3.2	CPS teams equipment	If accidents happens where some of the CPS teams equipment is damaged	Accidents can happend where the personal equipment is damaged due to improper handling or carelessness		Do not test anything before the proper procedures are known and can be used	1,0	4,0	4,0

Table 93: Human risks page 2 - iteration 2

R2.4	New risks	If new risks are discovered that previously was unplanned for	There are many points of views to consider when discovering risks and it is as good as impossible to count for all aspects of a project		Ask the team about it. Make it a point in the next sprint or add to current one if it is vital	5,0	2,0	10,0
R2.5	Plagiarism	If a member of the CPS team copies others work and claims it as their own	If the CPS team don't use references in a proper way so some of the documentation becomes plagiarism		Ask internal adviser if the references is proper and all CPS team member can question content if it seems off	1,0	5,0	5,0

28.3.3 Management risks:

Table 94: Management risks - iteration 2

R3	Management risks					3,0	3,8	11,3
R3.1	Time management	If the CPS team does not meet time limits	Events can happend that takes time to solve properly		Follow scrum and do the highest priority task first	2,0	5,0	10,0
R3.2	Cost	If unexpected costs appear	If the CPS team needs to buy in additional equipment or components to be able to continue the project		Overestimate what equipment cost	3,0	4,0	12,0
R3.3	Delivery	If deliveries of components take longer than anticipated	Deliveries that requires shipping can be uncertain if it will arrive in time		Order as soon as possible and look for backup parts	3,0	3,0	9,0
R3.4	New requirements	If the stakeholder arrives with new requirements	New requirements can be introduced all along the project timeline and require rework of the design		Make it a point in the next sprint or add to current one if vital	4,0	3,0	12,0

28.4 Risk analysis - 20.03.2019

28.4.1 Hardware risks:

Table 95: Hardware risks page 1 - iteration 3

ID	Risk	Description	Root cause	Mitigating actions done	Mitigating action	Probability	Severity	Risk level
R1	Hardware faults					2,7	2,2	5,7
R1.1	Sensor failure					3,0	1,5	4,5
R1.1.1	External magnetic field	If the sensor starts to show incorrect data due to external magnetic field	Other electronic components near the sensor could affect the readings of the sensor	Taken action to properly ground the PCB	Take EMC into consideration under the construction phase	5,0	2,0	10,0
R1.1.2	Cracked or bent PCB	If the PCB becomes cracked or bent leaving it unusable	Structural damage due to the intensive testing as well as mishandling it		Have several backup PCBs. Correctly assemble the PCB to the test station	1,0	1,0	1,0
R1.2	Sensor limitation					2,3	2,3	5,4
R1.2.1	Weight capacity	If the weight of the sensor becomes problematic	Under testing the weight of the components for the sensor can be damaging if it is too heavy. The more weight added, the more stress for the connections		Research on material to use material with low density	2,0	2,0	4,0
R1.2.2	Continuous rotation	Problems that could arise from the required continuous rotation of the sensor	The wires to and from the sensor could tangle if handled improperly as well as the overall precision of the sensor could degrade	This will not happen due to choices in design	Run tests on the prototype borrowed from KDA	1,0	3,0	3,0
R1.2.3	Scale sensor to mechanical interface	Problems that could arise when scaling the sensor to fit the standard mechanical interface KDA use today	The sensor may become more unstable due to noise		Analyse the sensor after scaling to make sure it still is viable. Choose frequency range less affected by external noise	4,0	2,0	8,0

Table 96: Hardware risks page 2 - iteration 3

R1.3	Test environment					2,6	2,6	6,9
R1.3.1	Extreme temperature					3,3	3,3	10,6
R1.3.1.1	Low temperatures - Sensor data	Problems that could arise from testing the sensor under cold temperatures	Under very low temperatures the electronics may become unreliable		Research into available material that can withstand low temperatures. Research effect of low temperatures on coils	4,0	3,0	12,0
R1.3.1.2	Low temperatures - Fractured PCB	Fracured PCB under low temperature due to shrinking	Materials shrinks when decreasing the temperature		Research into available material that can withstand low temperatures. Research effect of low temperatures on PCB	2,0	4,0	8,0
R1.3.1.3	High temperatures - Sensor data	Problems that could arise from testing the sensor under high temperatures	Under very high temperatures the electronics may become unreliable		Research into available material that can withstand high temperatures	3,0	3,0	9,0
R1.3.1.4	High temperatures - Defect components	Components failing due to high temperatures	Wires and components could become defective under high		Shielding components and material choice	2,0	3,0	6,0
R1.3.2	Vibration	Problems that could arise from testing the sensor under severe vibrations	Under vibration testing the different layers in the sensor could start hitting each other		Fasten components firmly and research into material that can withstand the vibration	2,0	2,0	4,0

28.4.2 Human risks:

Table 97: Human risks page 1 - iteration 3

R2	Human risks					1,9	4,1	7,6
R2.1	Absence					2,3	2,3	5,4
R2.1.1	Absence within the CPS team	If a member of the CPS team is absent	The members can fall ill or have other events happening in their life that require them to be elsewhere		Early notification to the rest of the team so precautions can be made for their absence	5,0	3,0	15,0
R2.1.2	Absent internal censor	If the internal censor becomes unavailable to attend an activity	The internal censor has other responsibilities that may take priority		Mutple possible dates for the activity to better fit their schedule	1,0	1,0	1,0
R2.1.3	Absent external censor	If the external censor becomes unavailable to attend an activity	The external censor has other responsibilities that may take priority	Early invitaions have been sent to presentation dates and multiple possible dates available	Mutple possible dates for the activity to better fit their schedule	1,0	3,0	3,0
R2.2	Data unavailable	If some data is not available to the whole team at crucial points	Some may have data local on their own computer when it needs to be integrated to the main document	The CPS team has frequently uploaded data to a cloud server. Backup downloaded weekly.	All documentation is to be available online even when it's being written	1,0	5,0	5,0
R2.3	Damaged equipment					1,0	5,0	5,0
R2.3.1	Borrowed equipment	If accidents happens where some of the borrowed equipment is damaged	Accidents can happend where the borrowed equipment is damaged due to improper handling or carelessness		Do not test anything before the proper procedures are known and can be used	1,0	5,0	5,0
R2.3.2	CPS teams equipment	If accidents happens where some of the CPS teams equipment is damaged	Accidents can happend where the personal equipment is damaged due to improper handling or carelessness		Do not test anything before the proper procedures are known and can be used	1,0	5,0	5,0

Table 98: Human risks page 2 - iteration 3

R2.4	New risks	If new risks are discovered that previously was unplanned for	There are many points of views to consider when discovering risks and it is as good as impossible to count for all aspects of a project		Ask the team about it. Make it a point in the next sprint or add to current one if it is vital	4,0	3,0	12,0
R2.5	Plagiarism	If a member of the CPS team copies others work and claims it as their own	If the CPS team don't use references in a proper way so some of the documentation becomes plagiarism		Ask internal adviser if the references is proper and all CPS team member can question content if it seems off	1,0	5,0	5,0

28.4.3 Management risks:

Table 99: Management risks - iteration 3

R3	Management risks					2,8	3,8	10,3
R3.1	Time management					3,0	4,0	12,0
R3.1.1	Documents ready for turn in	If the CPS team does not meet time limits to turn in documentations	Events can happend that takes time to solve properly		Follow scrum and do the highest priority task first	1,0	5,0	5,0
R3.1.2	Meeting all requirements	If CPS team can fulfill the stakeholder requirements before the project end	Challenges appear along the way and other events can happend to delay the project to the point where meeting all the requirements are not possible		Strict structural approach throughout the project	5,0	3,0	15,0
R3.2	Cost	If unexpected costs appear	If the CPS team needs to buy in additional equipment or components to be able to continue the project		Overestimate what equipment cost. Consult the group before items are bought	2,0	3,0	6,0
R3.3	Delivery	If deliveries of components take longer than anticipated	Deliveries that requires shipping can be uncertain if it will arrive in time		Order as soon as possible and look for backup parts	3,0	4,0	12,0
R3.4	New requirements	If the stakeholder arrives with new requirements	New requirements can be introduced all along the project timeline and require rework of the design	Consulted with employer that new requirements are unlikely to happen	Make it a point in the next sprint or add to current one if vital	3,0	4,0	12,0

28.5 Risk analysis - 1.05.2019

28.5.1 Hardware risks:

Table 100: Hardware risks page 1 - iteration 4

ID	Risk	Description	Root cause	Mitigating actions done	Mitigating action	Probability	Severity	Risk level
R1	Hardware faults					2,4	2,0	5,0
R1.1	Sensor failure					3,0	1,5	4,5
R1.1.1	External magnetic field	If the sensor starts to show incorrect data due to external magnetic field	Other electronic components near the sensor could affect the readings of the sensor	Taken action to properly ground the PCB	Take EMC into consideration under the construction phase	5,0	2,0	10,0
R1.1.2	Cracked or bent PCB	If the PCB becomes cracked or bent leaving it unusable	Structural damage due to the intensive testing as well as mishandling it	Multiple PCBs has been ordered and delivered	Have several backup PCBs. Correctly assemble the PCB to the test station	1,0	1,0	1,0
R1.2	Sensor limitation					1,7	2,0	3,3
R1.2.1	Weight capacity	If the weight of the sensor becomes problematic	Under testing the weight of the components for the sensor can be damaging if it is too heavy. The more weight added, the more stress for the connections		Research on material to use material with low density	2,0	1,0	2,0
R1.2.2	Continuous rotation	Problems that could arise from the required continuous rotation of the sensor	The wires to and from the sensor could tangle if handled improperly as well as the overall precision of the sensor could degrade	This will not happen due to choices in design	Run tests on the prototype borrowed from KDA	1,0	3,0	3,0
R1.2.3	Scale sensor to mechanical interface	Problems that could arise when scaling the sensor to fit the standard mechanical interface KDA use today	The sensor may become more unstable due to noise	Proper research, testing has been done. The distance between coils on stator helps to eliminate noise	Analyse the sensor after scaling to make sure it still is viable. Choose frequency range less affected by external noise	2,0	2,0	4,0

Table 101: Hardware risks page 2 - iteration 4

R1.3	Test environment					2,6	2,6	6,9
R1.3.1	Extreme temperature					3,3	3,3	10,6
R1.3.1.1	Low temperatures - Sensor data	Problems that could arise from testing the sensor under cold temperatures	Under very low temperatures the electronics may become unreliable		Research into available material that can withstand low temperatures. Research effect of low temperatures on coils	4,0	3,0	12,0
R1.3.1.2	Low temperatures - Fractured PCB	Fracured PCB under low temperature due to shrinking	Materials shrinks when decreasing the temperature	Verification from external advisor that current PCB material is suitable for low temperature	Research into available material that can withstand low temperatures. Research effect of low temperatures on PCB	2,0	4,0	8,0
R1.3.1.3	High temperatures - Sensor data	Problems that could arise from testing the sensor under high temperatures	Under very high temperatures the electronics may become unreliable		Research into available material that can withstand high temperatures	3,0	3,0	9,0
R1.3.1.4	High temperatures - Defect components	Components failing due to high temperatures	Wires and components could become defective under high temperatures	Choice in material in the PCB is suitable for high temperatures	Shielding components and material choice	2,0	3,0	6,0
R1.3.2	Vibration	Problems that could arise from testing the sensor under severe vibrations	Under vibration testing the different layers in the sensor could start hitting each other	The stiffness of the PCB and how it is fixed helps to avoid this	Fasten components firmly and research into material that can withstand the vibration	2,0	2,0	4,0

28.5.2 Human risks:

Table 102: Human risks page 1 - iteration 4

R2	Human risks					1,5	4,6	7,1
R2.1	Absence					1,7	3,0	5,0
R2.1.1	Absence within the CPS team	If a member of the CPS team is absent	The members can fall ill or have other events happening in their life that require them to be elsewhere		Early notification to the rest of the team so precautions can be made for their absence	3,0	5,0	15,0
R2.1.2	Absent internal censor	If the internal censor becomes unavailable to attend an activity	The internal censor has other responsibilities that may take priority		Mutiple possible dates for the activity to better fit their schedule	1,0	1,0	1,0
R2.1.3	Absent external censor	If the external censor becomes unavailable to attend an activity	The external censor has other responsibilities that may take priority	Early invitaions have been sent to presentation dates and multiple possible dates available	Mutiple possible dates for the activity to better fit their schedule	1,0	3,0	3,0
R2.2	Data unavailable	If some data is not available to the whole team at crucial points	Some may have data local on their own computer when it needs to be integrated to the main document	The CPS team has frequently uploaded data to a cloud server. Backup downloaded weekly.	All documentation is to be available online even when it's being written	1,0	5,0	5,0
R2.3	Damaged equipment					1,0	5,0	5,0
R2.3.1	Borrowed equipment	If accidents happens where some of the borrowed equipment is damaged	Accidents can happend where the borrowed equipment is damaged due to improper handling or carelessness	Read data sheet and taken precaution to avoid possible damaging scenarios for componets	Do not test anything before the proper procedures are known and can be used	1,0	5,0	5,0
R2.3.2	CPS teams equipment	If accidents happens where some of the CPS teams equipment is damaged	Accidents can happend where the personal equipment is damaged due to improper handling or carelessness	Read data sheet and taken precaution to avoid possible damaging scenarios for componets	Do not test anything before the proper procedures are known and can be used	1,0	5,0	5,0

Table 103: Human risks page 2 - iteration 4

R2.4	New risks	If new risks are discovered that previously was unplanned for	There are many points of views to consider when discovering risks and it is as good as impossible to count for all aspects of a project		Ask the team about it. Make it a point in the next sprint or add to current one if it is vital	3,0	5,0	15,0
R2.5	Plagiarism	If a member of the CPS team copies others work and claims it as their own	If the CPS team don't use references in a proper way so some of the documentation becomes plagiarism	Proper use of references from sources when gathering information	Ask internal adviser if the references is proper and all CPS team member can question content if it seems off	1,0	5,0	5,0

28.5.3 Management risks:

Table 104: Management risks - iteration 4

R3	Management risks					2,8	4,8	13,1
R3.1	Time management					3,0	4,0	12,0
R3.1.1	Documents ready for turn in	If the CPS team does not meet time limits to turn in documentations	Events can happend that takes time to solve properly		Follow scrum and do the highest priority task first	1,0	5,0	5,0
R3.1.2	Document completed	Everything important added and ready for turn in	Not enough time	Internal deadline one week before the proper deadline		3,0	4,0	12,0
R3.1.3	Not meeting all requirements	If CPS team is not capable to fulfill the stakeholder requirements before the project end	Challenges appear along the way and other events can happend to delay the project to the point where meeting all the requirements is not possible		Strict structural approach throughout the project	5,0	3,0	15,0
R3.2	Cost	If unexpected costs appear	If the CPS team needs to buy in additional equipment or components to be able to continue the project		Overestimate what equipment cost. Consult the group before items are bought	2,0	5,0	10,0
R3.3	Delivery	If deliveries of components take longer than anticipated	Deliveries that requires shipping can be uncertain if it will arrive in time		Order as soon as possible and look for backup parts	5,0	5,0	25,0
R3.4	New requirements	If the stakeholder arrives with new requirements	New requirements can be introduced all along the project timeline and require rework of the design	Consulted with employer that new requirements are unlikely to happen	Make it a point in the next sprint or add to current one if vital	1,0	5,0	5,0

29 Appendix B - Working with Qt

29.1 Document history

Table 105: Learning Qt document history

Version	Date	Author	Description
1.0.0	26.03.2019	MBC, HS & JS	Document created.Proofreading and corrections.
2.0.0	17.05.2019	MBC	Changed setup of sections and moved this section to appendix.

