University of
South-Eastern Norway

www.usn.no

FMH606 Master's Thesis 2024
Industrial IT and Automation

Batchwise dosing controller

Isak Skeie

Faculty of Technology, Natural Sciences and Maritime Sciences
Campus Porsgrunn

University of South-Eastern Norway

www.usn.no

Course: FMH606 Master's Thesis 2024
Title: *Batchwise dosing controller*
Pages: *129*
Keywords: *<Control System, Industrial Application, Batch Control>*

Student: *Isak Skeie*
Supervisor: *Carlos Pfeiffer*
External partner: *Goodtech, Biomar*

Summary:
A new controller scheme is developed for raw material dosing in pellet production; more specifically, in fish feed production with testing and implementation in a French plant. The proposed controller aims to address a number of challenges that are prevalent across different dosing configurations in plants around the world. The new controller scheme aspires to be more easily configurable and more robust against external disturbances, which should result in a dosing control system with an increased accuracy, fewer alarms produced, and less time spent tuning the system.

Through an iterative process the final dosing controller is developed. A proportional feedback controller is tested, which gave insight into the process dynamics with continuous control inputs. A multi-state feedback controller is then tested, where the weight is regulated based on a proportional controller, and the raw material flow rate being controlled by an integrating feedback controller. For the flow-based control to be reliable a filter is used to attenuate external disturbances. Non-linear weighted error functions are then used to obtain more desirable controller dynamics. Lastly an algorithm for optimal control parameter estimation is used to reduce the complexity when interacting with the new controller scheme.

By tuning the components of a performance index, one can easily control the accuracy vs the speed for each raw material, with disturbances successfully attenuated with a Kalman filter. This forms a succesfull controller scheme that's applicable for a range of dosing systems. A shortcoming to the new controller scheme, is the dependency on an estimated model that's sufficiently accurate. This proves challenging under control initialization, and when the dosing run is too short for the model estimator to extract meaningful data.

Preface

Several people and organizations have made contributions in creating this paper: with knowledge, insight and resources.
The thesis has been written for Goodtech, an Industrial Systems integrator. Goodtech have provided hardware, logistical aid and the support of colleagues with relevant competence. Goodtech colleague Marcus Nordhal raised the challenges and potential for a master thesis problem for raw material dosing in fish feed production. Having developed the current control scheme and with years of experience with the process. Marcus has helped uncover the challenges with raw material dosing, by understanding the root cause of the challenges, and how to address them.
A fundamental contributor in developing a new controller solution, has been BioMar's fish-feed factory in France. With the sites Continous Improvement Responsible, Ludovic Bonhome facilitating live testing which provided further insight, as well as results for the thesis. Lastly, Carlos Pfeiffer has been instructing and supervising the process of writing the master thesis. He has provided academic knowledge in setting the analytical foundation, and in finding appropriate methods. He has also aided the writing process to assure that each element of the thesis is presented in a structured and understandable way.

Porsgrunn, 14th May 2024

Isak Skeie

Contents

List of Figures

List of Tables

Nomenclature

Symbol	**Explanation**
PLC	Programmable Logic Controller: An industrial computer containing logic for operating hardware
MES	Manufacturing Execution System: ”computerized systems used in manufacturing to track and document the transformation of raw materials to finished goods”. [1]
SCADA	Supervisory Control and Data Acquisition: Control system for industrial applications, which contains graphical interfaces and programmable logic
API	Application Programming Interface: A communication protocoll which enables different programs and computers to communicate.
CSV	Comma Seperated Values: A file format, which allows data to be store in a table-like format.
JSON	A file format, where data objects is created. Often used as configuration files.
OLS	Ordinary Least Squares
MPC	Model Predictive controller
ILC	Iterative Learning Control
PID	Proportional-Integral-Derivative
cv	Control Value
SP	Set Point
e	Error
w	Weight in Kg
Δw	Change in weight

1 Introduction

In fish-feed pellet production, BioMar is one of the key players, with 18 factories around the world producing fish-feed. They are in a process of standardizing the industrial automation control system in each of their factories, by implemententing a global template. Scalable improvements would then have a huge potential, as it easily can be implemented across all their factories. Goodtech is an industrial systems integrator, and responsible for developing and implemententing the global standard to each of BioMar's factories. A firm understanding of the process of making fish feed, combined with a technical foundation; Goodtech is able to identify feasible areas of improvement and how to realize them. One of these areas is raw material dosing. The current raw material dosing is one of the largest sources of alarms in their site. The dosing controller is strenuous to get to perform well, and buckles under changing conditions.

1.1 Process Description

In pellet production, a composition of different raw materials goes through a number of processes before being turned into pellets. Pellet production is a batch process which starts off by dosing raw materials, before the mixture is ground, mixed, extruded, dried, coated with oil, and finally cooled. An array of silos contains the different raw materials necessary to create a pellet. Each of the raw materials serves a different purpose, some are providing protein and other nutritional additions, while other raw materials have more of a structural purpose for the pellet itself, like starch, which hardens the pellet under pressure and heat. The different raw materials thus greatly vary in price and characteristics when being handled. The first step in the process is to create the correct composition of different raw materials. A figure of the dosing screen for a fish feed factory can be seen in Figure 1.1 and Figure 1.2. Screw conveyors are marked with yellow in Figure 1.1 and 1.2, they are attached at the bottom of each silo, which is marked with red, and transport the respective raw material to a weigh bin marked in green. In the case of the French plant, there are three weighing bins, each with a different range in kg. Figure 1.1 shows the largest scale which doses up to 500-600 kg of each raw material. Figure 1.2 shows two dosing systems, with the left weighing bin dosing raw materials in the range of 0.1-2 kg, and the right one 3-4 kg. The raw material composition is dependent on specific recipes that are meant to produce a specific product, with the process trying to dose as accurately as possible to the specified amount in the recipe.

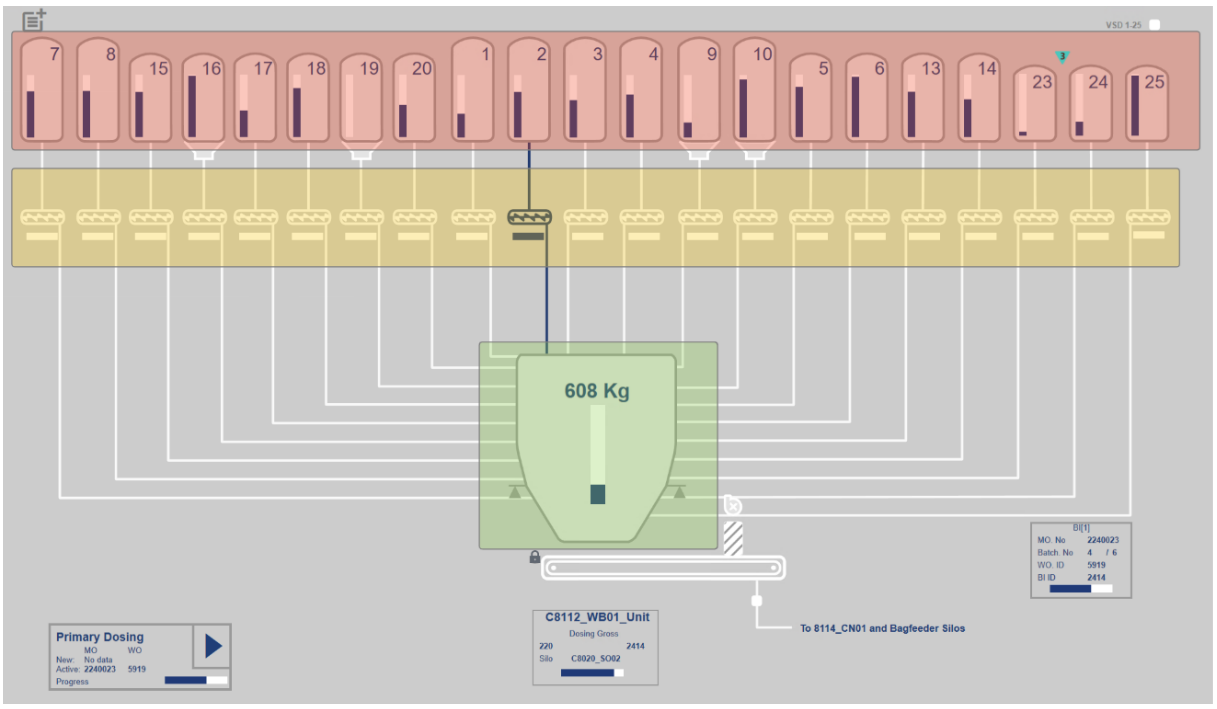


Figure 1.1: Primary dosing SCADA page

1.2 Dosing Review

The areas of importance and potential for improvement are mapped through separate interviews with staff in separate fish feed factories, one in France, a single-line factory, and a second in Northern Norway with parallel production lines. One of the main challenges they face is the number of alarms the current system is producing. An alarm is raised if the system doses too little, too much, or if the flow of material goes below a certain threshold. Reducing the number of alarms is important to increase the attentiveness to operators and to increase the perceived importance of each individual alarm. Alarms and, in turn, the accuracy of the dosing, are especially problematic when production of a new product is starting, or, as for the case of multiple lines, the same silo is dosed from two separate lines. Another common challenge is with the number of parameters used to control dosing for the current system. It's both difficult to understand and time-consuming to adjust them to optimize the system, with several hours being spent per week to perform adjustments. The accuracy of the dosing system is satisfactory most of the

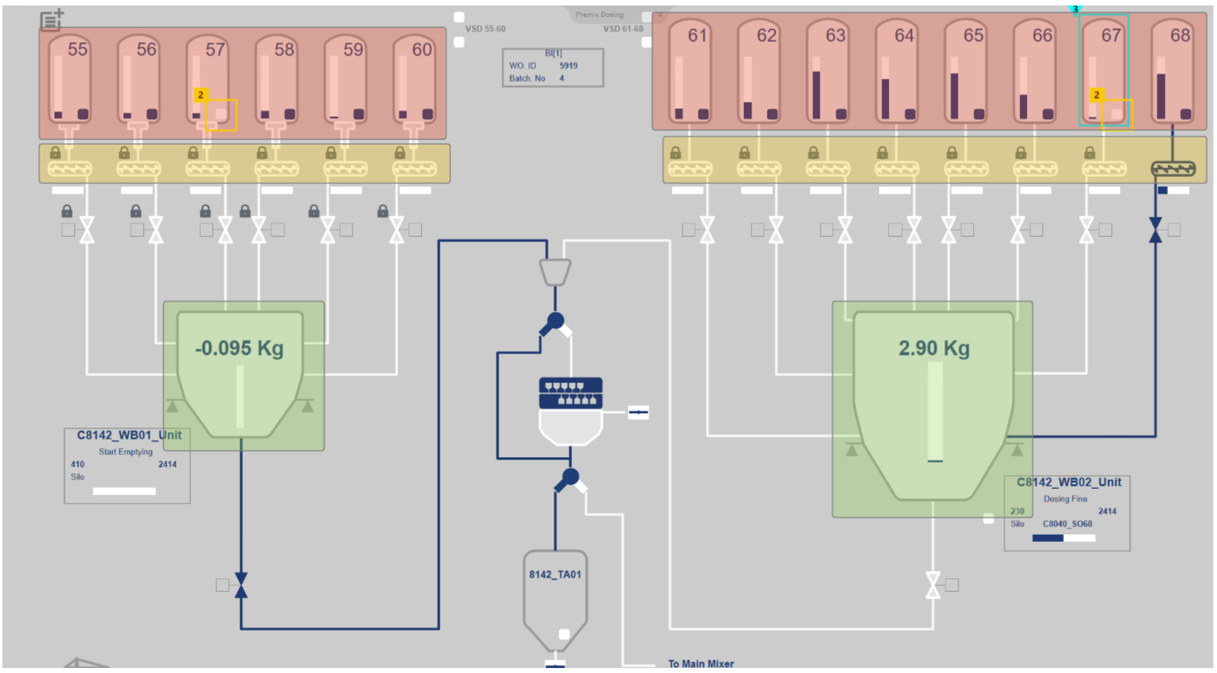


Figure 1.2: Micro dosing SCADA page

time, but with raw materials consisting of 70% of total costs, substantial savings could be made by increasing the accuracy of the dosing system. Disturbances in the dosing system are also a common problem, especially for smaller weigh bins. Vibrations from external equipment running and changes in pressure due to valve and slide gates opening and closing. These disturbances are interfering with the dosing process and affect the final product quality. Data analysis is necessary to examine the shortcomings of the current system and to establish the relevance of parameters to land on an appropriate solution.

1.2.1 Current Control Solution

Having an understanding of how the current dosing control works is beneficial for further analysis of the system. It would make it possible to identify if faults in dosing are due to shortcomings in the controller or something outside a controller's scope. A single PLC holds the logic for the entire plant and is where dosing logic resides. Figure 1.3 shows the data flow for the current control solution, The PLC receives a recipe from a MES system and uses the information given from the MES system to determine which silos to dose.

Table 1.1 shows the most important variables used in the current control scheme. The Screw conveyor is dosing raw material in predetermined speeds. It starts off in gross dosing, with the screw speed determined by DoseGross_speed. The amount specified in FineDose determines how far the weight is from the setpoint before the screw conveyor runs at the speed determined by DoseFine_Speed. Inflight gives the amount of raw material that is in the air or in its way after the screw conveyor has stopped. DoseAfter is used to compensate for deviations in the control system. By iteratively calculating the deviation from the setpoint, and using the deviation to alter the duration the screw is running for the next dosing run. DoseAfter_Speed is used to determine the speed of After dosing. FineDose, Inflight, and all of the speed variables need to be determined by an operator for each screw in the plant. The French plant has around 37 dosing screws, making the total amount of parameters to tune around 200. Figure 1.11, 1.12, and 1.13 all show the current control scheme in action. The parameters for gross and fine dosing become evident in these graphs, with a clear transition between the two of them.