29.2 Introduction

This section will explain the activities that went in to learning Qt, and how it is related to the end software product.

29.3 Potentiometer values on a graphical user interface

The Qt framework is an extensive framework as explained in section 20, and despite knowing C++, learning the Qt framework was necessary. At the end of the project the system will read data from the contactless position sensor, thus the team decided to learn Qt by projecting a potentiometer. The potentiometer value was to be sent by arduino to the GUI, and if the user wanted to, log the sample data.

29.4 Arduino code to read and send potentiometer value

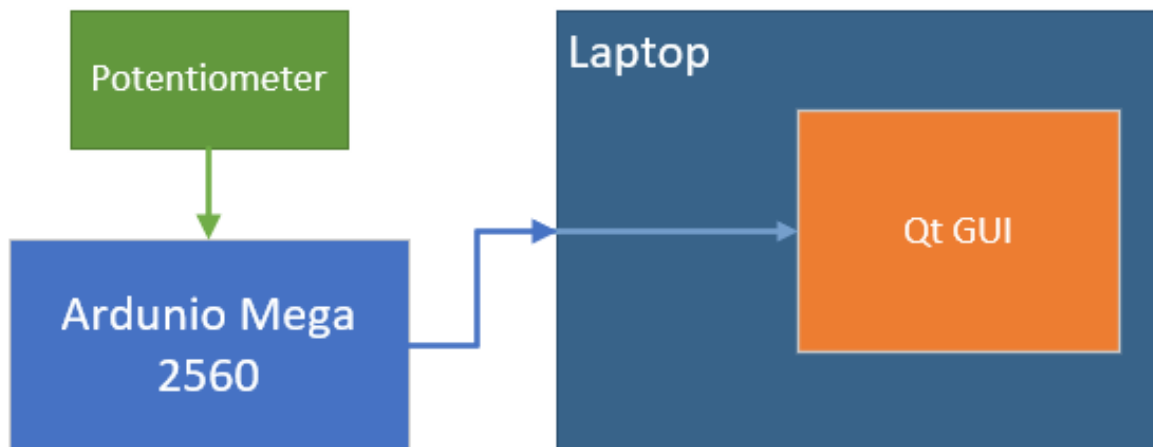


Figure 166: Reading potentiometer on GUI system

Figure 166 illustrates how the potentiometer is connected to the arduino, which sends data to the GUI through serial communication. The arduino code is listed in listing 6.

```
1 //Initialise the potmeter pin.
2 int potPin = A12;
3 //Integer to store potvalue.
4 int val = 0;
5
6 void setup() {
7   //Start serialcommunication
8   //at baudrate 9600.
9   Serial.begin(9600);
10 }
```



```
11
12 void loop() {
13     //Store the potentiometer
14     //value read from analogpin
15     //potpin.
16     val = analogRead(potPin);
17     //Print the value over serial
18     //communication and flush the
19     //serial channel
20     Serial.print(val);
21     Serial.print(",");
22     Serial.flush();
23     //Then wait for 50ms before
24     //looping over again.
25     delay(50);
26 }
```

Listing 6: Arduino code to read and send potentiometer values

The arduino code loops through a sequence of reading analog data and storing it in an integer. Then this value is sent over serial communication, followed by a comma. Then the serial communication is flushed and the program waits 50ms before repeating.

29.5 Potentiometer reading GUI

The comma that is sent after the potentiometer value has the effect of an escape sign. When data is sent over serial communication, it is not guaranteed that all the data will be received at once. Data can be split and result in incorrect readings. This can be prevented by having a buffer on the receiving end, that stores everything in the first element of a list until a comma is received. This has been implemented by a GitHub user called Vannevar-Morgan. How the list is split up can be viewed in the `dialog.cpp` file at the Vannevar-Morgan github [39]. Not wanting to spend time "reinventing the wheel", this solution was implemented.

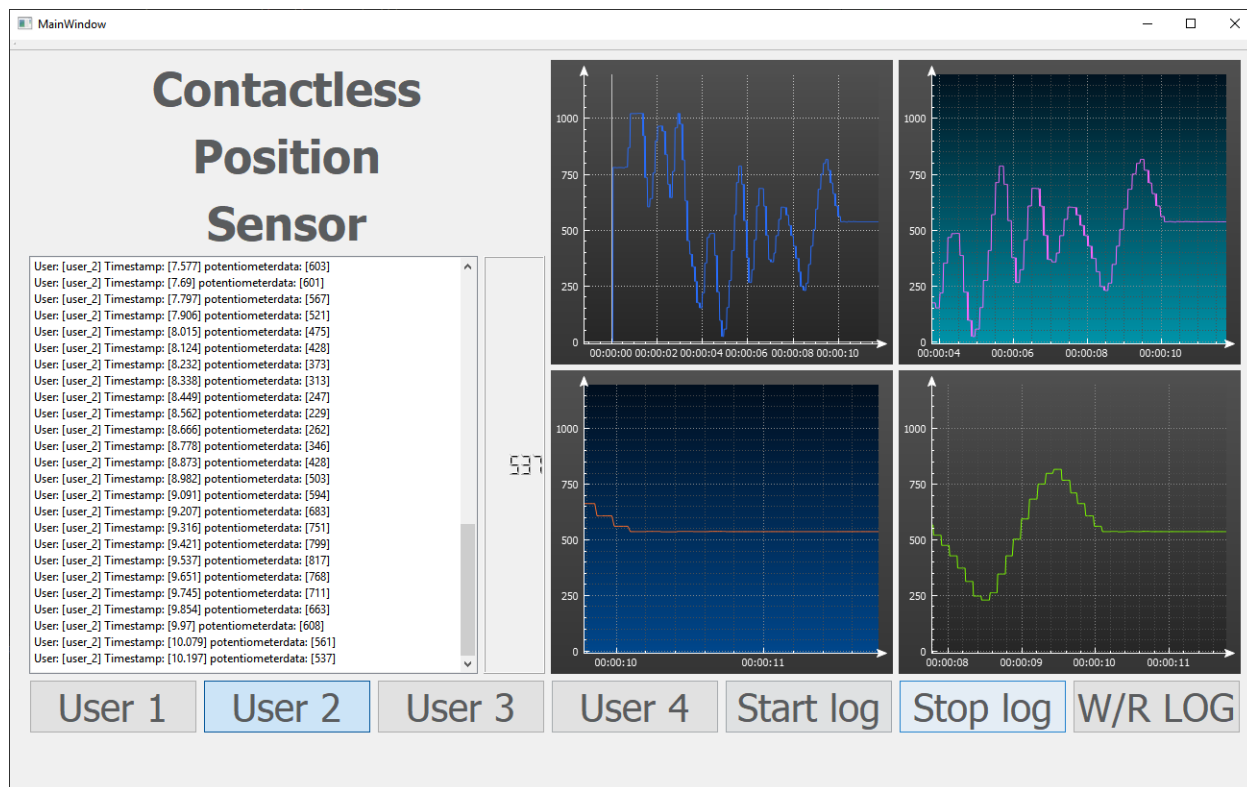


Figure 167: Qt potentiometer reading GUI

The GUI created in Qt to read the potentiometer value is illustrated in figure 167. This GUI allows for selection of up to 4 users and the option to start and stop the logging of files. The user also has the option to write .txt file and reset the log. The log, is the window with text that is located under the title "Contactless Position Sensor". The graphs are created using an external library called QCustomPlot [17]. The potentiometer value is also displayed in digits between the graphs and the log window.

29.6 Working with Qt conclusion

Working on this has given a better understanding of how Qt works, and how Qt can be used to the final software solution. The development of the potentiometer reader GUI has not been done with the layers architecture as a focus, as the main objective was to learn the tools. The full Qt Solution and and clean text version can be found in the attached folder QtAttachments.

30 Appendix C - Directory structure doxygen documentation

30.1 Introduction

This appendix contains the doxygen documentation for the directory structure code as of 25.03.2019.

My Project

Generated by Doxygen 1.8.15

1 Class Index	1
1.1 Class List	1
2 Class Documentation	3
2.1 fileGenerate Class Reference	3
2.1.1 Detailed Description	3
2.1.2 Member Function Documentation	3
2.1.2.1 fileStruct()	3
2.1.2.2 gen()	4

Chapter 1

Class Index

1.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

[fileGenerate](#) 3

Chapter 2

Class Documentation

2.1 fileGenerate Class Reference

Public Member Functions

- void [gen](#) ()
Test function for new approaches.
- void [fileStruct](#) (std::string &prName, std::string &cName, std::string &tID, std::string &peName, std::string ¬e, std::string &sMode, std::string &cur, std::string &dist, std::string &spd, std::string &acc)
Function to create directory and xml file.

2.1.1 Detailed Description

Definition at line 5 of file fileGenerate.h.

2.1.2 Member Function Documentation

2.1.2.1 fileStruct()

```
void fileGenerate::fileStruct (
    std::string & prName,
    std::string & cName,
    std::string & tID,
    std::string & peName,
    std::string & note,
    std::string & sMode,
    std::string & cur,
    std::string & dist,
    std::string & spd,
    std::string & acc )
```

Function to create directory and xml file.

Current working function for generating a directory structure and xml file to log parameters and settings from an external source. The paramaters are all the variables that will be gathered from the GUI

Parameters

<i>prName</i>	A string parameter for the project name
<i>cName</i>	A string parameter for the component id
<i>tID</i>	A string parameter for the test id
<i>peName</i>	A string parameter for the personnel name
<i>note</i>	A string parameter for notes about the test
<i>sMode</i>	A string parameter for stepmode setting of the driver
<i>cur</i>	A string parameter for the current setting of the driver
<i>dist</i>	A string parameter for the distance setting of the driver
<i>spd</i>	A string parameter for the speed setting of the driver
<i>acc</i>	A string parameter for the acceleration setting of the driver

Definition at line 136 of file fileGenerate.cpp.

2.1.2.2 gen()

```
void fileGenerate::gen ( )
```

Test function for new approaches.

This is a test function only used to test new approaches and is not part of the end code

Definition at line 41 of file fileGenerate.cpp.

The documentation for this class was generated from the following files:

- C:/Users/magnu/Documents/Bachelor project testing/fileGenerate.h
- C:/Users/magnu/Documents/Bachelor project testing/fileGenerate.cpp

31 Appendix D - Potentiometer reader GUI doxygen documentation

31.1 Introduction

This appendix contains the doxygen documentation for the Potentiometer reader GUI as of 26.03.2019.

QtLearn

Generated by Doxygen 1.8.15

1 Namespace Index	1
1.1 Namespace List	1
2 Hierarchical Index	3
2.1 Class Hierarchy	3
3 Class Index	5
3.1 Class List	5
4 Namespace Documentation	7
4.1 Ui Namespace Reference	7
4.1.1 Detailed Description	7
5 Class Documentation	9
5.1 MainWindow Class Reference	9
5.1.1 Detailed Description	11
5.1.2 Constructor & Destructor Documentation	11
5.1.2.1 MainWindow()	11
5.1.2.2 ~MainWindow()	12
5.1.3 Member Function Documentation	12
5.1.3.1 graphStyle	12
5.1.3.2 on_button_start_log_clicked	12
5.1.3.3 on_button_stop_log_clicked	12
5.1.3.4 on_button_user_1_clicked	12
5.1.3.5 on_button_user_2_clicked	13
5.1.3.6 on_button_user_3_clicked	13
5.1.3.7 on_button_user_4_clicked	13
5.1.3.8 on_button_write_reset_clicked	13
5.1.3.9 readSerialPort	13
5.1.3.10 realtimeDataSlot	13
5.1.3.11 setupRealtimeData	14
5.1.3.12 updatePotList	14
5.1.3.13 updatePotVal	14
5.1.4 Member Data Documentation	14
5.1.4.1 arduino	15
5.1.4.2 arduinoProductId	15
5.1.4.3 arduinoVendorId	15
5.1.4.4 button1Pressed	15
5.1.4.5 button2Pressed	16
5.1.4.6 button3Pressed	16
5.1.4.7 button4Pressed	16
5.1.4.8 currentUser	16
5.1.4.9 dataRecieved	17
5.1.4.10 dataTimer	17

5.1.4.11 documentNumber	17
5.1.4.12 ok	17
5.1.4.13 parsedData	18
5.1.4.14 QStr	18
5.1.4.15 serialBuffer	18
5.1.4.16 serialData	18
5.1.4.17 startLog	19
5.1.4.18 tmpQstring	19
5.1.4.19 ui	19

Chapter 1

Namespace Index

1.1 Namespace List

Here is a list of all documented namespaces with brief descriptions:

Ui	Namespace	7
--------------------	---------------------	-------------------

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

- QMainWindow
- MainWindow 9

Chapter 3

Class Index

3.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

MainWindow	
Mainwindow class	9

Chapter 4

Namespace Documentation

4.1 Ui Namespace Reference

Namespace.

4.1.1 Detailed Description

Namespace.

This provides a namespace.

Chapter 5

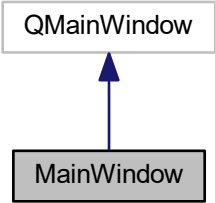
Class Documentation

5.1 MainWindow Class Reference

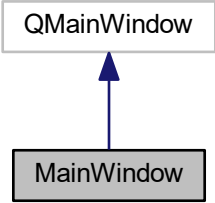
Mainwindow class.

```
#include <mainwindow.h>
```

Inheritance diagram for MainWindow:



Collaboration diagram for MainWindow:



Public Member Functions

- [MainWindow](#) (QWidget *parent=nullptr)
A public constructor.
- [~MainWindow](#) ()
A public destructor.

Private Slots

- void [readSerialPort](#) ()
A private slot for reading serial data.
- void [updatePotVal](#) (QString)
A private slot for updating the lcd display.
- void [updatePotList](#) (QString)
A private slot for updating the data list.
- void [setupRealtimeData](#) ()
A private slot for setting up the graph.
- void [realtimeDataSlot](#) ()
A private slot for updating the graph data.
- void [on_button_user_1_clicked](#) ()
A private slot for marking a button as pressed.
- void [on_button_user_2_clicked](#) ()
A private slot for marking a button as pressed.
- void [on_button_user_3_clicked](#) ()
A private slot for marking a button as pressed.
- void [on_button_user_4_clicked](#) ()
A private slot for marking a button as pressed.
- void [graphStyle](#) ()
A private slot for styling the graphs.
- void [on_button_start_log_clicked](#) ()
A private slot for starting the logging.
- void [on_button_stop_log_clicked](#) ()
A private slot for stopping the logging.
- void [on_button_write_reset_clicked](#) ()
A private slot for stopping the logging.

Private Attributes

- Ui::MainWindow * [ui](#)
A private variable.
- QByteArray [serialData](#)
A private QByteArray variable.
- QString [serialBuffer](#)
A private QString variable.
- QString [parsedData](#)
A private QString variable.
- QString [QStr](#)
A private QString variable.
- QString [tmpQString](#)

- A private QString variable.*

 - std::string `currentUser` = "User not selected yet"

A private string variable.
- QTimer `dataTimer`

A private QTimer variable.
- QSerialPort * `arduino`

A private QSerialPort variable.
- long `documentNumber` = 0

A private long variable.
- double `dataRecieved`

A private variable.
- bool `ok`

A private double variable.
- bool `startLog` = false

A private variable.
- bool `button1Pressed`

A private variable.
- bool `button2Pressed`

A private variable.
- bool `button3Pressed`

A private variable.
- bool `button4Pressed`

A private variable.

Static Private Attributes

- static const quint16 `arduinoVendorId` = 9025

A private static const quint16 variable.
- static const quint16 `arduinoProductId` = 66

A private static const quint16 variable.

5.1.1 Detailed Description

Mainwindow class.

This is the class that provides the main application window.

Definition at line 36 of file mainwindow.h.

5.1.2 Constructor & Destructor Documentation

5.1.2.1 MainWindow()

```
MainWindow::MainWindow (
    QWidget * parent = nullptr ) [explicit]
```

A public constructor.

This is the constructor for the [MainWindow](#) Class.

5.1.2.2 ~MainWindow()

```
MainWindow::~MainWindow ( )
```

A public destructor.

This is the destructor for the [MainWindow](#) Class.

5.1.3 Member Function Documentation

5.1.3.1 graphStyle

```
void MainWindow::graphStyle ( ) [private], [slot]
```

A private slot for styling the graphs.

This slot sets the style of the graphs by setting color, background color etc.

5.1.3.2 on_button_start_log_clicked

```
void MainWindow::on_button_start_log_clicked ( ) [private], [slot]
```

A private slot for starting the logging.

This slot starts the logging of data, by setting a boolean value to "true".

5.1.3.3 on_button_stop_log_clicked

```
void MainWindow::on_button_stop_log_clicked ( ) [private], [slot]
```

A private slot for stopping the logging.

This slot stops the logging by setting a boolean value to "false".

5.1.3.4 on_button_user_1_clicked

```
void MainWindow::on_button_user_1_clicked ( ) [private], [slot]
```

A private slot for marking a button as pressed.

This slot sets user 1 as the selected user, and deselects the other user that was selected. user buttons. This function also sets the currentUser string to "User 1"

5.1.3.5 on_button_user_2_clicked

```
void MainWindow::on_button_user_2_clicked ( ) [private], [slot]
```

A private slot for marking a button as pressed.

This slot sets user 2 as the selected user, and deselects the other user that was selected. user buttons. This function also sets the currentUser string to "User 2"

5.1.3.6 on_button_user_3_clicked

```
void MainWindow::on_button_user_3_clicked ( ) [private], [slot]
```

A private slot for marking a button as pressed.

This slot sets user 3 as the selected user, and deselects the other user that was selected. user buttons. This function also sets the currentUser string to "User 3"

5.1.3.7 on_button_user_4_clicked

```
void MainWindow::on_button_user_4_clicked ( ) [private], [slot]
```

A private slot for marking a button as pressed.

This slot sets user 4 as the selected user, and deselects the other user that was selected. user buttons. This function also sets the currentUser string to "User 4"

5.1.3.8 on_button_write_reset_clicked

```
void MainWindow::on_button_write_reset_clicked ( ) [private], [slot]
```

A private slot for stopping the logging.

This slot stops the logging by setting a boolean value to "false".

5.1.3.9 readSerialPort

```
void MainWindow::readSerialPort ( ) [private], [slot]
```

A private slot for reading serial data.

Sending serial data is not always reliable and data can be split. This function takes the serial data and puts in if the correct sign recieved.

5.1.3.10 realtimeDataSlot

```
void MainWindow::realtimeDataSlot ( ) [private], [slot]
```

A private slot for updating the graph data.

This function updates the

Parameters

<i>QString</i>	String read from readSerial();
----------------	--------------------------------

5.1.3.11 setupRealtimeData

```
void MainWindow::setupRealtimeData ( ) [private], [slot]
```

A private slot for setting up the graph.

This function was used to test the library QCustomGraph. It is not used.

5.1.3.12 updatePotList

```
void MainWindow::updatePotList (
    QString ) [private], [slot]
```

A private slot for updating the data list.

This function updates the list widget with the paramaterasdasd

Parameters

<i>QString</i>	String read from readSerial();
----------------	--------------------------------

5.1.3.13 updatePotVal

```
void MainWindow::updatePotVal (
    QString ) [private], [slot]
```

A private slot for updating the lcd display.

This function updates the lcd widget with the paramater

Parameters

<i>QString</i>	String read from readSerial();
----------------	--------------------------------

5.1.4 Member Data Documentation

5.1.4.1 arduino

```
QSerialPort* MainWindow::arduino [private]
```

A private QSerialPort variable.

This is the variable used to establish serial communication.

Definition at line 223 of file mainwindow.h.

5.1.4.2 arduinoProductId

```
const quint16 MainWindow::arduinoProductId = 66 [static], [private]
```

A private static const quint16 variable.

This variable is used to identify the arduino product ID for serial communication.

Definition at line 237 of file mainwindow.h.

5.1.4.3 arduinoVendorId

```
const quint16 MainWindow::arduinoVendorId = 9025 [static], [private]
```

A private static const quint16 variable.

This variable is used to identify the arduino vendor for serial communication.

Definition at line 230 of file mainwindow.h.

5.1.4.4 button1Pressed

```
bool MainWindow::button1Pressed [private]
```

A private variable.

Details.

Definition at line 272 of file mainwindow.h.

5.1.4.5 `button2Pressed`

```
bool MainWindow::button2Pressed [private]
```

A private variable.

Details.

Definition at line 278 of file `mainwindow.h`.

5.1.4.6 `button3Pressed`

```
bool MainWindow::button3Pressed [private]
```

A private variable.

Details.

Definition at line 284 of file `mainwindow.h`.

5.1.4.7 `button4Pressed`

```
bool MainWindow::button4Pressed [private]
```

A private variable.

Details.

Definition at line 290 of file `mainwindow.h`.

5.1.4.8 `currentUser`

```
std::string MainWindow::currentUser = "User not selected yet" [private]
```

A private string variable.

This is the string used to store the current user, and its initially set to no one.

Definition at line 209 of file `mainwindow.h`.

5.1.4.9 dataRecieved

```
double MainWindow::dataRecieved [private]
```

A private variable.

Details.

Definition at line 253 of file mainwindow.h.

5.1.4.10 dataTimer

```
QTimer MainWindow::dataTimer [private]
```

A private QTimer variable.

This is the timer used in deciding how often to replot the graphs.

Definition at line 216 of file mainwindow.h.

5.1.4.11 documentNumber

```
long MainWindow::documentNumber = 0 [private]
```

A private long variable.

This variable is incremented each time a file is logged. The files are logged with this long in the file name. Example: "test0.txt" and when the new logfile is saved and this variable is now "1" the next file is saved as "test1.txt"

Definition at line 247 of file mainwindow.h.

5.1.4.12 ok

```
bool MainWindow::ok [private]
```

A private double variable.

This variable is the numerical value that gets added to the graphs.

Definition at line 260 of file mainwindow.h.

5.1.4.13 parsedData

```
QString MainWindow::parsedData [private]
```

A private QString variable.

This variable is used to store the second buffer item, and be further used by the widgets.

Definition at line 185 of file mainwindow.h.

5.1.4.14 QStr

```
QString MainWindow::QStr [private]
```

A private QString variable.

This variable is used to store the message as it is converted to the appropriate type.

Definition at line 193 of file mainwindow.h.

5.1.4.15 serialBuffer

```
QString MainWindow::serialBuffer [private]
```

A private QString variable.

This variable is used to store the incoming data in a buffer as to guarantee that the whole message is received. See the mainwindow.cpp file for more information.

Definition at line 177 of file mainwindow.h.

5.1.4.16 serialData

```
QByteArray MainWindow::serialData [private]
```

A private QByteArray variable.

This is the bytearray used to store received bytes from the arduino.

Definition at line 167 of file mainwindow.h.

5.1.4.17 startLog

```
bool MainWindow::startLog = false [private]
```

A private variable.

Details.

Definition at line 266 of file mainwindow.h.

5.1.4.18 tmpQString

```
QString MainWindow::tmpQString [private]
```

A private QString variable.

This variable is used to store the message as it is converted to the appropriate type.

Definition at line 201 of file mainwindow.h.

5.1.4.19 ui

```
Ui::MainWindow* MainWindow::ui [private]
```

A private variable.

This is the mainwindow object used to access different widgets to the UI.

Definition at line 160 of file mainwindow.h.

The documentation for this class was generated from the following file:

- C:/Users/magnu/Desktop/QtDoxygen/source code/mainwindow.h

32 Appendix E - Final software doxygen documentation

32.1 Introduction

This appendix contains the doxygen documentation for the Potentiometer reader GUI as of 26.03.2019.

Contactless Position Sensor - Final software report - Bachelor group 3

Generated by Doxygen 1.8.15

1 Hierarchical Index	1
1.1 Class Hierarchy	1
2 Class Index	3
2.1 Class List	3
3 File Index	5
3.1 File List	5
4 Class Documentation	7
4.1 arduino_control_final Class Reference	7
4.1.1 Detailed Description	9
4.1.2 Member Function Documentation	9
4.1.2.1 aLowerByte()	10
4.1.2.2 aUpperByte()	10
4.1.2.3 bLowerByte()	11
4.1.2.4 buildString()	11
4.1.2.5 bUpperByte()	12
4.1.2.6 clearCounter()	12
4.1.2.7 loop()	13
4.1.2.8 moveStep()	13
4.1.2.9 printBytes()	14
4.1.2.10 saveBytes()	14
4.1.2.11 setup()	15
4.1.2.12 snapShotRegister()	15
4.1.3 Member Data Documentation	16
4.1.3.1 a1	16
4.1.3.2 a2	17
4.1.3.3 a3	17
4.1.3.4 a4	17
4.1.3.5 c1	17
4.1.3.6 c2	18
4.1.3.7 c3	18
4.1.3.8 c4	18
4.1.3.9 calibrating	18
4.1.3.10 CCLR	19
4.1.3.11 clk	19
4.1.3.12 currentValueA	19
4.1.3.13 currentValueC	19
4.1.3.14 currentValueL	20
4.1.3.15 dir	20
4.1.3.16 enb	20
4.1.3.17 GAL	20

4.1.3.18 GAU	21
4.1.3.19 GBL	21
4.1.3.20 GBU	21
4.1.3.21 I1	21
4.1.3.22 I2	22
4.1.3.23 I3	22
4.1.3.24 I4	22
4.1.3.25 message	22
4.1.3.26 para	23
4.1.3.27 RCLK	23
4.1.3.28 runStepCount	23
4.1.3.29 size	23
4.1.3.30 varSpd	24
4.2 cps_ts_gui Class Reference	24
4.2.1 Detailed Description	30
4.2.2 Constructor & Destructor Documentation	30
4.2.2.1 cps_ts_gui()	30
4.2.3 Member Function Documentation	30
4.2.3.1 dataRealtime()	30
4.2.3.2 getArduinoInfo()	30
4.2.3.3 graphStyle()	31
4.2.3.4 on_calibrateButton_clicked	31
4.2.3.5 on_componentInput_textChanged	31
4.2.3.6 on_currentComboBox_currentIndexChanged	31
4.2.3.7 on_directionClockwiseButton_clicked	32
4.2.3.8 on_directionCounterClockwiseButton_clicked	32
4.2.3.9 on_distanceComboBox_currentIndexChanged	32
4.2.3.10 on_distanceSlider_valueChanged	32
4.2.3.11 on_distanceSpinBox_valueChanged	33
4.2.3.12 on_gearRatioInput_textChanged	33
4.2.3.13 on_generatePdfButton_clicked	33
4.2.3.14 on_listView_doubleClicked	34
4.2.3.15 on_motorEnableButton_clicked	34
4.2.3.16 on_notesInput_textChanged	34
4.2.3.17 on_projectInput_textChanged	34
4.2.3.18 on_reconnectButton_clicked	35
4.2.3.19 on_runButton_clicked	35
4.2.3.20 on_speedComboBox_currentIndexChanged	35
4.2.3.21 on_speedSlider_valueChanged	35
4.2.3.22 on_speedSpinBox_valueChanged	36
4.2.3.23 on_startButton_clicked	36
4.2.3.24 on_stepmodeComboBox_currentIndexChanged	36