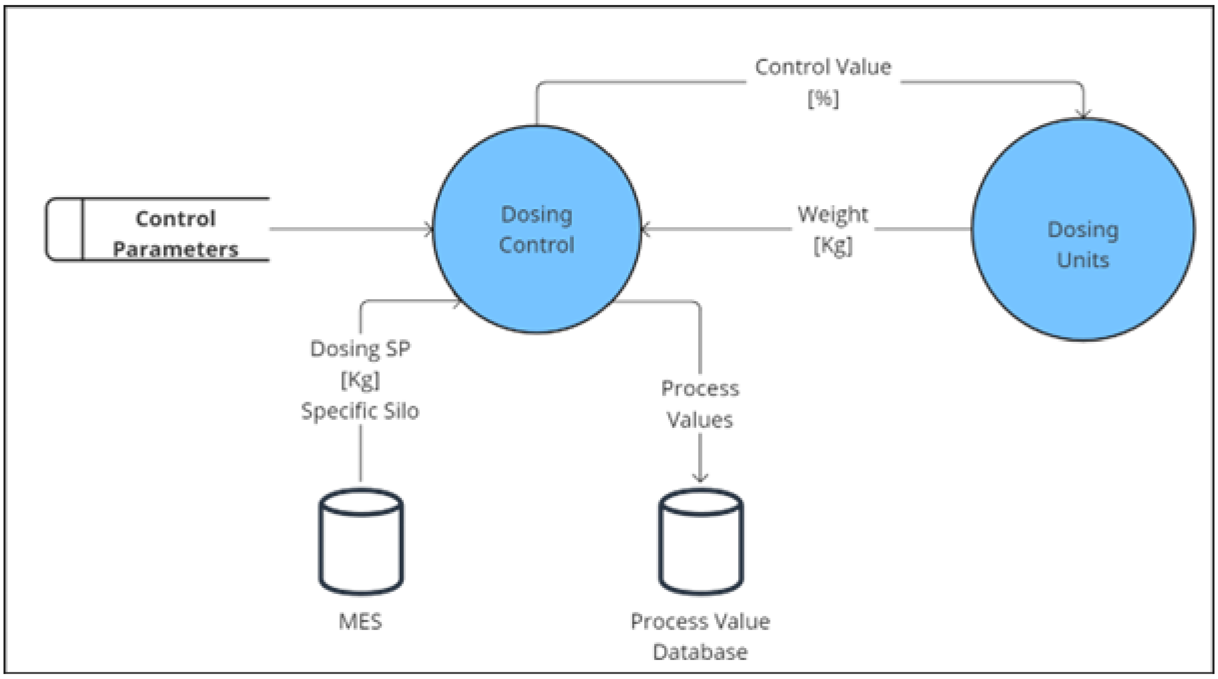


Figure 1.3: Data Flow for the processes with the current control scheme

1.3 Dosing Analysis

Before starting to look at time series data, it's beneficial to look at the batch data. Figure 1.4 shows the different components in the IT infrastructure of plant and how they are connected. The Process Value Database is an SQL server containing time series data for

Variable	Description	Unit
DoseFine	Point to start finer dosing	kg
DoseAfter	Iteratively calculated dosing amiun	kg
Inflight	How much of the raw material is mid air in transport	kg
DoseFine speed	Fine Dosing Speed	%
DoseGros speed	Gross Dosing speed	%
DoseAfter speed	After dosing speed	%

Table 1.1: Control Variable Explanation

all of the objects on the site,and through this database batch data is extracted with a storage procedure, a function programmed as a SQL script. 1.2 shows an overview of the different tags that are queried together with a brief explanation of them. Analyzing these data points should help clarify the relevance of each parameter to the system.

Variable	Description	Unit
DoseTotal	Total Amount Dosed on scale	kg
DoseTimeStart	Start Time of dosing	time
ScrewInputTotal	Accumulated screw input	$\%\cdot s$
ScrewWeightRatio	Ratio between amount dosed and accumulated screw input	$\frac{kg}{\%\cdot s}$
ScrewIdleTime	Difference in DoseTimeStart for each batch on each screw	seconds
SiloQuantity	Calculated amount present in silo	kg
Temperature	Room temperature for the site	C°
ddAfter	Delayed delivery of material to the scale after the screw has stopped	seconds

Table 1.2: Dosing related variables

Weight deviation

To extract the relevant information from data gathered from the current system, it is important to understand how the current system works. The current dosing system doses the raw materials in stages: Coarse, Fine, and After-Dosing. For each of these stages, a speed set point is assigned manually, with the fine dosing duration being found iteratively. The relevance of the external parameters needs to be based on the performance of the existing model. The correlation between the weight deviation and the absolute z-score for external parameters for several batches is calculated. Deviations from the mean for each parameter should affect the weight deviation if it has an effect on the material at all. The thought is that an iterative algorithm poorly handles deviations of parameters and scenarios outside the normal operating point, as the result from the previous batch becomes progressively irrelevant as the deviation increases. The results can be seen in Figure 1.5, 1.6, and 1.7.

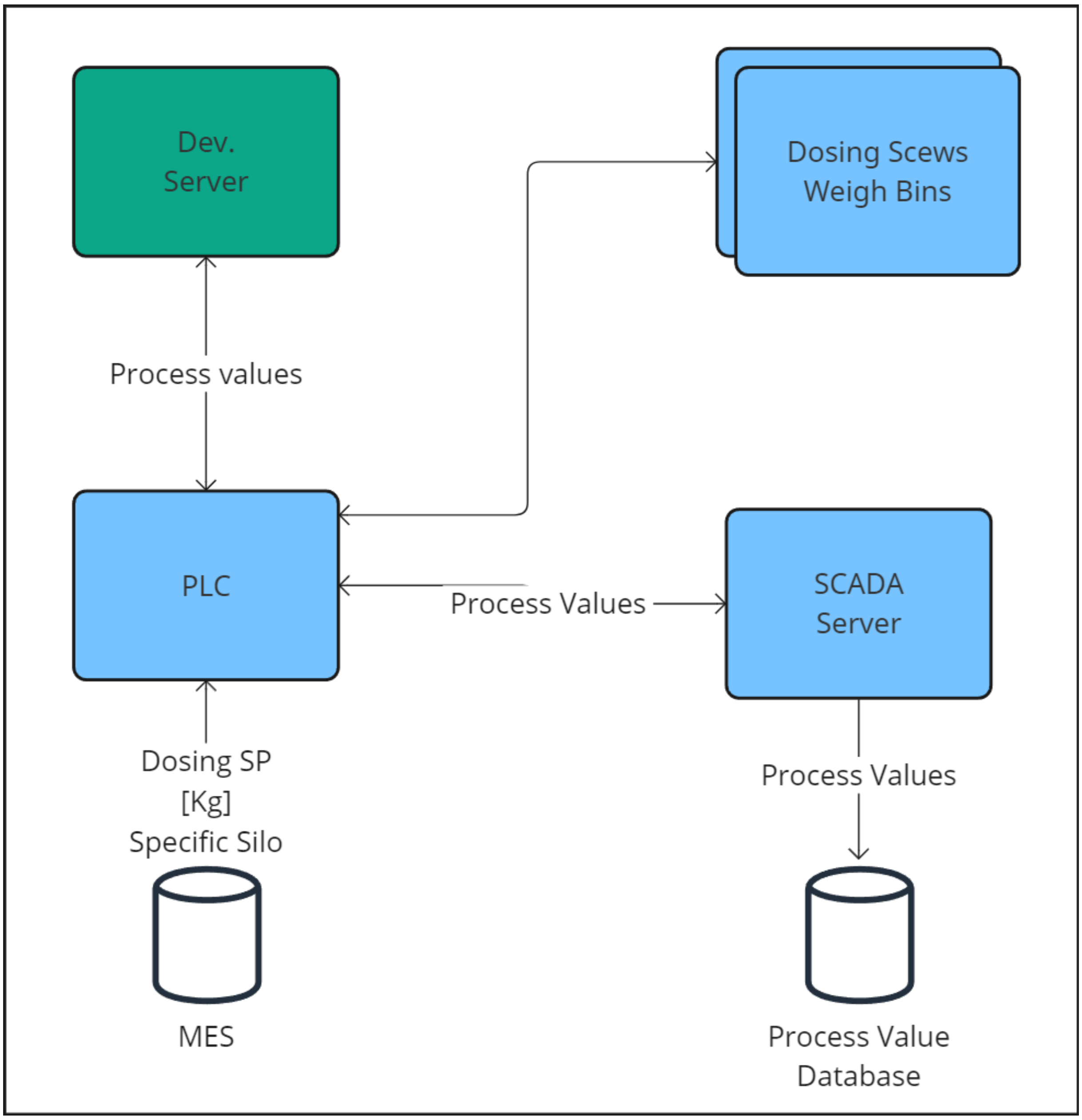


Figure 1.4: Data Flow for components in current Control solution

The 3 separate bar plots each represent parameters, with each bar being a separate screw (unique raw material). All of the parameters have a substantial degree of correlation, with the degree of correlation varying between the different raw materials. Based on the results given in the 3 bar graphs, screw idle time, temperature, and the quantity in a silo are affecting the dosing systems. With some of the raw materials like silo 25 being highly

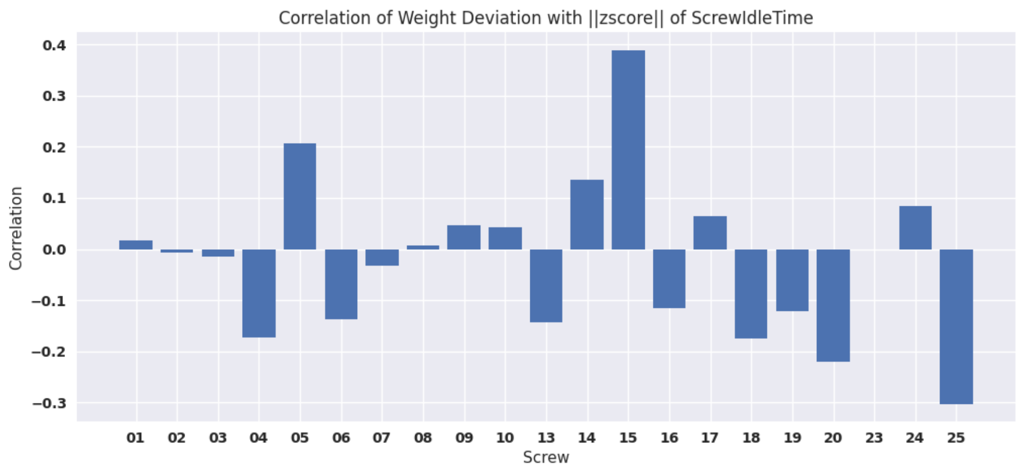


Figure 1.5: Z-Score between Screw Weight Ratio and ScrewIdleTime

prone to changes, while others like silo 2 not having any significant effect.

Material Density

Trying to look past the shortcomings of the existing system and still trying to evaluate the relevance of the parameters is done by calculating the correlation between the ratio of the change in weight over accumulated screw input for each parameter. A higher ratio should indicate a higher density of the raw material, which needs to be accounted for. Figure 1.8, 1.9, and 1.10 show the plot for the correlation calculation in the same way as with the z-scores. The plots provide the same insight, in that there is a significant variation in the relevance of each parameter for each raw material. The bar graphs indicate that the density is affected to a varying degree for each raw material by the respective variables.

Time-series data analysis and Non-Linearity

By looking at time-series data for the process, it should be possible to better understand the current process, control, and its effects on the weight. The goal of the current control process is identical to the new one being developed. The time-series data will, in this regard, clarify how this is currently being executed and how it fails to do so in some areas. Time-series data will, in turn, be useful for finding a proper control scheme. Figure 1.11, 1.12, and 1.13 show three different raw materials being dosed (with the remaining plots available in the appendix), each with a different setpoint and dosing screw. Out of the nine different time-series raw material dosing figures, three were chosen for their

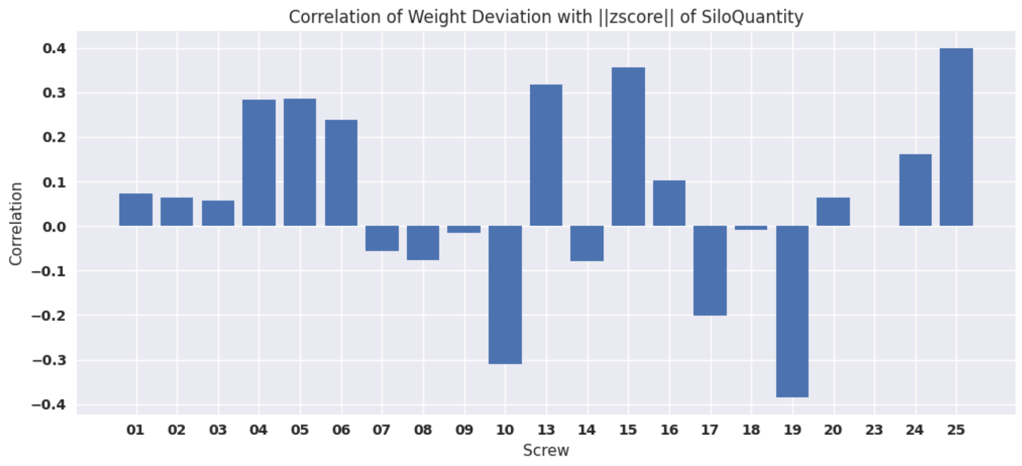


Figure 1.6: Z-Score between Screw Weight Ratio and SiloQuant

inherently different input response characteristics. Figure 1.13 shows the screw response characteristics for screw number 17.

Several dosing runs are plotted over each other to enable a more robust interpretation of the data. For Screw 17 in Figure 1.13, the dosing screw starts off at 100% for around 3 seconds, and the weight overshoots considerably. The apparent overshoot introduces several challenges to the current control process. It finishes the dosing too early, making it necessary to retry the same dosing procedure. By retrying the dosing procedure, alarms are generated, and new overshoots are introduced as the weight gets closer to the setpoint. The dosing retries can be seen by the screw speed going down to 0% before rising to 30% for a retry. Both the presence of overshoots and its delay suggest that for some of the raw materials, the dosing is non-linear in nature.

Figure 1.11 shows Screw number 8 and its corresponding weight. This is an example of the dosing control working as intended. Looking closely at the transition between 100% and 30%, there is a negligible overshoot. Screw 8 has a weight setpoint around 300kg, compared to 80kg for screw 17. The relative screw speeds are, in this case, much slower, making the dosing characteristics appear significantly more linear compared to the dosing with screw 17. Screw 10 shown in Figure 1.12 exhibits much of the same characteristics as screw 8, with the dosing setpoint being different. Another difference is the oscillations in the weight. With the screw speed running at 100%, the oscillations are considerably higher than they are for the screw speed running at 30%. An explanation for this could be found in the way a dosing screw works. A screw is placed inside a pipe, transporting material through moving chambers. With an increase in the screw speed, the pockets of material will be delivered to the weigh bin in bursts.

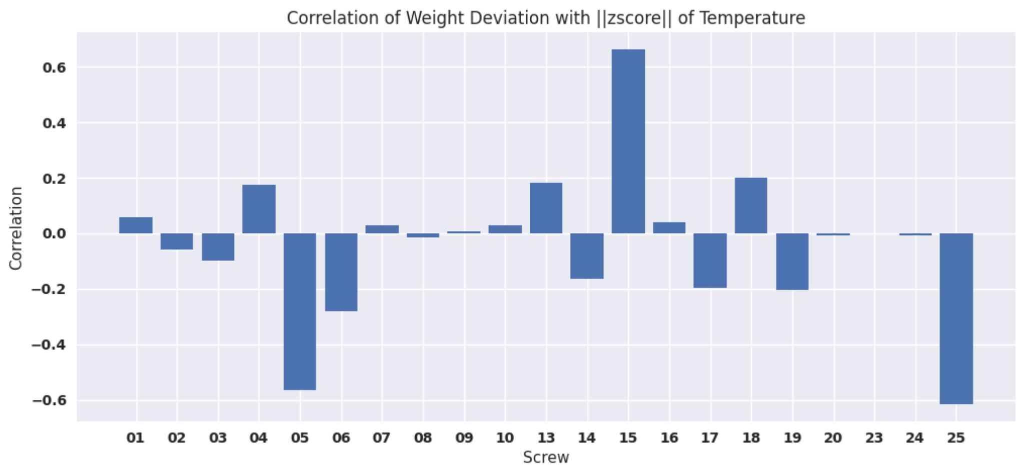


Figure 1.7: Z-Score between Screw Weight Ratio and Temperature

1.3.1 Literature Review

The dosing review and data analysis need to be accounted for when finding a new approach for raw material dosing. The dynamics, non-linearities, uncertainties, and disturbances of the dosing system have to be handled by a potential solution. Feedback Control, Machine Learning Control, Model Predictive Control, and Iterative Learning Control are all relevant solutions to the problem. Feedback controllers are one of the most widespread types of controllers throughout a vast number of industries, specifically the PID controller. A major challenge with feedback control is instability in the control scheme. The correct control parameters have to be found in order to mitigate the risk of instability and to ensure adequate control performance [2]. When using a feedback controller, non-linear weighted feedback could be utilized to generate controller dynamics that are more desirable in the case for raw material dosing. In order for feedback control to be applicable in the case of raw material dosing, control parameters have to be self-tuned. With an estimated dosing model, a minimization algorithm could be used against a performance index to find the ideal parameters [3].

Within machine learning control, a Reinforcement Learning Algorithm could be used to control the system. Using Reinforcement Learning has the benefit of being model-free but is both computationally intensive and requires a large amount of data. A Reinforcement Learning Algorithm will also have to ”try” and ”fail”, which in industrial applications might not be feasible or economically viable [4]. Iterative Learning Control could also be used, a popular method for batch processes. A version of ILC is already in place. Its shortcoming has been highlighted in the previous dosing review and analysis. If the

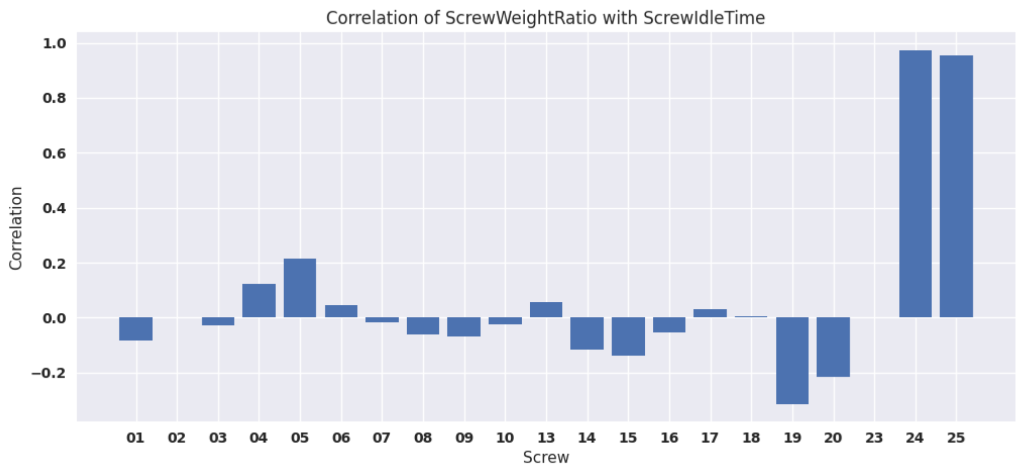


Figure 1.8: Correlation between ScrewWeightRatio and ScrewIdleTime

process gets outside the normal operating point or the process experiences disturbances during batching, the control performance would start to deteriorate. The third control method is Model Predictive Control, which would handle in-line disturbances better, with the presumption of an accurate enough model. Additionally, the computational efficiency needs to be high enough for a model predictive controller.

A number of control approaches are discussed. A model estimation is necessary. It is therefore important to have a good strategy for model estimation. An estimated model should be as simple as possible while still being able to pick up the general system characteristics. The model estimation strategy would also need to accommodate time-varying model parameters. Dealing with mechanical equipment in a real industrial process will cause the model to change its parameters over time due to wear, maintenance, changes to the process, etc [5].

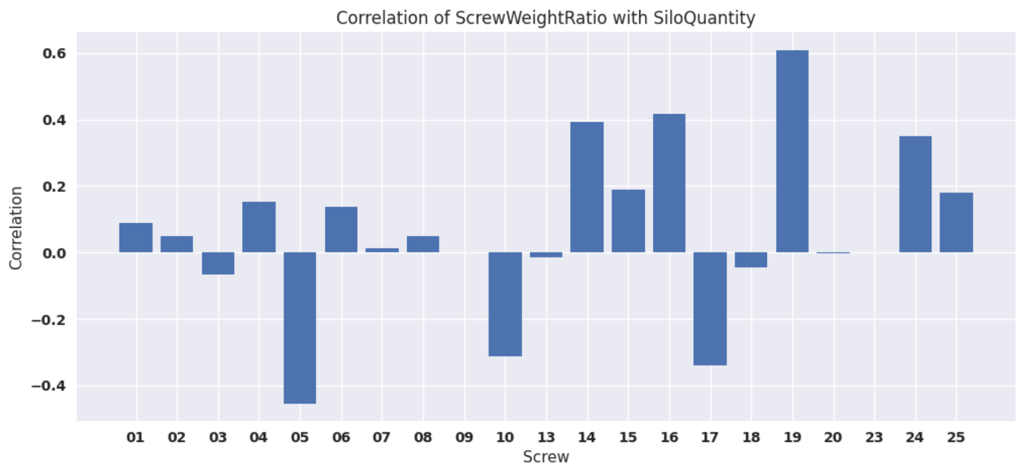


Figure 1.9: Correlation between ScrewWeightRatio and SiloQuantity

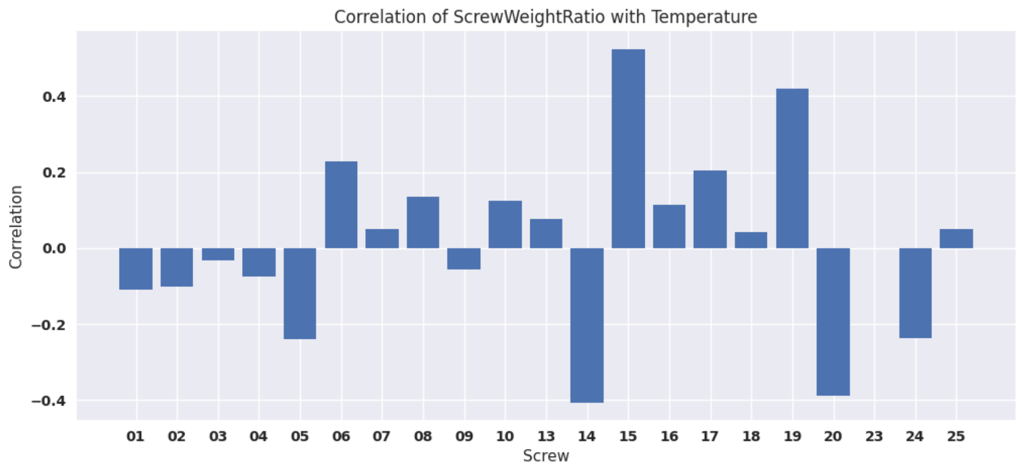


Figure 1.10: Correlation between ScrewWeightRatio and Temperature

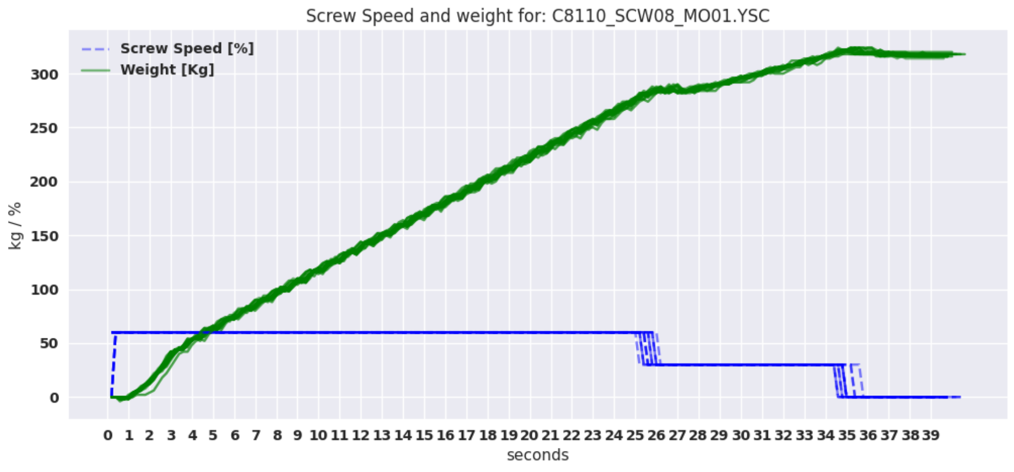


Figure 1.11: Dosing Control for screw 8 with existing control system

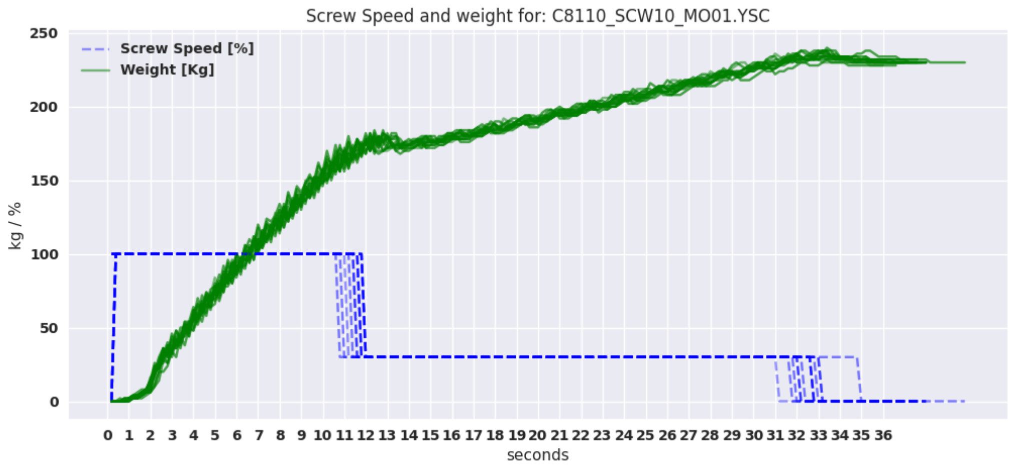


Figure 1.12: Dosing Control for screw 10 with existing control system

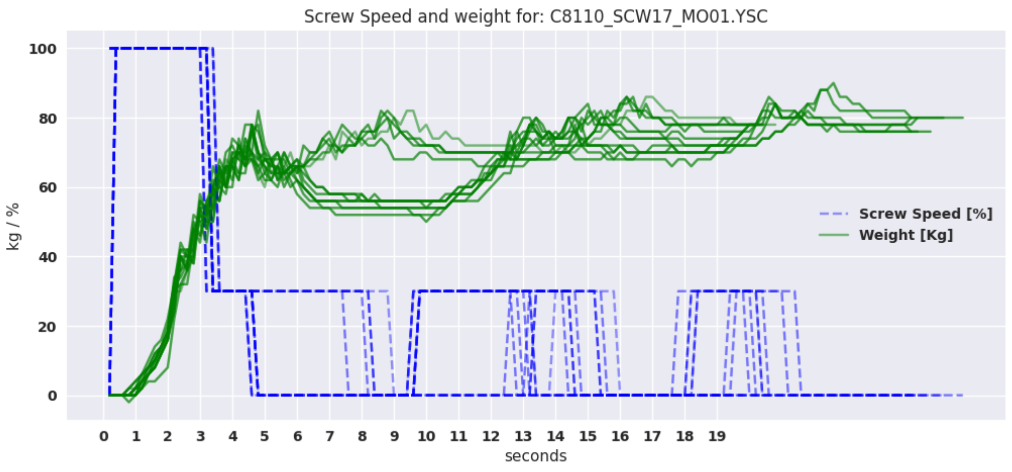


Figure 1.13: Dosing Control for screw 17 with existing control system

2 Methods

Based on the literature review, a number of solutions could prove viable in raw material dosing. In finding the final solution, complexity should be kept at a minimum while adhering to all the requirements of an industrial process. Having the dosing review and dosing analysis based on the current control scheme is, to a certain degree, limiting our understanding of how the system responds and the utility of a new control system. In order to gain a better understanding of the system and potential new hurdles, several dosing control methods are tried out, each providing more insight into the process. Different control schemes are considered through an iterative process, with an increase in complexity for each step.
Before starting controller development, a control architecture needs to be established. Each silo in the existing controller scheme uses an exclusive controller, hence the large number of parameters. In favor of parameter reduction, one could have a single controller for each of the weighing bins. This would impose additional challenges for potential controllers, as they need to account for the different raw material characteristics. Instead, the controller architecture from the original controller scheme is kept.

2.1 Proportional Feedback Controller

The first developed dosing controller is the P-controller. Having the additional Integral and derivative part would not be of any use to the dosing process. When dosing, material gets added, and not subtracted, the process itself is integrating. There would therefor be no use of an integral feedback controller. A Derivative feedback controller would for the same reason be irrelevant, the process is not trying to reach a steady state, meaning there is no need to compensate for changes in the flow.

The simplest and one of the most widespread forms of control. A proportional feedback controller creates a control input that is proportional to the deviation between the state and setpoint. By itself, the P-controller wouldn't work for raw material dosing. When the deviation gets smaller, so does the proportional gain, and the state would never reach the desired setpoint, but instead converge towards it. The asymptotic behavior also becomes problematic when considering the dosing time. It needs to be fast enough to not introduce bottlenecks in the production line. A solution could be to increase the gain, but this would in turn increase the instability of the control system.

2.1.1 Heuristics and considerations for P-Controller

For a P-controller to be able to control the dosing system, some heuristic logic needs to be in place. The initial speed needs to be determined by the setpoint multiplied by a coefficient. This ensures that smaller dosing amounts start off at slower speeds, reducing potential overshoots and non-linearities. A bias is also added to ensure that the screw doesn't run too slowly to be able to finish off the dosing. The bias is proportional to the initial speed, determined by a coefficient. Lastly, logic for when the P-controller is set to activate is also added. The controller runs with the initial speed until the weight reaches a specified percentage of the setpoint. Then the proportional gain becomes active.

Intial parameters

$$\text{Initial Control output: } cv_{ini} = sp * cv_{ini_k} \tag{2.1}$$

$$\text{Limits Control output: } cv_{ini} \begin{cases} 100 & \text{if } cv_{ini} > 100, \\ u_{ini_k} & \text{otherwise.} \end{cases} \tag{2.2}$$

$$\text{Bias: } C = cv_i ni * C_k \tag{2.3}$$

P-Control

$$cv(k) \begin{cases} cv_{ini} & \text{if } e > cp \\ cv_{ini} * (1 - pv/sp) * K_p & \text{otherwise} \end{cases} \tag{2.4}$$

$$cv(k) \begin{cases} C & \text{if } cv(k) < C \\ cv_{ini} & \text{if } cv(k) > cv_{ini} \\ 0 & \text{if } e < \text{cutoff} \\ cv(k) & \text{otherwise} \end{cases} \tag{2.5}$$

where:

- C is the minimum control output
- cv_{ini} is the initial control output

- cv_{ini_k} is the gain for finding the initial control output
- cutoff is the point where the control output is set to 0
- cp is the "Control Point", where the p-controller takes over from the initial speed
- sp is the setpoint
- $cv(t)$ is the control output,
- K_p is the proportional gain,
- $e(t)$ is the error signal, defined as the difference between the desired setpoint and the actual process variable.