4.2.3.25 on_testInput_textChanged	37
4.2.3.26 on_treeView_clicked	37
4.2.3.27 on_userInput_textChanged	37
4.2.3.28 on_writeLogButton_clicked	38
4.2.3.29 readSerial	38
4.2.3.30 realtimeDataSlot	38
4.2.3.31 updateCoils	38
4.2.3.32 updateExpectedValue()	39
4.2.3.33 writeSerial	39
4.2.4 Member Data Documentation	39
4.2.4.1 arduino	39
4.2.4.2 arduinoProductId	40
4.2.4.3 arduinoVendorId	40
4.2.4.4 calib	40
4.2.4.5 calibList1	41
4.2.4.6 calibrateListAAAAAAAA	41
4.2.4.7 calibrateListBBBBBBBB	41
4.2.4.8 calibrateListCCCCCCC	41
4.2.4.9 componentID	42
4.2.4.10 currentStep	42
4.2.4.11 currentValue	42
4.2.4.12 dataCoilA	42
4.2.4.13 dataCoilB	43
4.2.4.14 dataCoilC	43
4.2.4.15 dataList	43
4.2.4.16 dataTimer	43
4.2.4.17 directionValue	44
4.2.4.18 dirModel	44
4.2.4.19 distanceMode	44
4.2.4.20 distanceValue	44
4.2.4.21 fileModel	45
4.2.4.22 gearRatio	45
4.2.4.23 motorEnable	45
4.2.4.24 notesInput	45
4.2.4.25 parsedData	46
4.2.4.26 projectName	46
4.2.4.27 serialBuffer	46
4.2.4.28 serialData	46
4.2.4.29 speedMode	47
4.2.4.30 speedValue	47
4.2.4.31 stepMode	47
4.2.4.32 testID	47

4.2.4.33 ui	48
4.2.4.34 userName	48
4.2.4.35 writePDF	48
4.3 fileGenerate Class Reference	48
4.3.1 Detailed Description	49
4.3.2 Member Function Documentation	49
4.3.2.1 fileStruct()	49
4.3.2.2 getCalData()	51
4.3.2.3 lastPos()	51
4.4 pdfGenerate Class Reference	52
4.4.1 Detailed Description	52
4.4.2 Member Function Documentation	53
4.4.2.1 fileStruct()	53
4.5 writeArduino Class Reference	54
4.5.1 Detailed Description	55
4.5.2 Member Function Documentation	55
4.5.2.1 calculateDistance()	55
4.5.2.2 calculateSpeed()	55
4.5.2.3 wArd()	56
5 File Documentation	57
5.1 C:/Users/magnu/Desktop/Doxygen/Doxygen source/cps_ts_gui_21_05_2019 - Copy/arduino_↔ control_final.c File Reference	57
5.1.1 Macro Definition Documentation	57
5.1.1.1 ARDUINO_CONTROL_FINAL_H	57
5.1.1.2 BAUD	57
5.1.1.3 DELAY_TIME	58
5.1.1.4 INPUT_SIZE	58
5.2 C:/Users/magnu/Desktop/Doxygen/Doxygen source/cps_ts_gui_21_05_2019 - Copy/cps_ts_gui.h File Reference	58
5.3 C:/Users/magnu/Desktop/Doxygen/Doxygen source/cps_ts_gui_21_05_2019 - Copy/fileGenerate.h File Reference	59
5.3.1 Macro Definition Documentation	59
5.3.1.1 FILEGENERATE_H	59
5.4 C:/Users/magnu/Desktop/Doxygen/Doxygen source/cps_ts_gui_21_05_2019 - Copy/pdfGenerate.h File Reference	60
5.4.1 Macro Definition Documentation	60
5.4.1.1 PDFGENERATE_H	60
5.5 C:/Users/magnu/Desktop/Doxygen/Doxygen source/cps_ts_gui_21_05_2019 - Copy/writeArduino.h File Reference	61
5.5.1 Macro Definition Documentation	61
5.5.1.1 WRITEARDUINO_H	61

Chapter 1

Hierarchical Index

1.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

arduino_control_final	7
fileGenerate	48
pdfGenerate	52
QDialog	
cps_ts_gui	24
writeArduino	54

Chapter 2

Class Index

2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

arduino_control_final	7
cps_ts_gui A class for the graphical user interface and its content	24
fileGenerate A class for generating files and getting calibration data	48
pdfGenerate A class for generating PDF reports	52
writeArduino A class for calculating parameters to arduino message code	54

Chapter 3

File Index

3.1 File List

Here is a list of all files with brief descriptions:

C:/Users/magnu/Desktop/Doxygen/Doxygen source/cps_ts_gui_21_05_2019 - Copy/[arduino_control_final.c](#)
57

C:/Users/magnu/Desktop/Doxygen/Doxygen source/cps_ts_gui_21_05_2019 - Copy/[cps_ts_gui.h](#) . . . 58

C:/Users/magnu/Desktop/Doxygen/Doxygen source/cps_ts_gui_21_05_2019 - Copy/[fileGenerate.h](#) . . . 59

C:/Users/magnu/Desktop/Doxygen/Doxygen source/cps_ts_gui_21_05_2019 - Copy/[pdfGenerate.h](#) . . . 60

C:/Users/magnu/Desktop/Doxygen/Doxygen source/cps_ts_gui_21_05_2019 - Copy/[writeArduino.h](#) . . . 61

Chapter 4

Class Documentation

4.1 arduino_control_final Class Reference

Collaboration diagram for arduino_control_final:

arduino_control_final
+ para + size + CCLR + GAU + GAL + RCLK + GBL + GBU + enb + dir and 20 more...
+ setup() + loop() + moveStep() + saveBytes() + printBytes() + buildString() + bUpperByte() + bLowerByte() + aUpperByte() + aLowerByte() + clearCounter() + snapShotRegister()

Public Member Functions

- void [setup](#) ()
A public member.
- void [loop](#) ()
A public member.
- void [moveStep](#) (char [message](#)[])
A public member.
- void [saveBytes](#) ()
A public member.
- void [printBytes](#) ()
A public member.
- String [buildString](#) ()
A public member.
- void [bUpperByte](#) ()
A public member.
- void [bLowerByte](#) ()
A public member.
- void [aUpperByte](#) ()
A public member.
- void [aLowerByte](#) ()
A public member.
- void [clearCounter](#) ()
A public member.
- void [snapShotRegister](#) ()
A public member.

Public Attributes

- int [para](#)
A global int variable.
- byte [size](#)
A global byte variable.
- const int [CCLR](#) = 11
A global const int variable.
- const int [GAU](#) = 9
A global const int variable.
- const int [GAL](#) = 8
A global const int variable.
- const int [RCLK](#) = 7
A global const int variable.
- const int [GBL](#) = 5
A global const int variable.
- const int [GBU](#) = 6
A global const int variable.
- const int [enb](#) = 2
A global const int variable.
- const int [dir](#) = 3
A global const int variable.
- const int [clk](#) = 4

- A global const int variable.*

 - volatile long `varSpd` = 50
- A global volatile long variable.*

 - bool `calibrating` = false
- A global bool variable.*

 - unsigned long `currentValueA` = 0
- A global unsigned long variable.*

 - unsigned long `currentValueC` = 0
- A global unsigned long variable.*

 - unsigned long `currentValueL` = 0
- A global unsigned long variable.*

 - long `runStepCount` = 0
- A global long variable.*

 - String `message` = ""
- A global string variable.*

 - byte `a1`
- A global byte variable.*

 - byte `a2`
- A global byte variable.*

 - byte `a3`
- A global byte variable.*

 - byte `a4`
- A global byte variable.*

 - byte `c1`
- A global byte variable.*

 - byte `c2`
- A global byte variable.*

 - byte `c3`
- A global byte variable.*

 - byte `c4`
- A global byte variable.*

 - byte `l1`
- A global byte variable.*

 - byte `l2`
- A global byte variable.*

 - byte `l3`
- A global byte variable.*

 - byte `l4`

4.1.1 Detailed Description

Definition at line 8 of file `arduino_control_final.c`.

4.1.2 Member Function Documentation

4.1.2.1 aLowerByte()

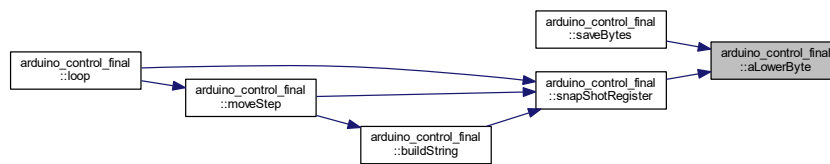
```
void arduino_control_final::aLowerByte ( ) [inline]
```

A public member.

Function to change the outputregister to get the lower bytes of the A counter.

Definition at line 579 of file arduino_control_final.c.

Here is the caller graph for this function:



4.1.2.2 aUpperByte()

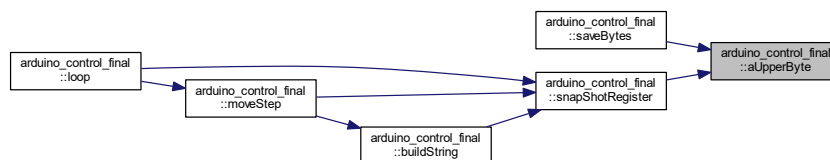
```
void arduino_control_final::aUpperByte ( ) [inline]
```

A public member.

Function to change the outputregister to get the upper bytes of the A counter

Definition at line 563 of file arduino_control_final.c.

Here is the caller graph for this function:



4.1.2.3 bLowerByte()

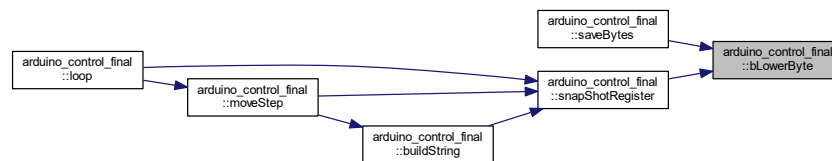
```
void arduino_control_final::bLowerByte ( ) [inline]
```

A public member.

Function to change the outputregister to get the lower bytes of the B counter

Definition at line 547 of file arduino_control_final.c.

Here is the caller graph for this function:



4.1.2.4 buildString()

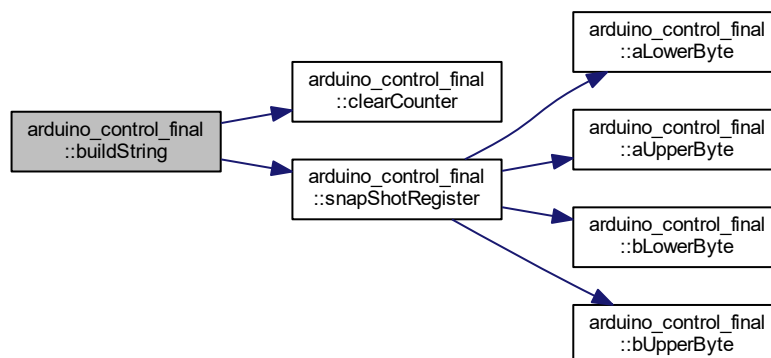
```
String arduino_control_final::buildString ( ) [inline]
```

A public member.

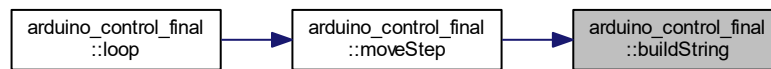
Function to build the message string that will be sent and deconstructed at the user interface. First we make sure that the string is cleared, and then we build the string and send it with 100 sensor readings at the time. The counters are reset and counts for x ammount of time before saving the the values stored.

Definition at line 502 of file arduino_control_final.c.

Here is the call graph for this function:



Here is the caller graph for this function:



4.1.2.5 bUpperByte()

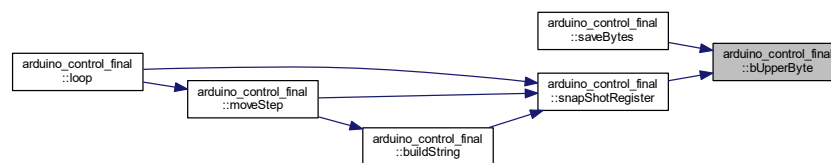
```
void arduino_control_final::bUpperByte ( ) [inline]
```

A public member.

Function to change the outputregister to get the upper bytes of the B counter.

Definition at line 531 of file arduino_control_final.c.

Here is the caller graph for this function:



4.1.2.6 clearCounter()

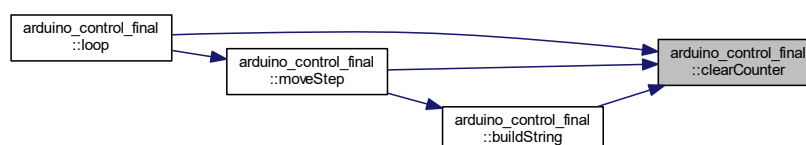
```
void arduino_control_final::clearCounter ( ) [inline]
```

A public member.

Function to clear the counters

Definition at line 595 of file arduino_control_final.c.

Here is the caller graph for this function:



4.1.2.7 loop()

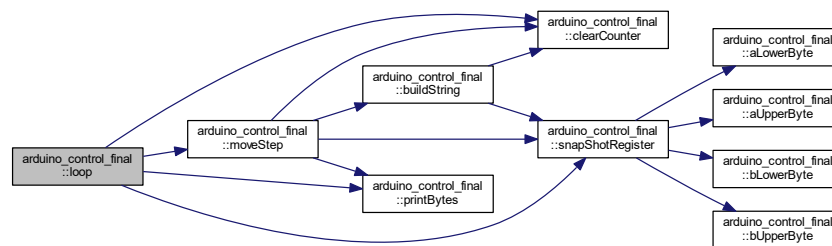
```
void arduino_control_final::loop ( ) [inline]
```

A public member.

This member is the loop. This is the main arduino loop and it does one of 2 things. First it checks if there are incoming messages. If there are the message is deciphered and parsed as a parameter into the function that controls the stepper motor. If however there are no incoming messages, the counters will count and the result will be sent to the software.

Definition at line 279 of file arduino_control_final.c.

Here is the call graph for this function:



4.1.2.8 moveStep()

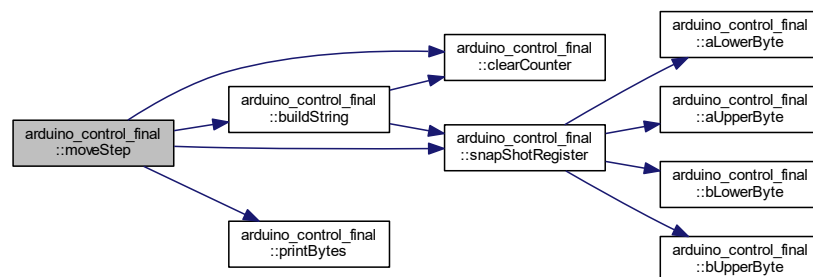
```
void arduino_control_final::moveStep (
    char message[] ) [inline]
```

A public member.

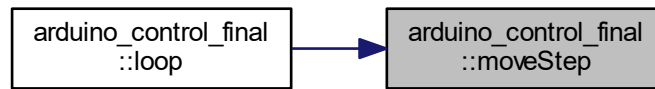
This member cycles through the command string and changes settings and/or drives the motor.

Definition at line 303 of file arduino_control_final.c.

Here is the call graph for this function:



Here is the caller graph for this function:



4.1.2.9 printBytes()

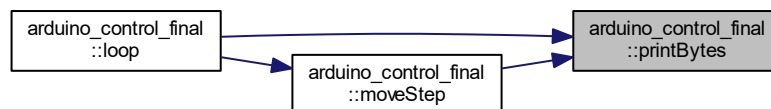
```
void arduino_control_final::printBytes ( ) [inline]
```

A public member.

Function to print the values collected.

Definition at line 469 of file `arduino_control_final.c`.

Here is the caller graph for this function:



4.1.2.10 saveBytes()

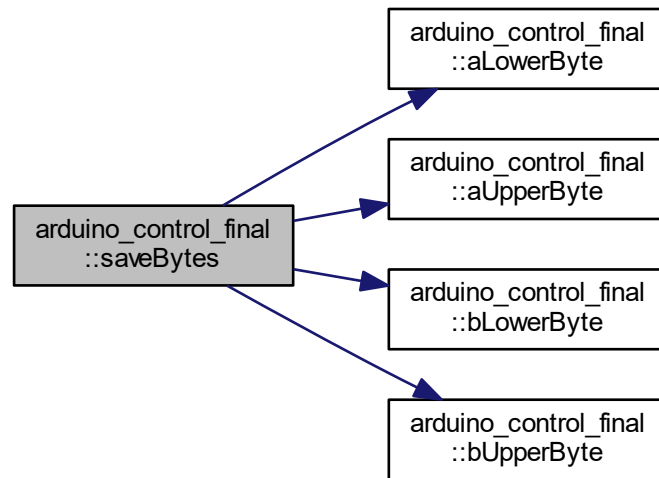
```
void arduino_control_final::saveBytes ( ) [inline]
```

A public member.

Function to get the values from the counters.

Definition at line 430 of file `arduino_control_final.c`.

Here is the call graph for this function:



4.1.2.11 setup()

```
void arduino_control_final::setup ( ) [inline]
```

A public member.

This member is run once the arduino is turned on. It sets the different registers to input registers, sets the different pinmodes, resets the counters, and gives the driver default values.

Definition at line 232 of file `arduino_control_final.c`.

4.1.2.12 snapShotRegister()

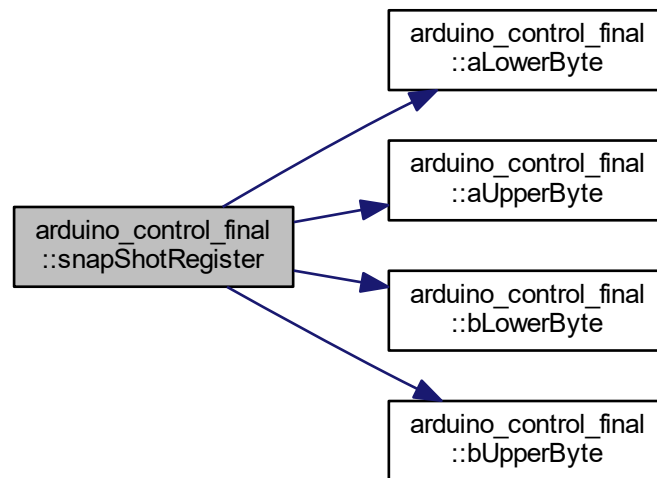
```
void arduino_control_final::snapShotRegister ( ) [inline]
```

A public member.

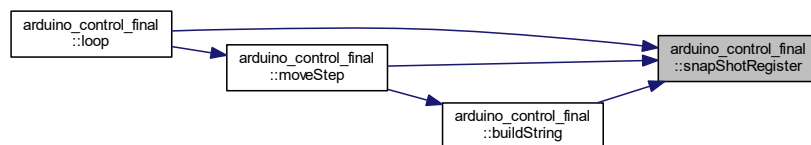
Function to save the content of the counters on the IC internal register

Definition at line 606 of file `arduino_control_final.c`.

Here is the call graph for this function:



Here is the caller graph for this function:



4.1.3 Member Data Documentation

4.1.3.1 a1

```
byte arduino_control_final::a1
```

A global byte variable.

This variable is used to hold the values of the A counter

Definition at line 146 of file `arduino_control_final.c`.

4.1.3.2 a2

```
byte arduino_control_final::a2
```

A global byte variable.

This variable is used to hold the values of the A counter

Definition at line 153 of file arduino_control_final.c.

4.1.3.3 a3

```
byte arduino_control_final::a3
```

A global byte variable.

This variable is used to hold the values of the A counter

Definition at line 160 of file arduino_control_final.c.

4.1.3.4 a4

```
byte arduino_control_final::a4
```

A global byte variable.

This variable is used to hold the values of the A counter

Definition at line 167 of file arduino_control_final.c.

4.1.3.5 c1

```
byte arduino_control_final::c1
```

A global byte variable.

This variable is used to hold the values of the B counter

Definition at line 174 of file arduino_control_final.c.

4.1.3.6 c2

```
byte arduino_control_final::c2
```

A global byte variable.

This variable is used to hold the values of the B counter

Definition at line 181 of file arduino_control_final.c.

4.1.3.7 c3

```
byte arduino_control_final::c3
```

A global byte variable.

This variable is used to hold the values of the B counter

Definition at line 188 of file arduino_control_final.c.

4.1.3.8 c4

```
byte arduino_control_final::c4
```

A global byte variable.

This variable is used to hold the values of the B counter

Definition at line 195 of file arduino_control_final.c.

4.1.3.9 calibrating

```
bool arduino_control_final::calibrating = false
```

A global bool variable.

This variable is used to check if calibrating or not.

Definition at line 101 of file arduino_control_final.c.

4.1.3.10 CCLR

```
const int arduino_control_final::CCLR = 11
```

A global const int variable.

This is the pinmapping for the CCLR pin on the SN74LV8154 counter.

Definition at line 31 of file arduino_control_final.c.

4.1.3.11 clk

```
const int arduino_control_final::clk = 4
```

A global const int variable.

This is the pinmapping for the clk pin on the DM420A driver.

Definition at line 87 of file arduino_control_final.c.

4.1.3.12 currentValueA

```
unsigned long arduino_control_final::currentValueA = 0
```

A global unsigned long variable.

This variable is used to store the value of coil A from arduino register A

Definition at line 109 of file arduino_control_final.c.

4.1.3.13 currentValueC

```
unsigned long arduino_control_final::currentValueC = 0
```

A global unsigned long variable.

This variable is used to store the value of coil B from arduino register C

Definition at line 116 of file arduino_control_final.c.

4.1.3.14 currentValueL

```
unsigned long arduino_control_final::currentValueL = 0
```

A global unsigned long variable.

This variable is used to store the value of coil C from arduino register L

Definition at line 123 of file arduino_control_final.c.

4.1.3.15 dir

```
const int arduino_control_final::dir = 3
```

A global const int variable.

This is the pinmapping for the direction pin on the DM420A driver.

Definition at line 80 of file arduino_control_final.c.

4.1.3.16 enb

```
const int arduino_control_final::enb = 2
```

A global const int variable.

This is the pinmapping for the enb pin on the DM420A driver.

Definition at line 73 of file arduino_control_final.c.

4.1.3.17 GAL

```
const int arduino_control_final::GAL = 8
```

A global const int variable.

This is the pinmapping for the GAL pin on the SN74LV8154 counter.

Definition at line 45 of file arduino_control_final.c.

4.1.3.18 GAU

```
const int arduino_control_final::GAU = 9
```

A global const int variable.

This is the pinmapping for the GAU pin on the SN74LV8154 counter.

Definition at line 38 of file arduino_control_final.c.

4.1.3.19 GBL

```
const int arduino_control_final::GBL = 5
```

A global const int variable.

This is the pinmapping for the GBL pin on the SN74LV8154 counter.

Definition at line 59 of file arduino_control_final.c.

4.1.3.20 GBU

```
const int arduino_control_final::GBU = 6
```

A global const int variable.

This is the pinmapping for the GBU pin on the SN74LV8154 counter.

Definition at line 66 of file arduino_control_final.c.

4.1.3.21 l1

```
byte arduino_control_final::l1
```

A global byte variable.

This variable is used to hold the values of the C counter

Definition at line 202 of file arduino_control_final.c.

4.1.3.22 I2

```
byte arduino_control_final::I2
```

A global byte variable.

This variable is used to hold the values of the C counter

Definition at line 209 of file arduino_control_final.c.

4.1.3.23 I3

```
byte arduino_control_final::I3
```

A global byte variable.

This variable is used to hold the values of the C counter

Definition at line 216 of file arduino_control_final.c.

4.1.3.24 I4

```
byte arduino_control_final::I4
```

A global byte variable.

This variable is used to hold the values of the C counter

Definition at line 223 of file arduino_control_final.c.

4.1.3.25 message

```
String arduino_control_final::message = ""
```

A global string variable.

This variable is the message that gets built and sent to the software. This message contains the values of the coils, and is initialised as empty.

Definition at line 139 of file arduino_control_final.c.

4.1.3.26 para

```
int arduino_control_final::para
```

A global int variable.

This is the parameterset, containing both parameter type and its value.

Definition at line 17 of file arduino_control_final.c.

4.1.3.27 RCLK

```
const int arduino_control_final::RCLK = 7
```

A global const int variable.

This is the pinmapping for the RCLK pin on the SN74LV8154 counter.

Definition at line 52 of file arduino_control_final.c.

4.1.3.28 runStepCount

```
long arduino_control_final::runStepCount = 0
```

A global long variable.

This variable is used to store number of steps stepped during the continuously run mode.

Definition at line 130 of file arduino_control_final.c.

4.1.3.29 size

```
byte arduino_control_final::size
```

A global byte variable.

This is size for the command message array.

Definition at line 24 of file arduino_control_final.c.

4.1.3.30 varSpd

```
volatile long arduino_control_final::varSpd = 50
```

A global volatile long variable.

This is the default speedvalue for the stepper motor.

Definition at line 94 of file `arduino_control_final.c`.

The documentation for this class was generated from the following file:

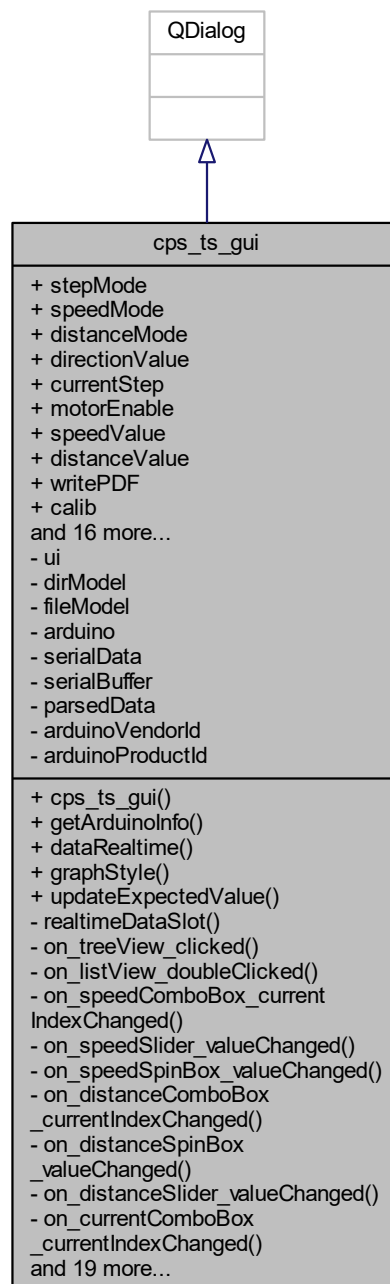
- `C:/Users/magnu/Desktop/Doxygen/Doxygen source/cps_ts_gui_21_05_2019 - Copy/arduino_control_final.c`

4.2 cps_ts_gui Class Reference

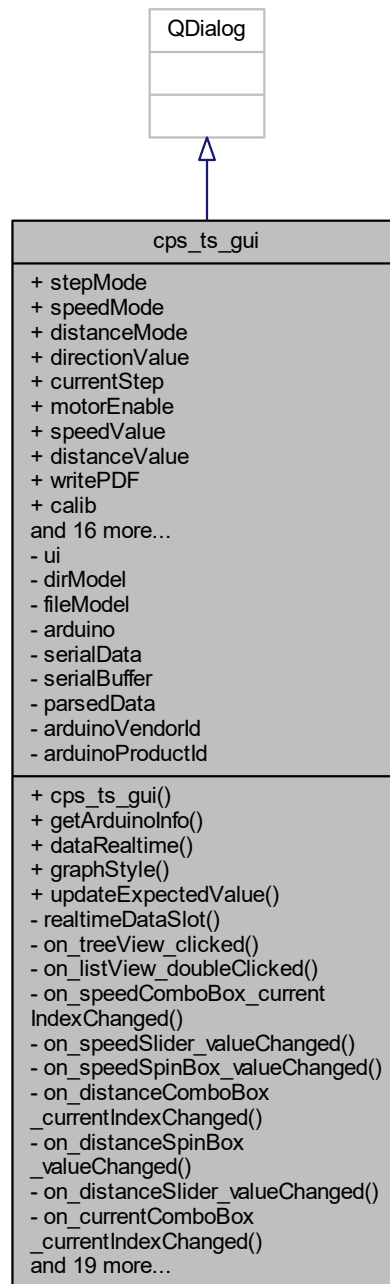
A class for the graphical user interface and its content.

```
#include <cps_ts_gui.h>
```

Inheritance diagram for cps_ts_gui:



Collaboration diagram for `cps_ts_gui`:



Public Member Functions

- `cps_ts_gui` (`QWidget *parent=Q_NULLPTR`)
A public constructor.
- void `getArduinoInfo` ()
A public member.
- void `dataRealtime` ()

- A public member.*
- void `graphStyle` ()
A public member.
- void `updateExpectedValue` (QString arg1, QString arg2, QString arg3)
A public member.