2.1.2 Review

In order for the P-controller to be applicable to the dosing system at hand, a number of heuristic logical statements are put in place to ensure stability while keeping the dosing speed within acceptable measures. With these considerations in place, a number of new variables are added to the P-controller scheme. The old control scheme had 5 parameters to tune, while this P-Controller has the same amount, giving no real improvement in usability. It also makes it harder to propagate the controller to other dosing systems. Nevertheless, trying out the P-Controller is useful to gain additional insight into how the system behaves under a continuous control input. which is not the case for the current control system.

2.2 Preliminaries

Moving on from the P-Controller, a priori data from the dosing systems is incorporated into the dosing control schemes and utilized to decrease the uncertainties of the states. This is done by employing a model of the system and filtering the measurements.

2.2.1 Model Estimation

Selection a good general model structure is adamant for a model to explain as much variance in the system as possible, while at the same time keeping the complexity to a minimum. If possible or sufficient, the estimated model should be linear. Too keep the computational speed up, and to be able to utilize classical control theory. The models are to predict the rate of change in each weighing bin. With each of the raw material silos with its corresponding screw being represented by separate models.

In developing the model, a simple first order linear model is used to predict the weight derivative from the screw speeds with the addition of lagged inputs. A general first order linear model is used instead of a physics-based model for its generality, without knowing how external factors are affecting the model, trial and error with different lags and nonlinear terms could prove viable as a general purpose solution. The discrete weight difference in kg is the dependent variable, and screw inputs in percent being the independent variable.

Each raw material dosing unit, comprises of several aspects affecting the model to be estimated: The specific raw materials, the dosing equipment, and the scale. Each of these aspects will change during production and over time. To handle this, live model estimation should be used. By implementing incoming data, and estimating the model live, system changes are accounted for. Theres several ways to perform a live estimation, with sliding window settled upon as the most practical, and easy to implement solution. A window of a predefined length contains previous process data, new data gets added to the window with the oldest data being removed in order to maintain a set size for the training data. The model is then estimated over again with the updated window. In estimating the model, Ordinary Least Squares is used to find the model parameters.

The relationship between screw speed and the weight difference could turn out to be non-linear in nature for some of the dosing units. This should ideally be addressed to get a better model fit, while the model still can be transferred to a state space representation. This can be solved by adding exponential versions of screw speed to the model, like screw speed squared, or the square root of the screw speed.

The linear model is given by:

$$\Delta\hat{w}(k+1) = b_{1,1}cv_1(k) + b_{1,2}cv_2(k) + ... + b_{n,m}cv_m(k-n) \tag{2.6}$$

The Ordinary Least Squares (OLS) formula is given by:

$$\hat{\beta} = (\mathbf{U}^T\mathbf{U} + \lambda * I)^{-1}\mathbf{U}^T\Delta\mathbf{w} \tag{2.7}$$

where:

- $\hat{\beta}$ is the estimated vector of parameters, $\hat{\beta} = \begin{bmatrix} b_{1,1} & ... & b_{n,m} \end{bmatrix}$

- $\mathbf{U}$ is 2D matrix of independent variables, consisting of vectors with control inputs.
  $\mathbf{U} = \begin{bmatrix} \mathbf{cv}(0)^T \\ \vdots \\ \mathbf{cv}(Np)^T \end{bmatrix}$

- $\mathbf{\Delta w}$ is the vector of the dependent variable. $\mathbf{\Delta w} = \begin{bmatrix} \Delta w(0) \\ \vdots \\ \Delta w(Np) \end{bmatrix}$
- Δw is the difference in weight in kg between time steps in , $w = w(k) - w(k-1)$
- $\mathbf{cv}(k)$ is a vector containing control inputs. $\mathbf{cv}(k) = \begin{bmatrix} cv_1(k) \\ \vdots \\ cv_m(k) \end{bmatrix}$
- b_n are vectors of coefficients for each lagged timestep, $b_n = \begin{bmatrix} b_n, 1 & \dots & b_n, m \end{bmatrix}$
- λ is a coefficient added to avoid singular matrices
- Np is the length of the sliding window,
- $cv_m(k)$ are the control inputs,
- n is the number of lagged variables,
- m is the number of input functions

State Space Model

By transferring the linear OLS model into a state space model, a more general formulation is used. If the model representing dosing is to stay linear, changes made to the model are easily implemented in the dosing application. The state vector contains the weight and weight derivative, with the state matrix accumulating the weight derivative to the weight. The control input vector takes in exogenous variables like the screw speeds, and if applicable, lagged and proportional variations of the screw speeds. The input matrix contains the coefficients found by OLS, which are added to the weight derivative state.

The state-space model is given by:

$$\begin{aligned} x(k+1) &= Ax(k) + Bu(k) \\ y(k) &= Cx(k) \end{aligned}$$

where:

- $x(k)$ is the state vector,

$$x(k) = \begin{bmatrix} w(k) \\ \Delta w(k) \end{bmatrix}$$

- $u(k)$ is the control input vector $u(k) = \mathbf{cv}(k)$

- $y(k)$ is the output vector,
- A is the state matrix $A = \begin{bmatrix} 1 & 1 \\ 0 & 0 \end{bmatrix}$
- B is the input matrix

  $$B = \begin{bmatrix} 0 \dots 0 \\ \hat{\beta} \end{bmatrix}$$

- C is the output matrix $C = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$

2.2.2 Filtering

As seen in the dosing analysis, the weighing bin experiences significant oscillations due to the chambers in the screw conveyor transporting the raw material onto the weighing bin. Additionally, some of the smaller weighing bins experience considerable disturbances which, by itself, could benefit from a filter.

If the derivative of the weight were to be used by a Multi-State controller, the oscillations caused by the screw chamber would most likely create an unstable controller. Through a Fourier analysis, the harmonics of the noise and the process are found. Based on the resulting analysis, a cutoff frequency is determined.

Low Pass Filter

A Low Pass Filter with the correct cutoff frequency should be able to attenuate noise in the weight derivative estimation. Low pass filters are commonly used, which makes it easy to implement, and is in this case provided pre-coded in the PLC. A potential challenge when employing a low pass filter is how it handles disturbances; the filter might need a significantly low cutoff frequency. This would in turn, make the response of changes in the screw derivative sluggish, which could be problematic if the weight derivative were to be used in control.
A low pass filter is designed with a cutoff frequency found through Fourier analysis of the system, before being tested on a number of dosing screws.

Kalman Filter

If inadequate, a Kalman filter is developed. The Kalman filter uses a model of the system to predict the next step and uses this together with the measurement to find an estimate that balances the predicted and measured value. With the already established model, predicting the weight derivative, the Kalman filter helps predict both the weight derivative and attenuate disturbances. The Kalman filter is made to filter out white Gaussian noise, which is reflected in the process and observation noise covariance matrix. Finding the true covariance matrix is difficult; initial estimates are instead created, which are then manually tuned based on trial and error.

By looking at the Mean Square Error (MSE), a statistical measure for how well the model is fitted, the MSE from a number of model validations is used for each weighing bin. The MSE is used for the variance in error for both the weight and the weight derivative for predictions. The covariance between the weight and weight difference can be assumed to be positively correlated. An initial covariance is therefore created with the product of the two variances.
For the measurement covariance error matrix, the weight and weight derivative variance is harder to initialize. Empirical methods are used to find the variance. By looking at dosing data from when the weighing bin's experience disturbances, the estimated model is used to predict the weight based on the screw input. Over shorter periods this is functioning as the real state compared to the measured value experiencing disturbances. The residual between the prediction and measurement is then used to find the variance. Over the same period the derivative is found for both the measurement and the prediction, this is then used to find the variance in the error for the derivative. Both the covariance matrices found are initial guesses. Through trial and error, the covariance matrices are adjusted to end up with the desired behavior. The intricacies of the Kalman filter can be understood through [6].

The Kalman filter equations are given by:

$$\begin{aligned}
\hat{x}_{k|k-1} &= A\hat{x}_{k-1|k-1} + Bu_k \\
P_{k|k-1} &= AP_{k-1|k-1}A^T + Q \\
K_k &= P_{k|k-1}H^T(HP_{k|k-1}H^T + R)^{-1} \\
\hat{x}_{k|k} &= \hat{x}_{k|k-1} + K_k(z_k - H\hat{x}_{k|k-1}) \\
P_{k|k} &= (I - K_kH)P_{k|k-1}
\end{aligned}$$

where:

- $\hat{x}_{k|k-1}$, $\hat{x}_{k|k}$, A, B are already defined in the subchapter for state space formulation.

- $\hat{x}_{k|k-1}$ is the a priori state estimate,
- $\hat{x}_{k|k}$ is the a posteriori state estimate,
- $P_{k|k-1}$ is the a priori estimate error covariance,
- $P_{k|k}$ is the a posteriori estimate error covariance,
- K_k is the Kalman gain,
- A is the state transition model,
- B is the control input model,
- H is the measurement model, defined as $H = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$,
- Q is the prediction noise covariance, defined as: $Q = \begin{bmatrix} \mathrm{V}_w & V_{\mathrm{cov}} \\ V_{cov} & V_{\Delta w} \end{bmatrix}$
  Initial:

  $V_w = V_{\Delta w} = \mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(\Delta w_i - \Delta\hat{w}_i)^2$
  $V_{\mathrm{cov}} = V_w V_{\Delta w}$
  With n being the duration of an entire validation dataset

  ,
- R is the measurement noise covariance, defined as: $R = \begin{bmatrix} W_w & W_{cov} \\ W_{cov} & W_{\Delta w} \end{bmatrix}$
  Initial:
  $W_w = \frac{1}{n}\sum_i^n (w_i - \hat{w}_i)^2$
  $W_{\Delta w} = \frac{1}{n}\sum_i^n (\Delta w_i - \Delta\hat{w}_i)^2$
  $W_{\mathrm{cov}} = W_w W_{\Delta w}$
  With ***i-n*** being the period over which the process experience disturbances ,
- v_w and $v_{\Delta w}$ is the variance for the error between the prediction and true state,
- w_w and $v_{\Delta w}$ is the variance for the error between the measurement and true state,
- v_{cov} and w_{cov} are covariance of error between variance for each state,
- u_k is the control vector,
- z_k is the actual measurement.

2.3 Multi State Feedback-Controller

A number of the heuristics and considerations added to the P-Controller could be replaced by having a separate controller for the material flow. By adding a controller based on the flow of material, material flow towards the end of a dosing run is ensured if feasible. This is done by having an integral controller based on the flow and a flow set point. Figure 2.1 shows a flow chart of the process, without any additional heuristics or considerations.

When dosing, it's desirable to maximize the material flow rate until the end of the dosing run, where a slower material flow is wanted. This is to minimize the duration of dosing while avoiding overshoots towards the end. This could conflict with having a static setpoint for the flow and needs to be addressed. The stability of a controller using a derivative is also a concern, especially if the weighing bins are subject to disturbances.

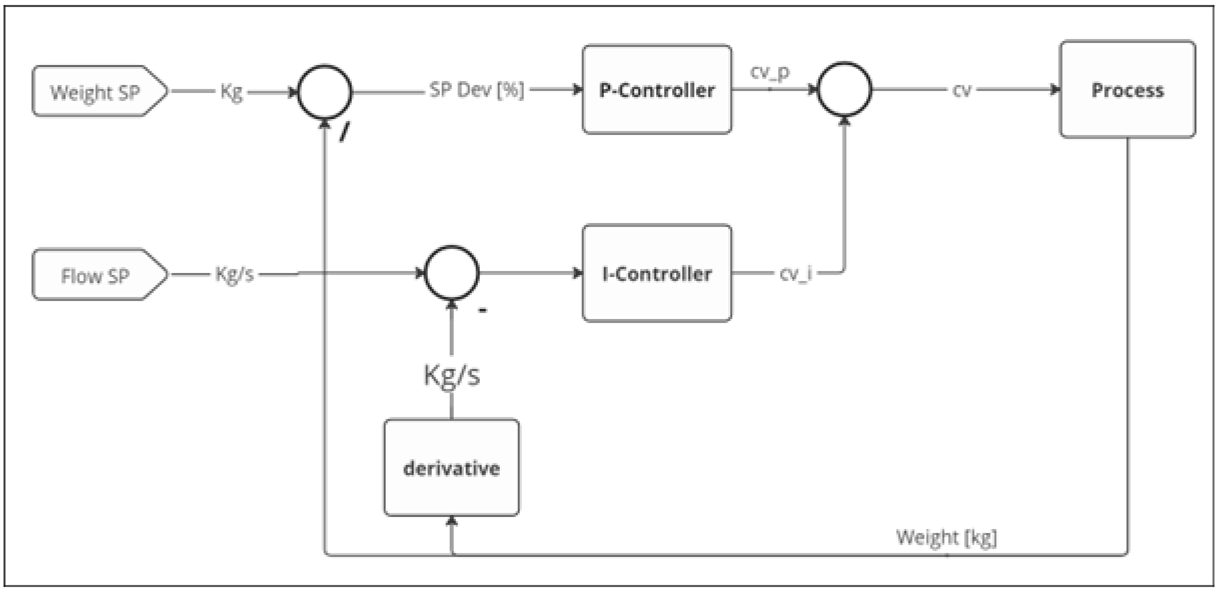


Figure 2.1: Multi-State Feedback control flow diagram

Heuristics and Considerations

To account for disturbances in the measured/estimated flow rate, a smaller gain for the integrating part of the flow control is used. This could lead to other problems in the system reacting too slowly. The accumulated variable is therefore initialized before being put to use.
The material flow setpoint should reflect the ideal flow rate towards the end of a dosing run and would therefore be too low for the majority of the dosing run. This is addressed

by activating the integrating controller for the flow rate once the weight reaches a certain percentage of the setpoint. In determining when to stop the screw input, the estimated model uses the current screw speed to forecast the weight a number of time steps into the future. If the future weight reaches a cutoff point defined as a percentage of the setpoint, the controller stops. In this way, the same cutoff point can be applied to all of the dosing units.

Review

Having an additional controller for the flow ensures that the controller delivers raw material to the weighing bin if possible. As for the proportional controller, additional heuristics need to be included to ensure a stable and fast dosing control. This results in a multi-state controller having 7 control parameters compared to 5 control parameters for the proportional controller and the initial controller scheme. The control parameters would have to be found and tuned for each screw, but could prove to be a viable control scheme. With the myriad of parameters added to make the control scheme work, finding a way to omit some of the control parameters or have them auto-tuned is needed to make the control scheme more easily propagate throughout the plant. The multi-state feedback controller formula is given by:

$$cv_i(k) = K_i \sum_{0}^{k} e_i(k) \tag{2.8}$$

$$cv_i(k) \begin{cases} 0 & \text{if } cv_i(k) < 0 \\ cv_i(k) & \text{otherwise} \end{cases} \tag{2.9}$$

$$cv(k) \begin{cases} cv_p(k) + cv_i(k) & \text{if } \frac{w(k)}{sp} < \text{cutoff} \\ 0 & \text{if } \frac{w(k+5)}{sp} => \text{cutoff} \end{cases} \tag{2.10}$$

where:

- $cv(k)$ is the control output, with $cv_p(k)$ and $cv_i(k)$ being control signals from proportional and integrating derivative respectively,
- K_i is the integrating gain for flow sp,
- $e_i(k)$ is the error signal for flow rate, defined as the difference between the desired setpoint and the actual flow rate of material,
- ”cutoff” is the deviation in % for the weight and setpoint.

2.4 Non-Linear gain functions for Multi-State Controller

Trying to reduce the number of control variables and make the control dynamics more favorable. Different functions for gain could be used both for the proportional weight control, as well as the integrating flow rate control.
By using the deviation from the setpoint in % instead of kg for the proportional part of the controller, initial control parameters become independent of a weighing bins range. The deviation in % is then used inside a hyperbolic tangent function (tanh) with K_p multiplied inside the function. The resulting output is a contribution to the screw speed in % ranging from 0 to 100. Figure 2.2 shows the resulting control loop diagram for the given multi-state feedback controller. By tuning K_p, its possible to adjust the sharpness of the decrease, whith more quadratic characteristics for the curve of the controller. This ties nicely in with controller having a reduced speeed towards the end to attenuate non-linearities in the dosing system.

Having a flow setpoint is needed to make sure the dosing controller finishes a batch. With a static flow setpoint, the flow would need to dip below for it to contribute to the screw speed, as the proportional controller ensure a high enough screw speed for the majority of the dosing run. This could be addressed by having a flow setpoint that changes according to the weight deviation from setpoint. With an exponential function for the flow setpoint, a constant like e is raised to the power of the weight deviation in percent decimal. This would give a high setpoint that makes the integrating flow controller contribute more for the transition when the proportional control gets negligible. Towards the end of the run, the deviation from setpoint will get smaller, the flow setpoint will then in turn converge towards the initial setpoint. The integrating coefficient K_i is then used to determine how much of the flow deviation that is to be accumulated for the control signal.

Heuristics and considerations

Some heuristics are added to the controller scheme to increase the robustness of the system. The integral flow controller is constrained to activate once the weight is within 50% of the set point. This ensures that the integrating controller doesn't get too high before the proportional weight controller fades out. An additional heuristic is limiting the control value to 100%. The controller logic is stopped in the same way as for the multi-state controller, with an estimated model used to predict if the weight reaches a cutoff point within a given prediction horizon.

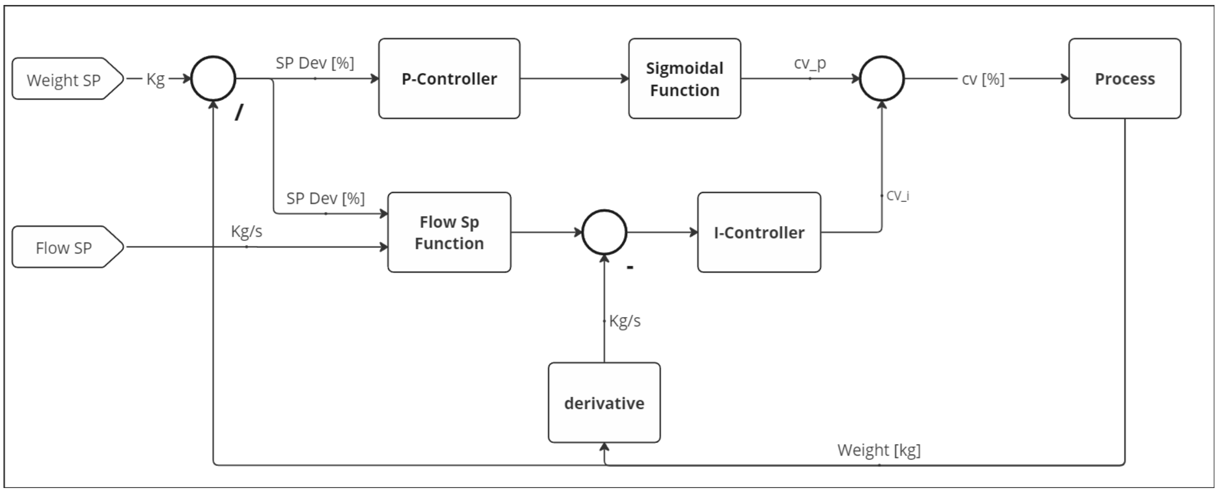


Figure 2.2: Multi-State Feedback controller with Non-Linear gain functions: flow diagram

Review

By allowing the controller to run more continuously, the number of parameters is reduced to only 3. A considerable improvement in the number of parameters, but made more abstract, which could prove harder to tune. However, an abstraction in the control parameters would be trivial if the optimal control parameters were to be estimated. In that case, a more continuous control scheme with fewer control parameters is beneficial. Having non-linear gain functions makes the transition from the proportional controller to the integrating derivative controller smoother, even with a decrease in heuristics and control parameters.

$$cv_p(k) = \tanh(\frac{e}{\text{Sp}} * K_p) \tag{2.11}$$

$$sp_{flow} = sp_{flow_{base}} * \exp^{\frac{e}{sp}} \tag{2.12}$$

$$cv_i(k) = \sum_{k_i}^{k} (1 - \frac{\Delta w}{sp_{\text{flow}}}) K_i \tag{2.13}$$

$$cv_i(k) \begin{cases} 0 & \text{if } cv_i(k) < 0 \\ cv_i(k) & \text{otherwise} \end{cases} \tag{2.14}$$

$$cv(k) = cv_p(k) + cv_i(k) \tag{2.15}$$

$$cv(k)\begin{cases} cv_p(k)+cv_i(k) & \text{if } \frac{w(k)}{sp} < \text{cutoff} \\ 0 & \text{if } \frac{w(k+5)}{sp} => \text{cutoff} \end{cases} \tag{2.16}$$

where:

- $cv(k)$ is the control output, with $cv_p(k)$ and $cv_i(k)$ being control signals from proportional and integrating derivative respectively,
- K_i is the integrating gain for flow sp,
- $e_i(k)$ is the error signal for flow rate, defined as the difference between the desired setpoint and the actual flow rate of material,
- e is the error between the weight setpoint and the weight,
- sp is the Set point for the weight,
- sp_{flow} is the set point for the flow,
- $sp_{flow_{base}}$ is the base flow set point
- ”cutoff” is the deviation in % for the weight and setpoint.

2.5 Auto Tuning

In reducing the number of control parameters for a new control scheme, it could be sufficient to have a set of control parameters for each of the weighing bins. However, each dosing units are dosing different amounts. With each of the silos containing different materials with different characteristics, which are delivered using screw conveyors that also posess different characteristics. Additionally, the screw and raw material characteristics will change over time. It’s clear that each of the dosing units should have bespoke control parameters for the best possible performance.

By defining a performance index, an optimization algorithm could be used to find the optimal control parameters for each of the dosing units. The already established model would be used together with the Multi-state Controller with non-linear gain functions to calculate the performance for the given controller parameters.

Performance Index / Cost Function

Creating the correct performance index (cost function) is crucial in finding the best control parameters for each dosing screw. The performance index penalizes integral squared errors in % for the setpoint, and integral squared controller change. The screw input is always in the range of 0% to 100%, for this reason, the errors in the weight are also expressed in % based on the final setpoint. This makes the ratio between the two components of the performance index independent of the dosing size.

Optimization algorithm

For the multi-state controler with non-linear gain functions there are three parameters to tune, the proportional gain, the flow setpoint in % of the maximum flow for the specific dosing unit, and the integrating gain for the flow. With the control algorithm heuristics and unconventional control structure it's unclear what the optimal optimization algorithm is. The use of nature-inspired algorithms to find the optimal control parameters for a PID-controller have been established as a solution to the problem [7]. But would require some effort to build. With python, SciPy provides a number of optimization techniques which could be tried. Brute Force could also be applied in finding the optimal control parameters. The final optimization algorithm needs to handle bounds for the controller parameters, and be fast enough for it to be applicable in an industrial setting. Scipy's function Optimize is therefore used, with a number of different optimization algorithms tested. Utilizing SciPy should be asily configured to address the efficiency, as well as the bounds for the given problem.

Bounds

The parameters needs to be bounded to ensure the estimated optimal control parameters results in a stable controller. The different units for the bounds create an uneven search space for an optimization algorithm to go through. This is handled by normalizing the control parameters with the bounds being the upper and lower limit when normalizing. The parameters are then reverted back from normalization before they are used by the control algorithm. The bounds themselves and the control parameters initial values are found through trials with a dosing simulation, the resulting control parameters are applied on the dosing model. With the bounds being tuned based on the results provided by plots.

The resulting control solution ends up only having 1 parameter for an operator to tune. That is the balance between efficiency and accuracy when dosing, a significant improvement, in both the number of parameters, as well as how apprehendable the single parameter is utilized.

Control parameters:

Table 2.1: Control parameters to tune

Variable	Description
K_p	Proportional gain
sp_{flow}	Flow Setpoint in % of max flow
K_i	Integrating gain

Performance Index:

$$\Delta cv(k) = \begin{cases} cv(k) - cv(k-1) & \text{if } cv(k) < r \\ 0 & \text{otherwise} \end{cases} \tag{2.17}$$

$$J = \sum_{k=0}^{Np} ((1 - \frac{e}{sp})10)^2 Qe + \sum_{k=0}^{Np} \Delta cv(k)^2 Pu \tag{2.18}$$

Where:

- Qe is the coefficient for integral squared errors for the weight.
- Pu is the coefficient for integral squared controller change.
- Np is the prediction horizon for a dosing run.

3 Results

In developing a new control system, Python is used. Both as a tool for performing analysis, but also for establishing and testing new control algorithms. Figure 3.1 shows a diagram of how the new components used for development are incorporated into the existing controller setup. All of the objects (motors, valves, etc.) in a plant are connected to a single PLC. These are accessed by Python through an API for the PLC, enabling a Python script to read and write to all of the tags in the PLC, giving Python access to the entire plant.

A configuration file written in JSON format is used. The config file contains PLC tags, the JSON file contains a structure that reflects the site and its mechanical dosing configuration. The config file is then used both in analysis, control, and simulation. When using Python to control the system, the script uses tag information from the config file to read and write to the PLC.

In analysis, a Python script uses tag info from the config file to read process values before storing them to a CSV file. The CSV file is then used to perform analysis or simulate the dosing system for further development. Each of the control methods are tested live on the site, with actual product orders while the plant is producing.

An increase in accuracy, a decrease in alarms, and a decrease in control parameters are used to evaluate the results that are generated. This is then used to determine how applicable potential dosing schemes are.

Sampling time

Each of the following developed Python scripts utilizes a sampling time to maintain a certain rate of read and write to the PLC. A sampling time is also used when dealing with discretized data in model estimation, utilizing the model, and calculating the rate of change for the weight. Based on the Nyquist-Shannon sampling theorem, the sampling should be at least twice as fast as the fastest component of the system [8]. Figure 3.2 shows a zoomed-in plot of the dosing run in Figure 1.12 with a corresponding Fourier analysis. The oscillations, seen between 20 and 60 seconds, are thought to be caused by the chambers inside the screw conveyor delivering the product. This is determined as the fastest system frequency, which gives off a peak around 2.2 Hz. The data was logged

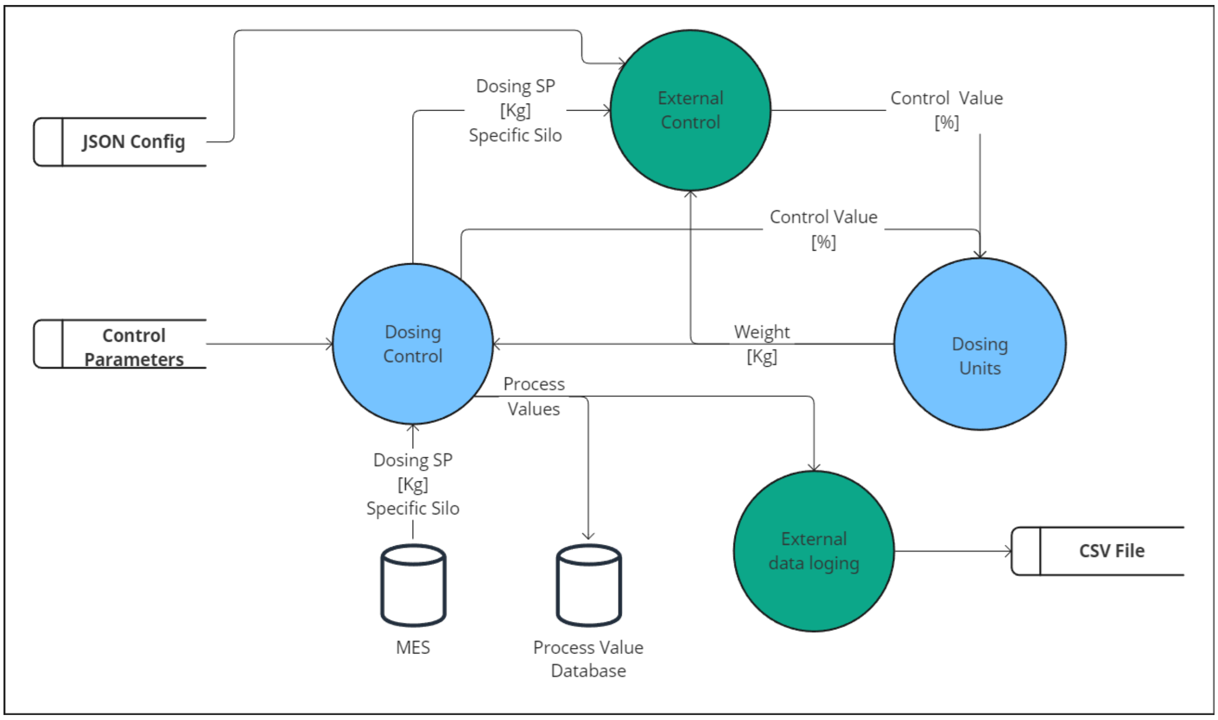


Figure 3.1: Data flow for new control scheme

with a 5 Hz frequency, which has been proven to be adequate for the remaining dosing controller development.