Public Attributes

- `std::string stepMode` = "128"
A public string variable.
- `std::string speedMode` = "step"
A public string variable.
- `std::string distanceMode` = "step"
A public string variable.
- `int directionValue` = 0
A public int variable.
- `int currentStep` = 0
A public int variable.
- `short motorEnable` = 0
A public short variable.
- `long speedValue` = 1
A public long variable.
- `long distanceValue` = 1
A public long variable.
- `bool writePDF` = true
A public boolean variable.
- `bool calib` = false
A public boolean variable.
- `QTimer dataTimer`
A public QTimer variable.
- `QString currentValue` = "0.31 RMS Current"
A public QString variable.
- `QString gearRatio` = "Gear ratio not selected"
A public QString variable.
- `QString projectName` = "Project name not selected"
A public QString variable.
- `QString userName` = "UserName not selected"
A public QString variable.
- `QString componentID` = "Component ID not selected"
A public QString variable.
- `QString testID` = "Test ID not selected"
A public QString variable.
- `QString notesInput` = "Notes not selected"
A public QString variable.
- `QString dataCoilA`
A public QString variable.
- `QString dataCoilB`
A public QString variable.
- `QString dataCoilC`
A public QString variable.

- [QStringList dataList](#)
A public `QStringList` variable.
- [QStringList calibList1](#)
A public `QStringList` variable.
- long [calibrateListAAAAAAAA](#) [51200]
A public `long[]` variable.
- long [calibrateListBBBBBBBB](#) [51200]
A public `long[]` variable.
- long [calibrateListCCCCCCCC](#) [51200]
A public `long[]` variable.

Private Slots

- void [realtimeDataSlot](#) ()
A private slot.
- void [on_treeView_clicked](#) (const QModelIndex &index)
A private slot.
- void [on_listView_doubleClicked](#) (const QModelIndex &index)
A private slot.
- void [on_speedComboBox_currentIndexChanged](#) (int value)
A private slot.
- void [on_speedSlider_valueChanged](#) (int value)
A private slot.
- void [on_speedSpinBox_valueChanged](#) (double arg1)
A private slot.
- void [on_distanceComboBox_currentIndexChanged](#) (int value)
A private slot.
- void [on_distanceSpinBox_valueChanged](#) (double arg1)
A private slot.
- void [on_distanceSlider_valueChanged](#) (int value)
A private slot.
- void [on_currentComboBox_currentIndexChanged](#) (int value)
A private slot.
- void [on_stepmodeComboBox_currentIndexChanged](#) (int value)
A private slot.
- void [on_projectInput_textChanged](#) (const QString &arg1)
A private slot.
- void [on_componentInput_textChanged](#) (const QString &arg1)
A private slot.
- void [on_testInput_textChanged](#) (const QString &arg1)
A private slot.
- void [on_userInput_textChanged](#) (const QString &arg1)
A private slot.
- void [on_notesInput_textChanged](#) ()
A private slot.
- void [on_gearRatioInput_textChanged](#) (const QString &arg1)
A private slot.
- void [on_directionCounterClockwiseButton_clicked](#) ()
A private slot.
- void [on_directionClockwiseButton_clicked](#) ()

- A private slot.*

 - void [on_writeLogButton_clicked](#) ()

A private slot.

 - void [on_calibrateButton_clicked](#) ()

A private slot.

 - void [on_generatePdfButton_clicked](#) ()

A private slot.

 - void [on_startButton_clicked](#) ()

A private slot.

 - void [on_motorEnableButton_clicked](#) ()

A private slot.

 - void [on_runButton_clicked](#) ()

A private slot.

 - void [on_reconnectButton_clicked](#) ()

A private slot.

 - void [readSerial](#) ()

A private slot.

 - void [updateCoils](#) (QString sensor_reading_1, QString sensor_reading_2, QString sensor_reading_3)

A private slot.

 - void [writeSerial](#) (const QByteArray &data)

A private slot.

Private Attributes

- Ui::cps_ts_guiClass [ui](#)

A private variable.
- QFileSystemModel * [dirModel](#)

A private variable.
- QFileSystemModel * [fileModel](#)

A private variable.
- QSerialPort * [arduino](#)

A private variable.
- QByteArray [serialData](#)

A private variable.
- QString [serialBuffer](#)

A private QString variable.
- QString [parsedData](#)

A private QString variable.

Static Private Attributes

- static const quint16 [arduinoVendorId](#) = 9025

A private const quint16 variable.
- static const quint16 [arduinoProductId](#) = 66

A private const quint16 variable.

4.2.1 Detailed Description

A class for the graphical user interface and its content.

This class is the base class of the project. It contains the controlsystem and the graphical user interface.

Definition at line 31 of file `cps_ts_gui.h`.

4.2.2 Constructor & Destructor Documentation

4.2.2.1 `cps_ts_gui()`

```
cps_ts_gui::cps_ts_gui (
    QWidget * parent = Q_NULLPTR )
```

A public constructor.

This is the constructor for the [cps_ts_gui](#).

4.2.3 Member Function Documentation

4.2.3.1 `dataRealtime()`

```
void cps_ts_gui::dataRealtime ( )
```

A public member.

This member sets up the graphs from the QCustomPlot library and connects the signal emitted from the

See also

[dataTimer](#), with the
[realtimeDataSlot\(\)](#) member.

4.2.3.2 `getArduinoInfo()`

```
void cps_ts_gui::getArduinoInfo ( )
```

A public member.

This member prints the product information about the devices present on the serialport, in order to setup automatic connection used in the constructor.

4.2.3.3 graphStyle()

```
void cps_ts_gui::graphStyle ( )
```

A public member.

This member styles the graphs setup in order to make them fit the theme of the GUI.

4.2.3.4 on_calibrateButton_clicked

```
void cps_ts_gui::on_calibrateButton_clicked ( ) [private], [slot]
```

A private slot.

This slot is used to start the calibration-mode. First the different variables are cleared and the calibration command is sent to the arduino. Then a continuously loop is started. First a number of samples is sent from the arduino and then deciphered and saved in the appropriate calibration arrays. This is repeated until the calibration is done.

4.2.3.5 on_componentInput_textChanged

```
void cps_ts_gui::on_componentInput_textChanged (
    const QString & arg1 ) [private], [slot]
```

A private slot.

This slot stores the component name the user inputs.

Parameters

<i>arg1</i>	A QString pointer variable that holds the the text the user enters
-------------	--------------------------------------------------------------------

4.2.3.6 on_currentComboBox_currentIndexChanged

```
void cps_ts_gui::on_currentComboBox_currentIndexChanged (
    int value ) [private], [slot]
```

A private slot.

This slot changes the

See also

[currentValue](#) variable accordingly.

Parameters

<i>value</i>	A int variable that holds the index value of the combobox.
--------------	------------------------------------------------------------

4.2.3.7 on_directionClockwiseButton_clicked

```
void cps_ts_gui::on_directionClockwiseButton_clicked ( ) [private], [slot]
```

A private slot.

This slot decides the direction of rotation on the stepper motor. If the opposite direction button is selected, it is deselected.

4.2.3.8 on_directionCounterClockwiseButton_clicked

```
void cps_ts_gui::on_directionCounterClockwiseButton_clicked ( ) [private], [slot]
```

A private slot.

This slot decides the direction of rotation on the stepper motor. If the opposite direction button is selected, it is deselected.

4.2.3.9 on_distanceComboBox_currentIndexChanged

```
void cps_ts_gui::on_distanceComboBox_currentIndexChanged (
    int value ) [private], [slot]
```

A private slot.

This slot changes the

See also

[distanceMode](#) variable and changes the max and min value of the slider and spinbox. This combobox changes the distance parameters from degrees to steps and vice versa.

Parameters

<i>value</i>	A int variable that holds the index value of the combobox.
--------------	------------------------------------------------------------

4.2.3.10 on_distanceSlider_valueChanged

```
void cps_ts_gui::on_distanceSlider_valueChanged (
    int value ) [private], [slot]
```

A private slot.

This slot changes the `\distanceValue` accordingly.

Parameters

<i>value</i>	A int variable that holds the value of the slider
--------------	---------------------------------------------------

4.2.3.11 on_distanceSpinBox_valueChanged

```
void cps_ts_gui::on_distanceSpinBox_valueChanged (
    double arg1 ) [private], [slot]
```

A private slot.

This slot changes the

See also

[speedValue](#) accordingly.

Parameters

<i>arg1</i>	A float variable that holds the value of the spinbox.
-------------	-------------------------------------------------------

4.2.3.12 on_gearRatioInput_textChanged

```
void cps_ts_gui::on_gearRatioInput_textChanged (
    const QString & arg1 ) [private], [slot]
```

A private slot.

This slot stores the gear ratio the user inputs.

Parameters

<i>arg1</i>	A QString pointer variable that holds the the text the user enters
-------------	--------------------------------------------------------------------

4.2.3.13 on_generatePdfButton_clicked

```
void cps_ts_gui::on_generatePdfButton_clicked ( ) [private], [slot]
```

A private slot.

This slot selects if a PDF version of the log is to be created as well as an XML version.

4.2.3.14 on_listView_doubleClicked

```
void cps_ts_gui::on_listView_doubleClicked (
    const QModelIndex & index ) [private], [slot]
```

A private slot.

This slot extracts the root path from the double-clicked item in the list view, and calls the open-function from the QDesktopServices library and opens the selected file.

Parameters

<i>index</i>	A QModelIndex pointer to the index of the file object
--------------	-------------------------------------------------------

4.2.3.15 on_motorEnableButton_clicked

```
void cps_ts_gui::on_motorEnableButton_clicked ( ) [private], [slot]
```

A private slot.

This slot is used to enable and disable the stepper motor.

4.2.3.16 on_notesInput_textChanged

```
void cps_ts_gui::on_notesInput_textChanged ( ) [private], [slot]
```

A private slot.

This slot stores the notes the user inputs.

4.2.3.17 on_projectInput_textChanged

```
void cps_ts_gui::on_projectInput_textChanged (
    const QString & arg1 ) [private], [slot]
```

A private slot.

This slot stores the project name the user inputs.

Parameters

<i>arg1</i>	A QString pointer variable that holds the the text the user enters
-------------	--------------------------------------------------------------------

4.2.3.18 on_reconnectButton_clicked

```
void cps_ts_gui::on_reconnectButton_clicked ( ) [private], [slot]
```

A private slot.

This slot is used to reconnect the arduino. It closes the connection to the object created and recreates the serial object. Then the com ports are searched for the correct product and vendor ID, and if found it connects with appropriate settings.

4.2.3.19 on_runButton_clicked

```
void cps_ts_gui::on_runButton_clicked ( ) [private], [slot]
```

A private slot.

This slot writes the command for continuous rotation on the stepper motor.

4.2.3.20 on_speedComboBox_currentIndexChanged

```
void cps_ts_gui::on_speedComboBox_currentIndexChanged (
    int value ) [private], [slot]
```

A private slot.

This slot changes the

See also

[speedMode](#) variable and changes the max and min value of the slider and spinbox to suit the appropriate values. This combobox changes the speed parameters from either degrees per seconds to steps per seconds.

Parameters

<i>value</i>	A int variable that holds the value of the spinbox
--------------	----------------------------------------------------

4.2.3.21 on_speedSlider_valueChanged

```
void cps_ts_gui::on_speedSlider_valueChanged (
    int value ) [private], [slot]
```

A private slot.

This slot changes the

See also

[speedValue](#) accordingly.

Parameters

<i>value</i>	A int variable that holds the value of the slider
--------------	---------------------------------------------------

4.2.3.22 on_speedSpinBox_valueChanged

```
void cps_ts_gui::on_speedSpinBox_valueChanged (  
    double arg1 ) [private], [slot]
```

A private slot.

This slot changes the

See also

[speedValue](#) accordingly.

Parameters

<i>arg1</i>	A double variable to hold the content of the spinbox
-------------	------------------------------------------------------

4.2.3.23 on_startButton_clicked

```
void cps_ts_gui::on_startButton_clicked ( ) [private], [slot]
```

A private slot.

This slot builds a command message from the combination of parameters set by the user. This command is then sent to the arduino for the execution of the command.