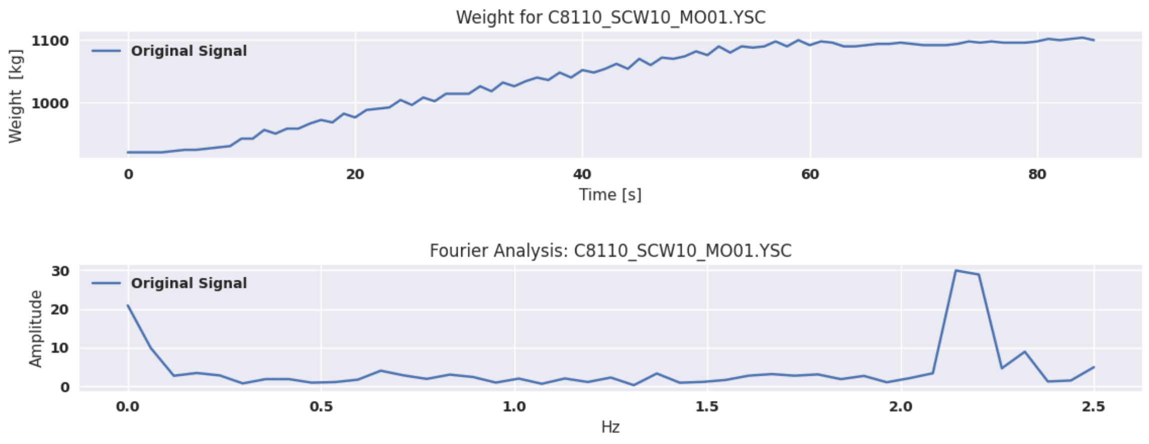


Figure 3.2: Fourier Analysis for finding sampling rate

3.1 P-Controller

P-controller logic and heuristics are transferred to a Python script which controls the dosing. A Python log file is also used to write the screw speeds and weight to a CSV file. The CSV file is then used to create plots which are used to interpret the results from the dosing trial.
The results from three different dosing screws have been included in figures 3.3, 3.4, and 3.5, each of them highlighting different shortcomings (the remaining figures with P-control can be viewed in the Appendix). Figure 3.3 with dosing for screw 03 shows the problem of having a common base speed for all of the screws. With the screw running at 15%, there is no substantial amount being transported to the weighing bin as opposed to screw 5 in 3.5. This resulted in alarms generated by too low flow. In this case, the P-controller got stuck, and the dosing had to momentarily be reverted back to the old system to be able to deliver the final amount of raw material. As seen by the two spikes of 30% screw speed at the end. Figure 3.4 together with figure 3.5 also shows another problem with the P-controller. That being the dosing time. When the screw speeds are decreasing, the time it takes to dose the final amount is too long. This could introduce bottlenecks, resulting in worsened plant efficiency. Figure 3.4 differs from 3.3 and 3.5 in that a smaller amount is being dosed. There's a coefficient in place that's reducing the initial speed when the target weight is smaller. This clearly needs to be tweaked in order to reduce the nonlinearities of the system. With a dosing time of only 6-8 seconds, disturbances on the scale and inconsistencies could have too big of an impact.

A P-Controller is clearly not fit as a final solution for the system; there's no decrease in the number of control parameters. The only noticeable improvement compared to the existing solution would be the more gradual decrease in screw speed, which ensures a decrease in the potential nonlinearities of the system but is, in this case, implemented on the expense of the system's efficiency. The P-controller could probably be tuned to work just as well as the existing solution, but it would defeat one of the reasons for implementing a new controller, as there's no clear reduction (if not an increase) in the number of parameters used. Each of the screws would need to be tuned, and constantly so, to accommodate changes to the dosing equipment and raw materials.

3.2 Model Estimation

The same Python logging script used to create CSV files for P-Controller results is used in model development. The data from the CSV file is loaded into Python and altered. The dependent dataset (Y) is created with the weight difference for several dosing runs. Independent data is extracted from the corresponding screw speeds. The initial model estimation is performed on several batches of material dosing. This leads to a poorer fit

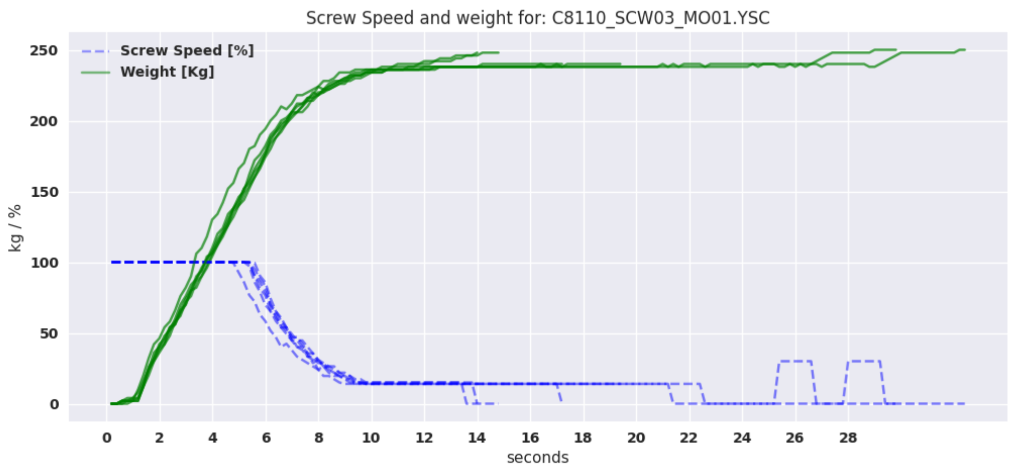


Figure 3.3: P-Control Trial for screw 3

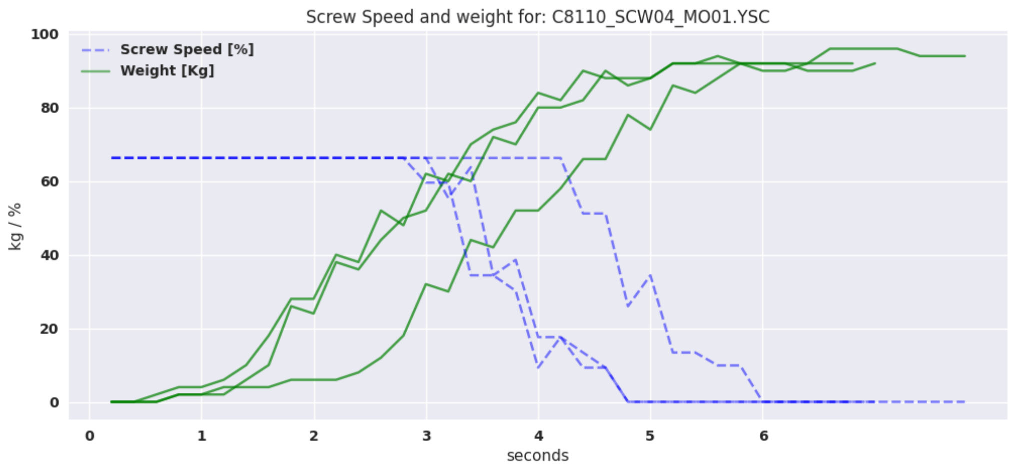


Figure 3.4: P-Control Trial for screw 4

for each screw but ensures that the model structure is general enough as to not get an overfitted model structure once the model is to be applied to specific dosing units.

The model parameters are found by performing ordinary least squares on the dataset, which is then evaluated with R-squared. R-squared is a statistical measure that indicates how much of the variance in the weight change is explained by the independent variables. R-squared is used instead of Root Mean Squared Error (RMSE) because of the units. The model validation is performed on several screw runs, each of them with different weight

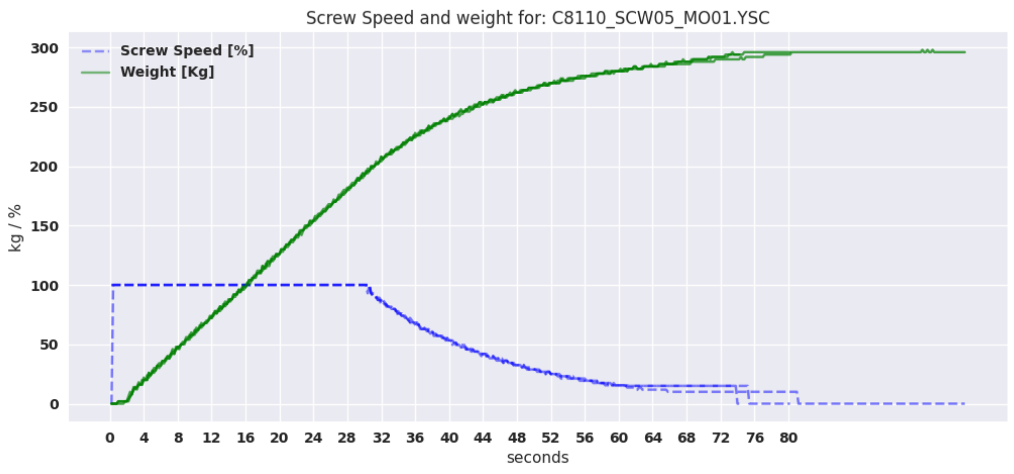


Figure 3.5: P-Control Trial for screw 5

ranges. RMSE keeps the units in the calculation and would therefore not be correct to use in this case when comparing the results for specific screw runs.

With the weight difference being considerably noisy, a better visualization of the model fit is by plotting the weight over time instead, as seen in Figure 3.6. The plot is made by accumulating the weight difference over each time step. When a new screw starts to dose, the accumulated value gets reset.

Knowing that there are time delays in the transportation of raw material through the chambers of the screw conveyor, lagged inputs are added. The results of which can be seen in Table 3.2. When increasing the lagged variables, the R-squared score gets incrementally better. A lag of 6 time steps is chosen as ideal. After 6 time steps of lag, the improvement in the R-squared score increases only marginally. Another reason to not include any further lags is to not increase the risk of overfitting the model once it is going to be applied to each specific screw.

Non-Linear control inputs

Looking at screw 3 in Figure 3.3, it is clear that there are subtle nonlinearities that a model with a linear relationship between speed and weight change is unable to pick up. During the Proportional control trials, the weight difference halted when the screw speed reached around 10%. This can also be seen when looking at the OLS for 6 lagged control inputs for screw 10 in Figure 3.6 (The remaining single OLS validations can be viewed in the appendix). The considerable overshoot in the weight seems to be caused by the fact that the model is trying to find a compromise between the two control speeds that affect the weight difference.

These nonlinearities need to be accommodated for in order to be able to explain more of the variance in the dependent variables. This is done by adding exponential versions of the screw input to the model input. The exponentials of $f_1 = cv^{3/2}$ and $f_2 = cv^{1/2}$ are added. The results from adding these control inputs can be seen in Table 3.1. The R-Squared score is considerably better by adding either one of the nonlinear terms by themselves. To accommodate more continuous control data by a new controller, both of the nonlinear input terms are kept for the final model.

Table 3.1: OLS results

	R-Squared
Lag-0	0.438
Lag-1	0.438
Lag-2	0.457
Lag-3	0.465
Lag-4	0.472
Lag-6	0.477
Lag-8	0.479
Lag-10	0.480

Table 3.2: OLS results specific screws

	R-Squared mean
Lag-6	0.472
Lag-6-f_1	0.486
Lag-6-f_2	0.486
Lag-6-f_{1-2}	0.486

Sliding window

Sliding window live estimation is tried with the final model with both non-linear terms and a lag of 6 time steps. The results are shown for screw 7 and 17 in Figure 3.7. Looking closely, it's clear that the model is overfitted and ends up including oscillations caused by the chambers within the screw conveyor. This needs to be fixed in order for an estimation of flow to be utilized. Figure 3.8 shows the same model structure with a reduced lag of only 2 time steps. However, it does not capture the initial delay in the transportation of raw materials. Utilizing the model in controller development will reveal if it captures enough of the system characteristics.

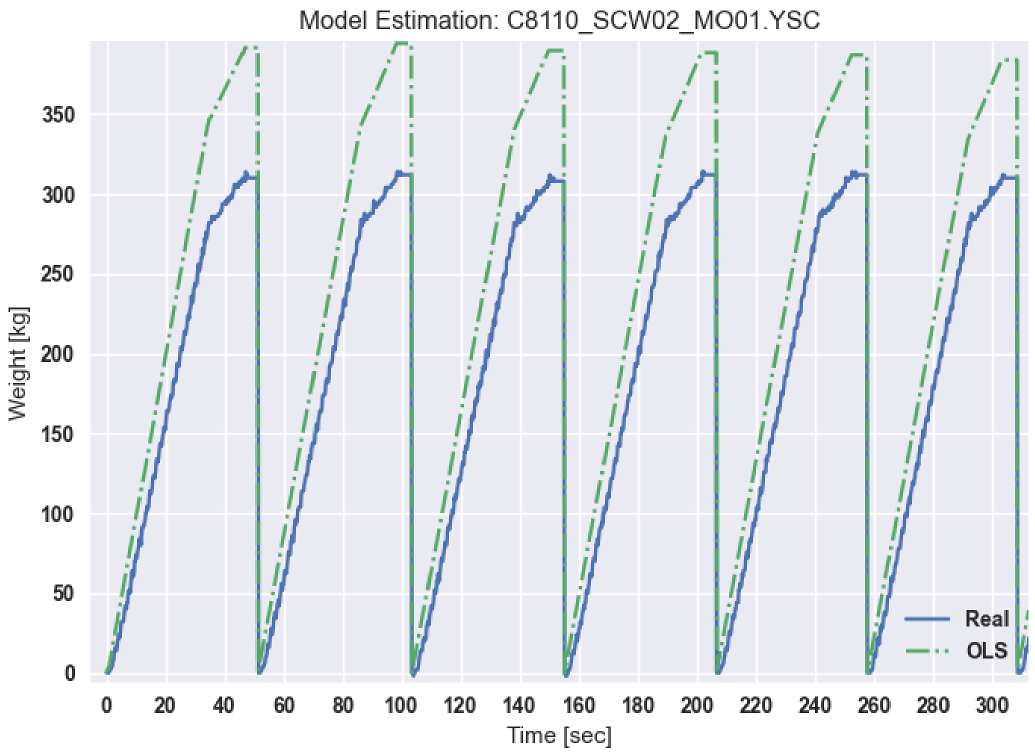


Figure 3.6: OLS-Validation for screw 2

The final model ends up having a lag of 2 time steps for the control input, with two additional non-linear screw inputs added to the system. The numerical values for the model parameters can be viewed in the config file in the appendix, with an array named theta for each model.

The final dosing model is given below:

$$\Delta y(k+1) = b_1u_1(k) + b_2u_2(k) + b_3u_3(k) + b_4u_1(k-1) + b_5u_2(k-1) + b_6u_3(k-1) \tag{3.1}$$

where:

- $u_1 = cv(k)$, $u_2 = f_1(ck(k))$, $u_3 = f_2(cv(k))$

- $x(k)$ is the state vector,

$$x(k) = \begin{bmatrix} w(k) \\ \Delta w(k) \end{bmatrix}$$

- $u(k)$ is the control input vector $\mathbf{u}(k) = \begin{bmatrix} u_1(k) \\ u_2(k) \\ u_3(k) \\ u_1(k-1) \\ u_2(k-1) \\ u_3(k-1) \end{bmatrix}$

This translates into the following state space model components:

- A is the state matrix $A = \begin{bmatrix} 1 & 1 \\ 0 & 0 \end{bmatrix}$
- B is the input matrix

  $B = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ b_1 & b_2 & b_3 & b_4 & b_5 & b_6 \end{bmatrix}$
- C is the output matrix $C = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$

Review

Knowing if the model estimation strategy is successful is hard to tell. Different use cases of the model would pose different requirements for the accuracy of the model. The results from a specific screw in Figure 3.6 look poor. But with the figure showing accumulated weight difference, so would the error in the prediction. The model seems to be accurate enough for a Kalman filter, where an accumulation in the predicted error won't occur due to the inclusion of live weight measurements in the estimated state. Looking closely at the difference between the prediction and the measurement in the general OLS-Validation, overshoots and perturbations are not included in the model estimate, which bodes well for a model-based filter like the Kalman filter. Using the model to estimate optimal control parameters would demand a higher accuracy of the model.

3.3 Filter

If a controller based on the derivative is going to be used, a stable derivative is needed. Figure 3.9 shows the discrete derivative of the weight for an entire batch together with

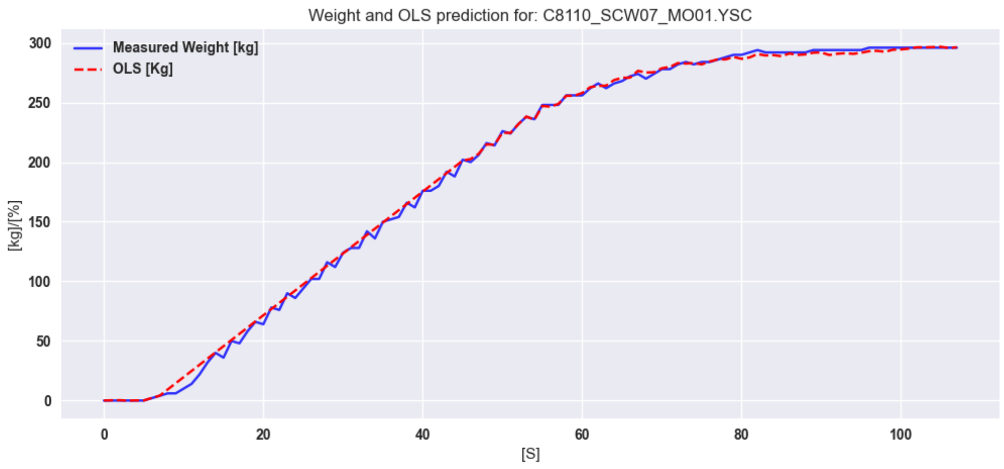


Figure 3.7: OLS for Screw 7

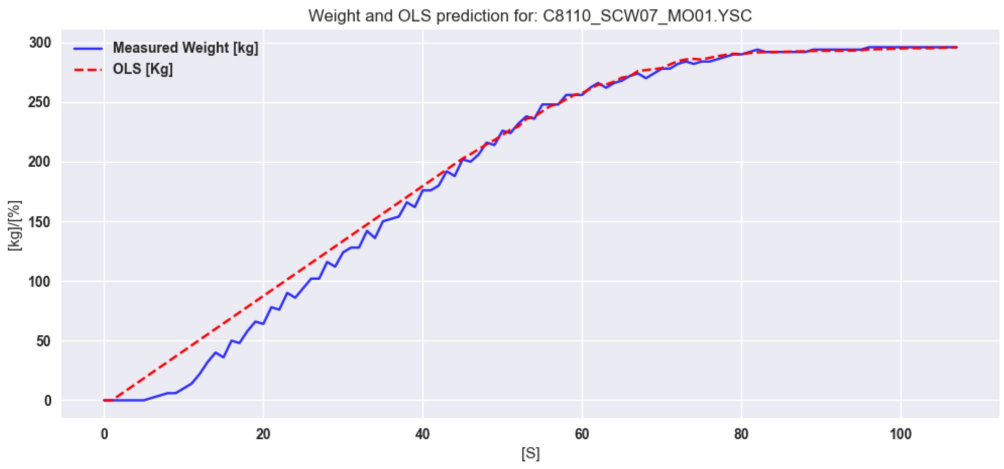


Figure 3.8: OLS for Screw 07 with reduced lag

its corresponding Fourier analysis. It is clear that the raw weight difference is too noisy to be used in any control.

3.3.1 Low Pass filter

For the development of the Low Pass filter, the same CSV data used for model development is used. A Python script is created that performs a Fourier Analysis of the data,

which is then plotted together with the weight difference. From this, the results from the filter can be deduced.
The cutoff frequency can be determined empirically by looking at the previous dosing runs. We want to keep the change in derivative caused by the screw input. This can be found by looking at the duration it takes for the weight derivative to become constant given a constant screw speed. This duration is then transferred to a certain frequency. It takes a dosing run around 1-5 seconds for the weight derivative to become constant, giving a frequency between 1 Hz to 0.2 Hz for the frequency response. Therefore, the cutoff frequency is determined to be 0.5 Hz, which is applied to a low pass filter of the 3rd order. The results of the Low pass filter can be seen in Figure 3.10. The derivative is visibly smoother, and the Fourier transform shows the higher frequencies to be filtered. The Low pass filter is verified by applying it to a portion of the dosing data, as shown in Figure 3.11, and it shows a much smoother weight change, but with a noticeable lag compared to the actual weight.

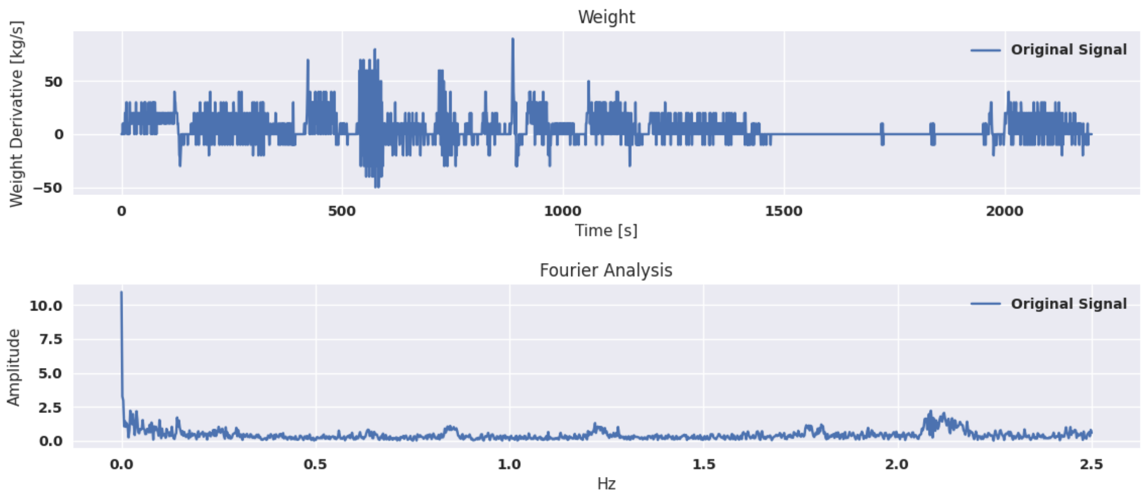


Figure 3.9: Fourier Analysis of dosing data

Figure 3.12 and 3.13 show the filtered weight derivative from the micro dosing scale while it is experiencing disturbances. The usage of a derivative part would create problems for the dosing system in these cases. The LPF is not able to handle disturbances of this nature. A solution could be to decrease the cutoff frequency, but this would make the filtered values even more lagged. The disturbances are, in some cases, problematic enough by themselves, affecting product quality and highlighting the importance of having a filter in place.

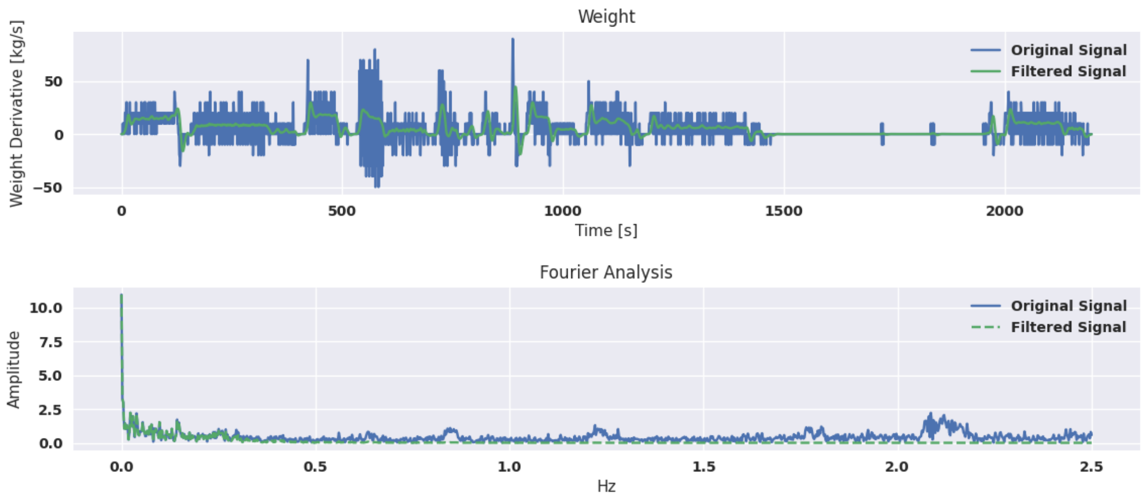


Figure 3.10: Fourier Analysis of filtered dosing data

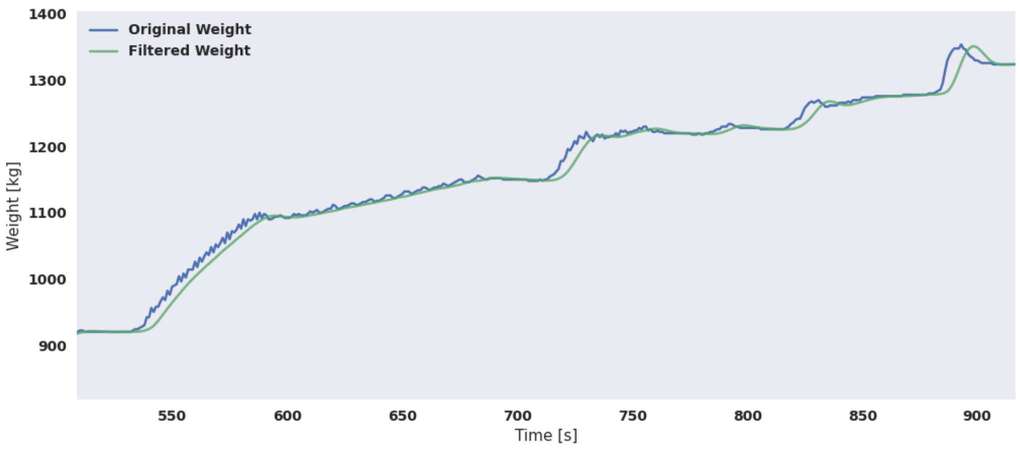


Figure 3.11: Applied Low Pass filter

3.3.2 Kalman filter

In constructing the filter, the already established state space model is used to make the predictions, and the weight from the weighing bins is used as measurements, both for the weight itself, but also in constructing a measured weight derivative. This is put together in a Python script with a Python library providing the Kalman filter algorithm [9]. In order for the filter to function properly, appropriate measurement and model error variances with corresponding covariances need to be found. With the different weighing

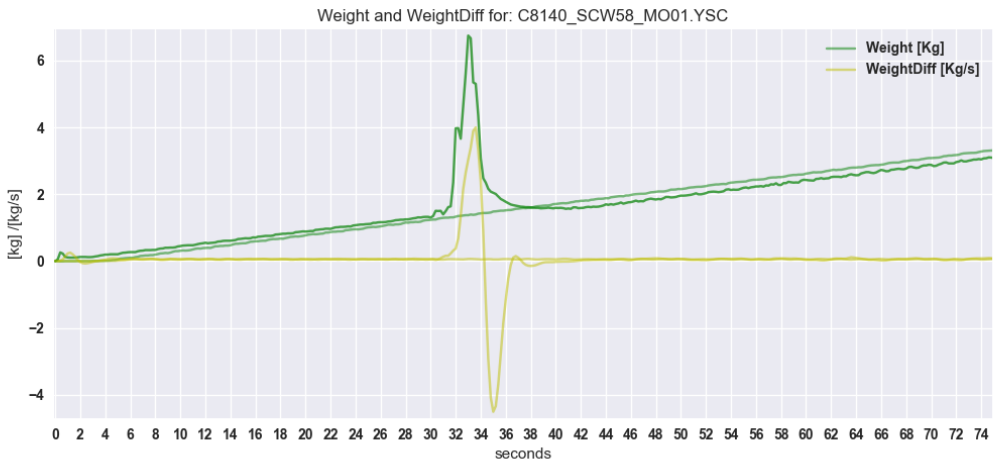


Figure 3.12: Filtered Weight derivative for screw 58

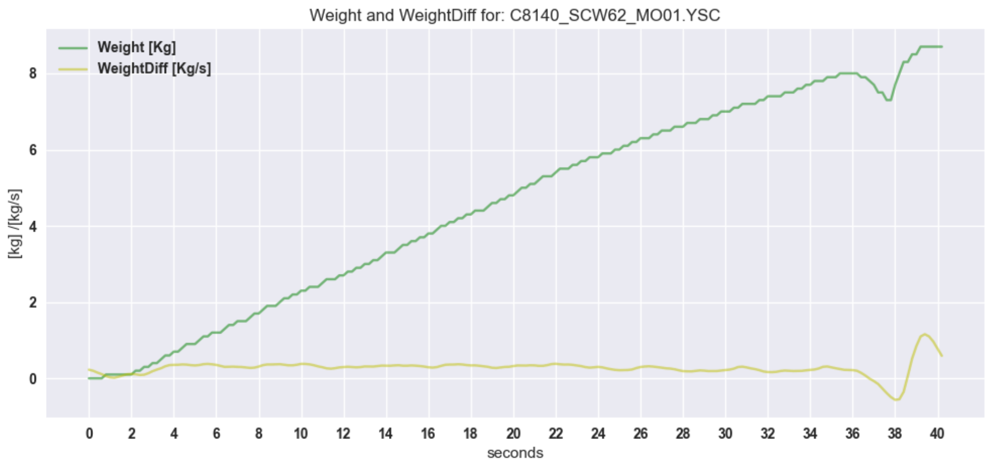


Figure 3.13: Filtered Weight derivative for screw 62

bins having different amounts of noise, each of them needs to have a separate measurement noise covariance matrix.

The Kalman filter assumes the noise to be white, with the noise covariance matrices representing Gaussian distributions. Noise is not the main issue causing production problems, but system disturbances. Disturbances come from external equipment, inducing oscillations or pressure changes.
Based on the methods, the covariance matrices are initialized. From this, the measurement noise and the process noise covariance matrices are manually tuned through trial

and error. The final variances can be seen in Table 3.3.