4.2.3.24 on_stepmodeComboBox_currentIndexChanged

```
void cps_ts_gui::on_stepmodeComboBox_currentIndexChanged (  
    int value ) [private], [slot]
```

A private slot.

This slot changes the

See also

[stepMode](#) variable accordingly.

Parameters

<i>value</i>	A int variable that holds the index value of the combobox.
--------------	------------------------------------------------------------

4.2.3.25 on_testInput_textChanged

```
void cps_ts_gui::on_testInput_textChanged (
    const QString & arg1 ) [private], [slot]
```

A private slot.

This slot stores the test name the user inputs.

Parameters

<i>arg1</i>	A QString pointer variable that holds the the text the user enters
-------------	--------------------------------------------------------------------

4.2.3.26 on_treeView_clicked

```
void cps_ts_gui::on_treeView_clicked (
    const QModelIndex & index ) [private], [slot]
```

A private slot.

This slot takes the active element's root path and set the new root path to this one. This way the tree is expanded one level.

Parameters

<i>index</i>	A QModelIndex pointer to the index of the directory object
--------------	------------------------------------------------------------

4.2.3.27 on_userInput_textChanged

```
void cps_ts_gui::on_userInput_textChanged (
    const QString & arg1 ) [private], [slot]
```

A private slot.

This slot stores the user name the user inputs.

Parameters

<i>arg1</i>	A QString pointer variable that holds the the text the user enters
-------------	--------------------------------------------------------------------

4.2.3.28 on_writeLogButton_clicked

```
void cps_ts_gui::on_writeLogButton_clicked ( ) [private], [slot]
```

A private slot.

This slot takes all available data such as parameters, project info, current CPS data, expected CPS data and creates a XML document from it. If the PDF bool is true, a PDF version is also created.

4.2.3.29 readSerial

```
void cps_ts_gui::readSerial ( ) [private], [slot]
```

A private slot.

This slot checks if the program is in calibration mode and then reads serial data from the serial port. If the CPS is calibrating, it does nothing but if not it continues to aquire data. The data is split by an escape sign and the software aquires bytes continously. By placing the incomming bytes in a buffer and splitting it by the escape sign, we can know when the full message has been recived and then further process the data.

4.2.3.30 realtimeDataSlot

```
void cps_ts_gui::realtimeDataSlot ( ) [private], [slot]
```

A private slot.

This slot adds new values to the graph plot and then replots every time it's called.

4.2.3.31 updateCoils

```
void cps_ts_gui::updateCoils (
    QString sensor_reading_1,
    QString sensor_reading_2,
    QString sensor_reading_3 ) [private], [slot]
```

A private slot.

This slot updates the value displayed on the lcdNumber widgets with the current parameters.

Parameters

<i>sensor_reading_1</i>	A QString variable that holds the current data on coil A
<i>sensor_reading_2</i>	A QString variable that holds the current data on coil B
<i>sensor_reading_3</i>	A QString variable that holds the current data on coil C

4.2.3.32 updateExpectedValue()

```
void cps_ts_gui::updateExpectedValue (
    QString arg1,
    QString arg2,
    QString arg3 )
```

A public member.

This member updates the expected coil values (given as parameters) and displays them on their respective "expected" location on the gui.

Parameters

<i>arg1</i>	A QString parameter for coil A data
<i>arg2</i>	A QString parameter for coil B data
<i>arg3</i>	A QString parameter for coil C data

4.2.3.33 writeSerial

```
void cps_ts_gui::writeSerial (
    const QByteArray & data ) [private], [slot]
```

A private slot.

Clear the serialBuffer and parsedData variable and write To the arduino with the given bytearray.

Parameters

<i>data</i>	A QByteArray pointer that holds the command to be sent
-------------	--------------------------------------------------------

4.2.4 Member Data Documentation

4.2.4.1 arduino

```
QSerialPort* cps_ts_gui::arduino [private]
```

A private variable.

This is a pointer object. This is the object that is used in the serial communication

Definition at line 567 of file cps_ts_gui.h.

4.2.4.2 `arduinoProductId`

```
const quint16 cps_ts_gui::arduinoProductId = 66 [static], [private]
```

A private const quint16 variable.

Arduino Mega: This variable is used to identify the product on the serial coms and if it matches this product ID and the vendor ID

See also

`arduino_uno_vendor_id` it establishes a connection.

Definition at line 610 of file `cps_ts_gui.h`.

4.2.4.3 `arduinoVendorId`

```
const quint16 cps_ts_gui::arduinoVendorId = 9025 [static], [private]
```

A private const quint16 variable.

Arduino Mega: This variable is used to identify the product on the serial coms and if it matches this vendor ID and the product ID

See also

`arduino_uno_vendor_id` it establishes a connection.

Definition at line 601 of file `cps_ts_gui.h`.

4.2.4.4 `calib`

```
bool cps_ts_gui::calib = false
```

A public boolean variable.

This variable is used to start and stop the calibration sequence. It is initialised to false.

Definition at line 119 of file `cps_ts_gui.h`.

4.2.4.5 calibList1

```
QStringList cps_ts_gui::calibList1
```

A public QStringList variable.

This variable is used to store the incoming calibration data.

Definition at line 212 of file cps_ts_gui.h.

4.2.4.6 calibrateListAAAAAAA

```
long cps_ts_gui::calibrateListAAAAAAA[51200]
```

A public long[] variable.

This array is used to store the calibration data from coil A.

Definition at line 219 of file cps_ts_gui.h.

4.2.4.7 calibrateListBBBBBBB

```
long cps_ts_gui::calibrateListBBBBBBB[51200]
```

A public long[] variable.

This array is used to store the calibration data from coil B.

Definition at line 226 of file cps_ts_gui.h.

4.2.4.8 calibrateListCCCCCCC

```
long cps_ts_gui::calibrateListCCCCCCC[51200]
```

A public long[] variable.

This array is used to store the calibration data from coil C.

Definition at line 233 of file cps_ts_gui.h.

4.2.4.9 componentID

```
QString cps_ts_gui::componentID = "Component ID not selected"
```

A public QString variable.

This variable is used to store the component ID of the component being tested. It is initialised to "Component ID not selected".

Definition at line 164 of file cps_ts_gui.h.

4.2.4.10 currentStep

```
int cps_ts_gui::currentStep = 0
```

A public int variable.

This variable is used to store the current step position.

Definition at line 80 of file cps_ts_gui.h.

4.2.4.11 currentValue

```
QString cps_ts_gui::currentValue = "0.31 RMS Current"
```

A public QString variable.

This variable is used to decide the current value and is initialised as 0.31 RMS current.

Definition at line 133 of file cps_ts_gui.h.

4.2.4.12 dataCoilA

```
QString cps_ts_gui::dataCoilA
```

A public QString variable.

This variable is used to store the data from coil A

Definition at line 185 of file cps_ts_gui.h.

4.2.4.13 dataCoilB

```
QString cps_ts_gui::dataCoilB
```

A public QString variable.

This variable is used to store the data from coil B

Definition at line 191 of file cps_ts_gui.h.

4.2.4.14 dataCoilC

```
QString cps_ts_gui::dataCoilC
```

A public QString variable.

This variable is used to store the data from coil C

Definition at line 197 of file cps_ts_gui.h.

4.2.4.15 dataList

```
QStringList cps_ts_gui::dataList
```

A public QStringList variable.

This list is used to store the incoming data from the arduino. This list is then split to extract the different coil data.

Definition at line 205 of file cps_ts_gui.h.

4.2.4.16 dataTimer

```
QTimer cps_ts_gui::dataTimer
```

A public QTimer variable.

This variable is used as a timer for plotting graphs on the GUI.

Definition at line 126 of file cps_ts_gui.h.

4.2.4.17 directionValue

```
int cps_ts_gui::directionValue = 0
```

A public int variable.

This variable is used to decide what direction the stepper motor shall move.

Definition at line 73 of file cps_ts_gui.h.

4.2.4.18 dirModel

```
QFileSystemModel* cps_ts_gui::dirModel [private]
```

A private variable.

This is a pointer object. This is the object used as the directory view.

Definition at line 551 of file cps_ts_gui.h.

4.2.4.19 distanceMode

```
std::string cps_ts_gui::distanceMode = "step"
```

A public string variable.

This variable is used to store the type of distance the user can control. It is initialised to steps.

Definition at line 66 of file cps_ts_gui.h.

4.2.4.20 distanceValue

```
long cps_ts_gui::distanceValue = 1
```

A public long variable.

This variable is used to control how far the user wants to turn on the stepper motor. It is initialised as 1.

Definition at line 103 of file cps_ts_gui.h.

4.2.4.21 fileModel

```
QFileSystemModel* cps_ts_gui::fileModel [private]
```

A private variable.

This is a pointer object. This is the object that is used as the file view.

Definition at line 559 of file cps_ts_gui.h.

4.2.4.22 gearRatio

```
QString cps_ts_gui::gearRatio = "Gear ratio not selected"
```

A public QString variable.

This variable holds information of the gear ratio the system is testing with. It is initialised to "Gear ratio not selected".

Definition at line 141 of file cps_ts_gui.h.

4.2.4.23 motorEnable

```
short cps_ts_gui::motorEnable = 0
```

A public short variable.

This variable is used to control if the stepper motor shall be enabled or not.

Definition at line 87 of file cps_ts_gui.h.

4.2.4.24 notesInput

```
QString cps_ts_gui::notesInput = "Notes not selected"
```

A public QString variable.

This variable stores the notes for the testing. It is initialised as "Notes not selected"

Definition at line 179 of file cps_ts_gui.h.

4.2.4.25 parsedData

```
QString cps_ts_gui::parsedData [private]
```

A private QString variable.

This variable is used to store the recieved data from the serial communication. The variable is then split to extract the different coildata.

Definition at line 590 of file cps_ts_gui.h.

4.2.4.26 projectName

```
QString cps_ts_gui::projectName = "Project name not selected"
```

A public QString variable.

This variable is used to store the project name. It is initialised to "Project name not selected".

Definition at line 148 of file cps_ts_gui.h.

4.2.4.27 serialBuffer

```
QString cps_ts_gui::serialBuffer [private]
```

A private QString variable.

This variable is used to buffer the recived bytes from the serial communication.

Definition at line 582 of file cps_ts_gui.h.

4.2.4.28 serialData

```
QByteArray cps_ts_gui::serialData [private]
```

A private variable.

This is a QByteArray and is the first variable to store the data from the serial communication.

Definition at line 575 of file cps_ts_gui.h.

4.2.4.29 speedMode

```
std::string cps_ts_gui::speedMode = "step"
```

A public string variable.

This variable is used to store the type of speed the user can control. It is initialised to steps per second.

Definition at line 58 of file cps_ts_gui.h.

4.2.4.30 speedValue

```
long cps_ts_gui::speedValue = 1
```

A public long variable.

This variable is used to control what speed the user wants to set on the stepper motor. It is initialised as 1.

Definition at line 95 of file cps_ts_gui.h.

4.2.4.31 stepMode

```
std::string cps_ts_gui::stepMode = "128"
```

A public string variable.

This variable is used to change the stepmode. It is initialised to the largest resolution of the stepper motor.

Definition at line 50 of file cps_ts_gui.h.

4.2.4.32 testID

```
QString cps_ts_gui::testID = "Test ID not selected"
```

A public QString variable.

This variable is used to store the test ID for the test being performed. It is initialised to "Test ID not selected".

Definition at line 172 of file cps_ts_gui.h.

4.2.4.33 ui

```
Ui::cps_ts_guiClass cps_ts_gui::ui [private]
```

A private variable.

The user interface object.

Definition at line 543 of file cps_ts_gui.h.

4.2.4.34 userName

```
QString cps_ts_gui::userName = "UserName not selected"
```

A public QString variable.

This variable is used to store the username of the active user performing tests. It is initialised to "UserName not selected".

Definition at line 156 of file cps_ts_gui.h.

4.2.4.35 writePDF

```
bool cps_ts_gui::writePDF = true
```

A public boolean variable.

This variable is used to decide if a PDF version of the log is to be generated or not. It is initialised to true.

Definition at line 111 of file cps_ts_gui.h.

The documentation for this class was generated from the following file:

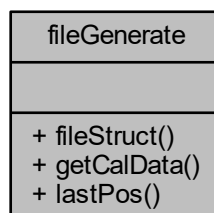
- C:/Users/magnu/Desktop/Doxygen/Doxygen source/cps_ts_gui_21_05_2019 - Copy/cps_ts_gui.h

4.3 fileGenerate Class Reference

A class for generating files and getting calibration data.

```
#include <fileGenerate.h>
```

Collaboration diagram for fileGenerate:



Public Member Functions

- void [fileStruct](#) (std::string &prName, std::string &cName, std::string &tID, std::string &peName, std::string ¬e, std::string &sMode, int sModeInt, std::string &cur, std::string &dist, std::string &spd, bool pdfGen, bool calib, std::string &gearRatio, long calibA[], long calibB[], long calibC[], std::string &coilAV, std::string &coilBV, std::string &coilCV, std::string &expectedValueA, std::string &expectedValueB, std::string &expectedValueC, std::string ¤tStep, std::string &calibratedDegree)

Function to create directory and xml file.

- std::string [getCalData](#) (int step)

Function to get data from the calibration XML file.

- int [lastPos](#) (int pos, int direction)

Function to log the last position.

4.3.1 Detailed Description

A class for generating files and getting calibration data.

This class has three functions; [filestruct\(\)](#), [getCalData\(\)](#) and [lastPos\(\)](#). Filestruct takes the different parameters and projectinformation set by the user, the sensordata stored in the calibration file and the current sensordata, and generates a XML file.

Definition at line 13 of file fileGenerate.h.

4.3.2 Member Function Documentation

4.3.2.1 fileStruct()

```
void fileGenerate::fileStruct (
    std::string & prName,
    std::string & cName,
    std::string & tID,
    std::string & peName,
    std::string & note,
    std::string & sMode,
    int sModeInt,
    std::string & cur,
    std::string & dist,
    std::string & spd,
    bool pdfGen,
    bool calib,
    std::string & gearRatio,
    long calibA[],
    long calibB[],
    long calibC[],
    std::string & coilAV,
    std::string & coilBV,
    std::string & coilCV,
    std::string & expectedValueA,
    std::string & expectedValueB,
    std::string & expectedValueC,
```

```
std::string & currentStep,  
std::string & calibratedDegree )
```

Function to create directory and xml file.

Current working function for generating a directory structure and xml file to log parameters and settings from an external source. The parameters are all the variables that will be gathered from the GUI

Parameters

<i>prName</i>	A string parameter for the project name
<i>cName</i>	A string parameter for the component id
<i>tID</i>	A string parameter for the test id
<i>peName</i>	A string parameter for the personnel name
<i>note</i>	A string parameter for notes about the test
<i>sMode</i>	A string parameter for stepmode setting of the driver
<i>sModeInt</i>	A Int parameter for stepmode setting of the driver
<i>cur</i>	A string parameter for the current setting of the driver
<i>dist</i>	A string parameter for the distance setting of the driver
<i>spd</i>	A string parameter for the speed setting of the driver
<i>pdfgen</i>	A bool parameter for the choice of generating a pdf
<i>calib</i>	A bool parameter for the choice of calibration
<i>gearRatio</i>	A string parameter for the gearratio of the system (if available)
<i>calibA[]</i>	A long array parameter containing calibration data of coil A
<i>calibB[]</i>	A long array parameter containing calibration data of coil B
<i>calibC[]</i>	A long array parameter containing calibration data of coil C
<i>coilAV</i>	A string parameter for the active value of coil A
<i>coilBV</i>	A string parameter for the active value of coil B
<i>coilCV</i>	A string parameter for the active value of coil C
<i>expectedValueA</i>	A string parameter for the expected value of coil A
<i>expectedValueB</i>	A string parameter for the expected value of coil B
<i>expectedValueC</i>	A string parameter for the expected value of coil C
<i>currentStep</i>	A string parameter for the current step position
<i>calibratedDegree</i>	A string parameter for the calibrated degree

4.3.2.2 getCalData()

```
std::string fileGenerate::getCalData (
    int step )
```

Function to get data from the calibration XML file.

This function takes the current number of steps and parses through the calibration file, until it reaches the same amount of steps taken and returns the CPS data on all three coils and the calibrated degree as a string.

Parameters

<i>step</i>	An int parameter that holds the current step position
-------------	-------------------------------------------------------

4.3.2.3 lastPos()

```
int fileGenerate::lastPos (
```

```
int pos,
int direction )
```

Function to log the last position.

This function takes the current number of steps taken and the direction of the movement and updates the position then returns it. This way the position is saved between system restarts

Parameters

<i>step</i>	An int parameter that holds the current number of steps taken
<i>step</i>	An int parameter that holds the direction the steps took

The documentation for this class was generated from the following file:

- C:/Users/magnu/Desktop/Doxygen/Doxygen source/cps_ts_gui_21_05_2019 - Copy/[fileGenerate.h](#)

4.4 pdfGenerate Class Reference

A class for generating PDF reports.

```
#include <pdfGenerate.h>
```

Collaboration diagram for pdfGenerate:



Public Member Functions

- int [fileStruct](#) (char *date, char *fName, std::string &prName, std::string &cName, std::string &tID, std::string &peName, std::string ¬e, std::string &sMode, std::string &cur, std::string &dist, std::string &spd, bool calib, std::string &gearRatio, long calibA[], long calibB[], long calibC[], std::string &coilAV, std::string &coilBV, std::string &coilCV, std::string &expectedValueA, std::string &expectedValueB, std::string &expectedValueC, float degreeValue, std::string ¤tStep)

Function to generate a PDF report file.

4.4.1 Detailed Description

A class for generating PDF reports.

This class generates the PDF documents for both the calibration run and the regular reports.

Definition at line 11 of file pdfGenerate.h.

4.4.2 Member Function Documentation

4.4.2.1 fileStruct()

```
int pdfGenerate::fileStruct (
    char * date,
    char * fName,
    std::string & prName,
    std::string & cName,
    std::string & tID,
    std::string & peName,
    std::string & note,
    std::string & sMode,
    std::string & cur,
    std::string & dist,
    std::string & spd,
    bool calib,
    std::string & gearRatio,
    long calibA[],
    long calibB[],
    long calibC[],
    std::string & coilAV,
    std::string & coilBV,
    std::string & coilCV,
    std::string & expectedValueA,
    std::string & expectedValueB,
    std::string & expectedValueC,
    float degreeValue,
    std::string & currentStep )
```

Function to generate a PDF report file.

Current working function for generating a PDF report file to log parameters, settings and test results from an external source. The values for the pointers are generated in fileGenerate.cpp while the other parameters are all variables that will be gathered from the GUI

Parameters

<i>date</i>	A pointer parameter to a char variable containing the current date
<i>fName</i>	A pointer parameter to a char variable containing the save path and file name
<i>prName</i>	A string parameter for the project name
<i>cName</i>	A string parameter for the component id
<i>tID</i>	A string parameter for the test id
<i>peName</i>	A string parameter for the personnel name
<i>note</i>	A string parameter for notes about the test
<i>sMode</i>	A string parameter for stepmode setting of the driver
<i>cur</i>	A string parameter for the current setting of the driver
<i>dist</i>	A string parameter for the distance setting of the driver
<i>spd</i>	A string parameter for the speed setting of the driver
<i>calib</i>	A bool parameter for the choice of calibration
<i>gearRatio</i>	A string parameter for the gearratio of the system (if available)

Parameters

<i>calibA[]</i>	A long array parameter containing calibration data of coil A
<i>calibB[]</i>	A long array parameter containing calibration data of coil B
<i>calibC[]</i>	A long array parameter containing calibration data of coil C
<i>coilAV</i>	A string parameter for the active value of coil A
<i>coilBV</i>	A string parameter for the active value of coil B
<i>coilCV</i>	A string parameter for the active value of coil C
<i>expectedValueA</i>	A string parameter for the expected value of coil A
<i>expectedValueB</i>	A string parameter for the expected value of coil B
<i>expectedValueC</i>	A string parameter for the expected value of coil C
<i>degreeValue</i>	A float parameter that holds the current position of the sensor in degrees
<i>currentStep</i>	A string parameter for the current position of the sensor in step

The documentation for this class was generated from the following file:

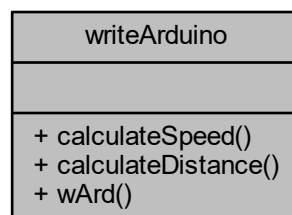
- C:/Users/magnu/Desktop/Doxygen/Doxygen source/cps_ts_gui_21_05_2019 - Copy/[pdfGenerate.h](#)

4.5 writeArduino Class Reference

A class for calculating parameters to arduino message code.

```
#include <writeArduino.h>
```

Collaboration diagram for writeArduino:



Public Member Functions

- long int [calculateSpeed](#) (std::string stepMode, std::string speedMode, int speedVal)
Function to calculate the command message content for speed.
- long int [calculateDistance](#) (std::string stepMode, std::string distMode, int distVal)
Function to calculate the command message content for distance.
- std::string [wArd](#) (int aEnb, long int aDist, int aSpd, int aDir)
Function to build the command message and send it to the Arduino.

4.5.1 Detailed Description

A class for calculating parameters to arduino message code.

This class calculates the speed and distance values from user parameters to values usable for the Arduino. It also combines the message string of parameters that is to be sent to the Arduino.

Definition at line 11 of file writeArduino.h.

4.5.2 Member Function Documentation

4.5.2.1 calculateDistance()

```
long int writeArduino::calculateDistance (
    std::string stepMode,
    std::string distMode,
    int distVal )
```

Function to calculate the command message content for distance.

This functions takes in the parameters set by the user and converts them to a value representing the distance that the stepper motor is to move.

Parameters

<i>stepMode</i>	A string parameter that holds the active resolution to the stepper motor
<i>speedMode</i>	A string parameter that holds the active distancemode
<i>speedMode</i>	A string parameter that holds the active speedvalue

4.5.2.2 calculateSpeed()

```
long int writeArduino::calculateSpeed (
    std::string stepMode,
    std::string speedMode,
    int speedVal )
```

Function to calculate the command message content for speed.

This functions takes in the parameters set by the user and converts them to a value representing the speed of the stepper motor.

Parameters

<i>stepMode</i>	A string parameter that holds the active resolution of the stepper motor
<i>speedMode</i>	A string parameter that holds the active speedmode
<i>speedMode</i>	A string parameter that holds the active speedvalue

4.5.2.3 wArd()

```
std::string writeArduino::wArd (
    int aEnb,
    long int aDist,
    int aSpd,
    int aDir )
```

Function to build the command message and send it to the Arduino.

This functions takes in parameters and creates the command for the movement that is to be performed and then sends it to the Arduino.

Parameters

<i>aEnb</i>	An int parameter that holds the enable/disable signal for the stepper motor
<i>aDist</i>	An int parameter that holds the calculated command content for the movement of the stepper motor
<i>aSpd</i>	An int parameter that holds the calculated command content for the speed of the stepper motor
<i>aDir</i>	An int parameter that holds the direction the driver is to turn

The documentation for this class was generated from the following file:

- C:/Users/magnu/Desktop/Doxygen/Doxygen source/cps_ts_gui_21_05_2019 - Copy/[writeArduino.h](#)

Chapter 5

File Documentation

5.1 C:/Users/magnu/Desktop/Doxygen/Doxygen source/cps_ts_gui_21_05_2019 - Copy/arduino_control_final.c File Reference

Classes

- class [arduino_control_final](#)

Macros

- #define [ARDUINO_CONTROL_FINAL_H](#)
- #define [BAUD](#) 9600
- #define [DELAY_TIME](#) 100
- #define [INPUT_SIZE](#) 255

5.1.1 Macro Definition Documentation

5.1.1.1 ARDUINO_CONTROL_FINAL_H

```
#define ARDUINO_CONTROL_FINAL_H
```

Definition at line 3 of file arduino_control_final.c.

5.1.1.2 BAUD

```
#define BAUD 9600
```

Definition at line 4 of file arduino_control_final.c.

5.1.1.3 DELAY_TIME

```
#define DELAY_TIME 100
```

Definition at line 5 of file `arduino_control_final.c`.

5.1.1.4 INPUT_SIZE

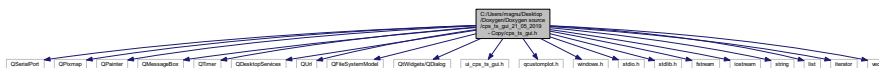
```
#define INPUT_SIZE 255
```

Definition at line 6 of file `arduino_control_final.c`.

5.2 C:/Users/magnu/Desktop/Doxygen/Doxygen source/cps_ts_gui_21_05_2019 - Copy/cps_ts_gui.h File Reference

```
#include <QSerialPort>
#include <QPixmap>
#include <QPainter>
#include <QMessageBox>
#include <QTimer>
#include <QDesktopServices>
#include <QUrl>
#include <QFileSystemModel>
#include <QtWidgets/QDialog>
#include "ui_cps_ts_gui.h"
#include "qcustomplot.h"
#include "windows.h"
#include <stdio.h>
#include <stdlib.h>
#include <fstream>
#include <iostream>
#include <string>
#include <list>
#include <iterator>
#include <vector>
```

Include dependency graph for `cps_ts_gui.h`:



Classes

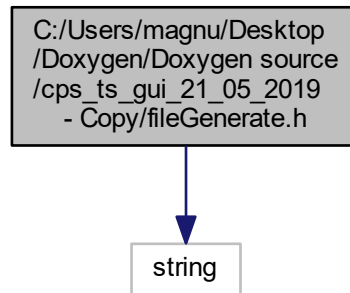
- class `cps_ts_gui`

A class for the graphical user interface and its content.

5.3 C:/Users/magnu/Desktop/Doxygen/Doxygen source/cps_ts_gui_21_05_2019 - Copy/fileGenerate.h File Reference

```
#include <string>
```

Include dependency graph for fileGenerate.h:



Classes

- class [fileGenerate](#)

A class for generating files and getting calibration data.

Macros

- `#define` [FILEGENERATE_H](#)

5.3.1 Macro Definition Documentation

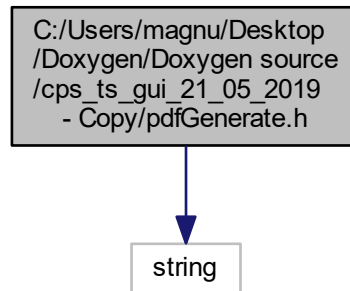
5.3.1.1 FILEGENERATE_H

```
#define FILEGENERATE_H
```

Definition at line 3 of file `fileGenerate.h`.

5.4 C:/Users/magnu/Desktop/Doxygen/Doxygen source/cps_ts_gui_21_05_2019 - Copy/pdfGenerate.h File Reference

```
#include <string>  
Include dependency graph for pdfGenerate.h:
```



Classes

- class [pdfGenerate](#)
A class for generating PDF reports.

Macros

- #define [PDFGENERATE_H](#)

5.4.1 Macro Definition Documentation

5.4.1.1 PDFGENERATE_H

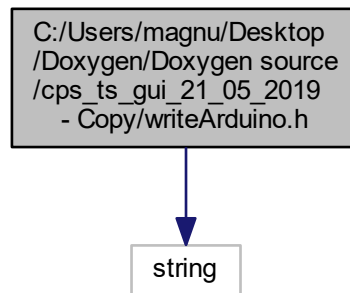
```
#define PDFGENERATE_H
```

Definition at line 3 of file pdfGenerate.h.

5.5 C:/Users/magnu/Desktop/Doxygen/Doxygen source/cps_ts_gui_21_05_2019 - Copy/writeArduino.h File Reference

```
#include <string>
```

Include dependency graph for writeArduino.h:



Classes

- class [writeArduino](#)
A class for calculating parameters to arduino message code.

Macros

- #define [WRITEARDUINO_H](#)

5.5.1 Macro Definition Documentation

5.5.1.1 WRITEARDUINO_H

```
#define WRITEARDUINO_H
```

Definition at line 3 of file writeArduino.h.