	W_w	$W_{\Delta w}$	W_{cov}	V_w	$V_{\Delta w}$	V_{cov}
Weigh Bin Primary	9	1	0.01	4	1	1
Weigh Bin 1 Micro	4	1	0.005	0.002	10^{-5}	0.001
Weigh Bin 2 Micro	2	1	0.01	0.02	10^{-4}	0.001

Table 3.3: Variances for the Kalman filter

The results from the Kalman filter can be seen in Figure 3.14 and 3.15 (with the remaining results available in the appendix). In Figure 3.14, the filter is significantly smoother and works quite well in filtering out the oscillations caused by the chambers of material delivered through the screw. Looking at Screw 66 in Figure 3.15, there's a significant disturbance happening around 150 seconds which the Kalman filter is able to significantly attenuate. The Kalman filter works as intended but does not address the mechanical issue that is the source of the problem. Finding the root cause is always preferable. However, the Kalman filter proves to be a solution that could address a number of exogenous perturbations.

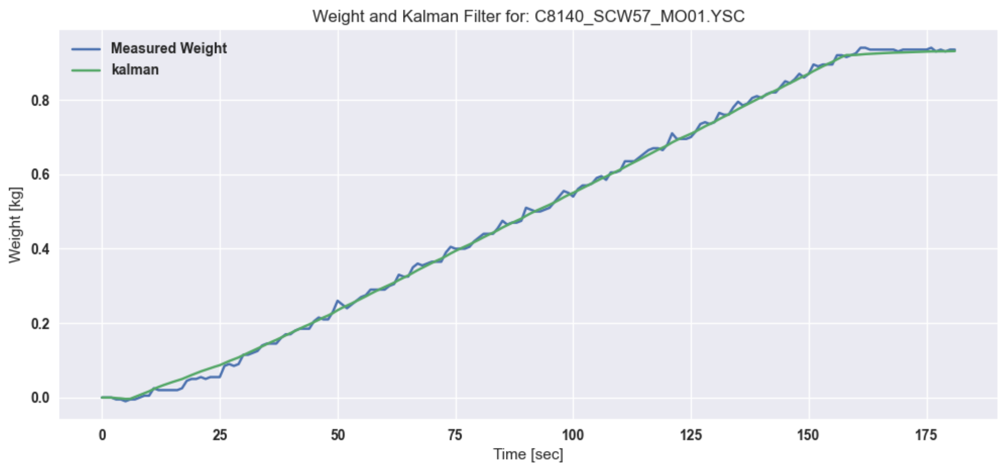


Figure 3.14: Kalman filter for screw 57

3.4 Multi State-Controller

The same procedure is used as for the P-controller. Multi State-controller logic and heuristics are transferred to a Python script. An additional Python script is used to log the results. The already established Kalman filter is employed for the smoother weight state

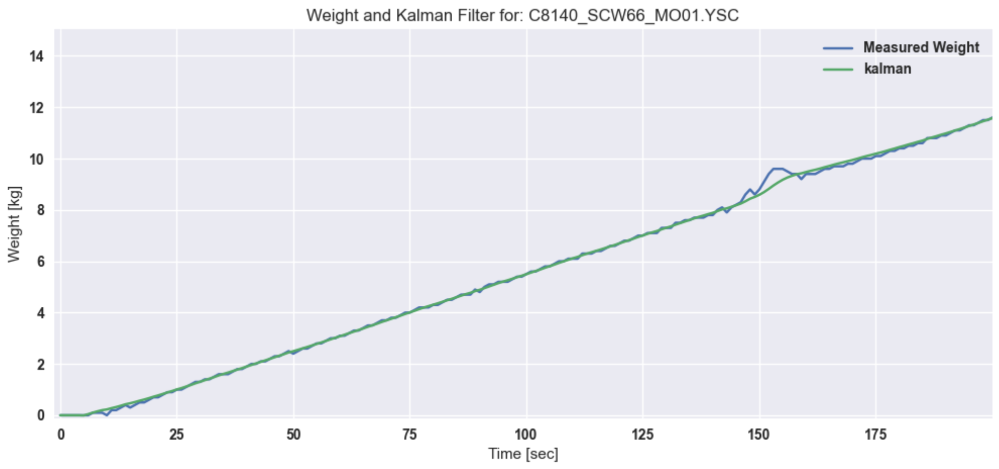


Figure 3.15: Kalman filter for screw 66

and for the weight derivative. The results from the Multi State-Controller are shown in Figure 3.16 and 3.17, and the remaining results are included in the appendix. Figure 3.16 with dosing with screw 9 shows promising results. Within a specified limit, the controller starts and decreases gradually to zero while reaching its setpoint. The controller for screw 9 is clearly working as intended. Figure 3.17 shows dosing with screw 10. Comparing this to Screw 9, it's clear how the differences in raw material characteristics and dosing equipment pose challenges. The dosing setpoint only deviates around 15% between screw 9 and 10, while screw 10 takes almost twice as long in dosing. The significant difference in material flow between dosing screws while using the same Multi State-Control parameters does not work as well in this case. When getting closer to the setpoint, the flow rate for Screw 10 gets too low. There is an additional delay in the rate of change in material dosing, causing the integrating part of the flow controller to rise too sharply at the end of the dosing run. The result of this increase is an overshoot which causes the controller to dose too much.

By maintaining a minimum flow rate through integrating feedback against a flow setpoint, the problem of the material flow stopping once the weight gets too close to the setpoint is addressed. For this to work properly, knowledge of the dosing system's max/min flow rate is needed, or it can be adjusted through trial and error. The issue of the dosing taking too long persists. Ideally, the controller should have a higher dosing speed for longer, followed by a sharper decrease towards the setpoint. This would be adjustable through the parameters established through the heuristic logic, with the degree of success being dependent on the setpoint and current raw material state.

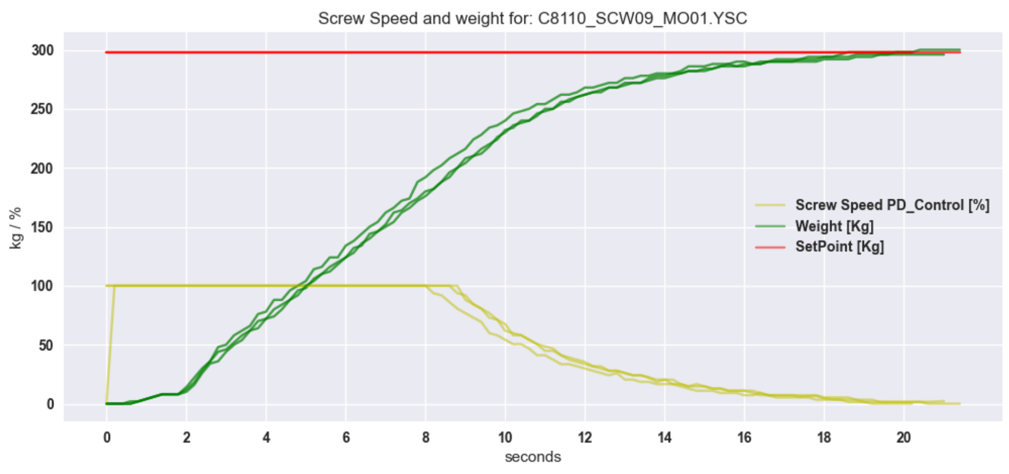


Figure 3.16: Multi State-Control for screw 9

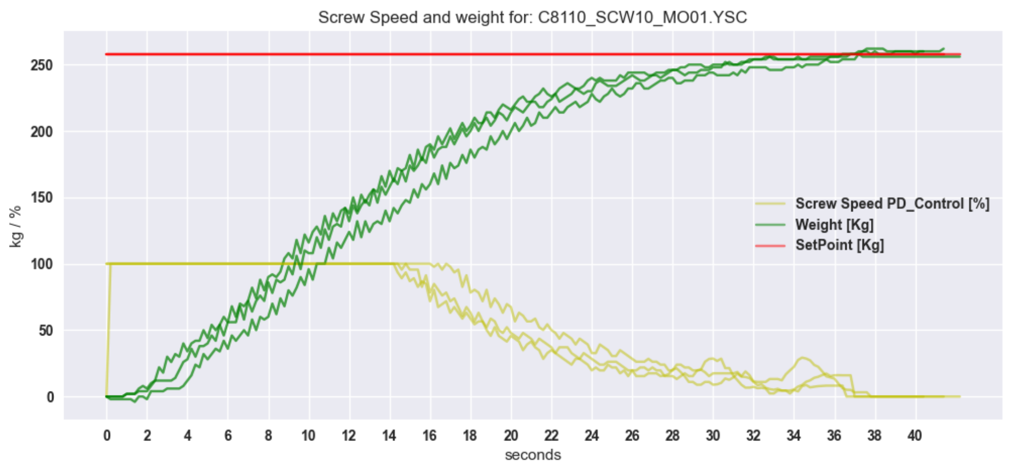


Figure 3.17: Multi State-Control for screw 10

3.5 Performance-Index based Controller tuning

A Python script is developed for finding the optimal control parameters. The already established model estimation scheme and the controller scheme are implemented into the Python script. Both of them are needed to create a performance index for a given set of control parameters. The performance index penalizes deviations from the setpoint, as well as changes made to the control input. In calculating the components for the performance index, the model is simulated together with the given controller scheme. A minimization algorithm provided by SciPy, a Python library, is used to find the optimal control parameters for a given estimated model.

Different optimization algorithms are tested in finding the algorithm most suitable for a multi-state feedback controller with non-linear gain functions. Controller heuristics generates a non-smooth search space for an optimization algorithm, this made it challenging for line-based search algorithms to find a minimum. A trust region search method is found to be the most promising. As its approximating a quadratic representation of the performance indexed within a smaller region.

For the optimization algorithm, there are three independent variables: the proportional gain, the flow setpoint in % of the maximum flow for the specific dosing unit, and the integrating gain for the flow. The maximum flow rate for a specific dosing unit is found by running a 100% screw input through the model over the given lagged duration. In order to reduce the execution time and make the optimization applicable for all of the dosing models, setting appropriate bounds for the independent variables is critical. Control parameters found by the optimization algorithm are used on a simulation with the same model, it shows that a gain up to 20 is necessary for the slowest dosing units to maintain a sharp enough decrease in controller input towards the end. For the flow setpoint, a range of 30% to 100% from the max flow rate is used. The current dosing scheme uses 100% screw speed most of the time for gross dosing and 30% for fine dosing, hence the 30% to 100% range. The final independent variable, the integral gain for the flow, needs to be selected carefully as to not introduce instabilities in the new controller scheme. The integral action is based on the deviation in decimal % for the flow, and the integrating input is also in %, mapping directly to the screw input. By imposing a desired limit on the rate of change for the screw speed of around 2 $\frac{\%}{sec}$ and no less than 0.5 $\frac{\%}{sec}$, the bounds for the integral gain become 0.005 to 0.02. A final overview of the established bounds are seen in Table 3.4.

Control parameters:

Table 3.4: Bounds for parameters

Variable	**Description**	**Upper Bound**	**Lower Bound**	**Initial Value**
K_p	Proportional gain	0	20	10
sp_{flow}	Flow Setpoint in % of max flow	30%	100%	65%
K_i	Integrating gain	0.005	0.02	0.0125

Figures 3.18 and 3.19 show the results from a simulated run based on the same dosing unit model with a ratio of 1 and $\frac{1}{100}$ respectively, against state deviation and control input change for the performance index. The optimization works as intended. Based on a priori dosing data that constructs a model, there is only one parameter that needs to

be tuned, which is the ratio between state deviation and control input change. A single performance index ratio variable should work for one weighing bin containing all of the dosing screws. But by having one performance index ratio variable for each of the dosing units, one should be able to tweak the accuracy against the speed for each of them. This would be desirable with different silos containing raw materials of varying costs. Based on Figures 3.18 and 3.19, the method of using a performance index to be minimized in order to find the optimal control parameters seems viable. However, the entire method assumes an estimated model, which includes the most crucial system dynamics.

3.6 Non-linear weighted error for Feedback control

Based on the already established Multi State-controller, non-linear weighted errors are used for the feedback of a Multi State-controller in order to obtain more desirable system dynamics. The use of a sigmoidal weighted error function is used for the proportional controller, while an exponential weighted function is used for the integrating part for the flow setpoint (*explain why).

The Multi State-controller python script is updated with these functions, along with some additional heuristics. In developing the Performance Index based controller tuning, a simulation function is used instead of running the controller live on the plant for simplicity.

The purpose of using non-linear weighted errors is to create more desirable controller dynamics and reduce the number of parameters used in the controller scheme. More desirable system dynamics include being able to run the screw input at 100% for a longer time while still having a smooth ramp down towards the end of the dosing run. This enables a faster dosing time while maintaining a stable controller. At the same time, a minimum flow rate should be maintained to ensure that the controller does not give an input that is too low, causing the dosing run to be unable to finish or take too long.

The results can be seen through the simulations in Figure 3.18 and 3.19, as well as in real dosing testing with correctly configured control parameters in Figure 3.20. Compared to the dosing runs in Figure 3.17 and 3.16, the screw input maintains a higher percentage for a longer time. The transition from going full screw speed until it starts decreasing is also much smoother. The initial Multi State-Controller scheme could probably have been configured to conform to the desired controller dynamics, but it would require more tweaking of the controller parameters. For the initial Multi State-controller, there are 5 parameters to be tuned, while for the non-linear weighted error feedback controller, there are only 3 parameters. This is made possible by the fact that fewer heuristics are needed for achieving the same controller properties. This should make it easier to tune and much faster for a minimization algorithm to find optimal parameters.

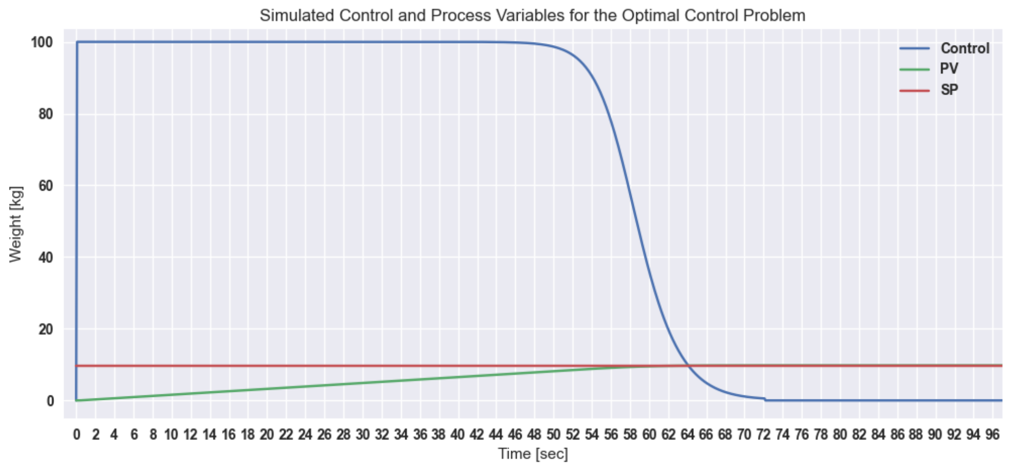


Figure 3.18: Simulated Optimal Fast control, with non-linear weighted error feedback Control

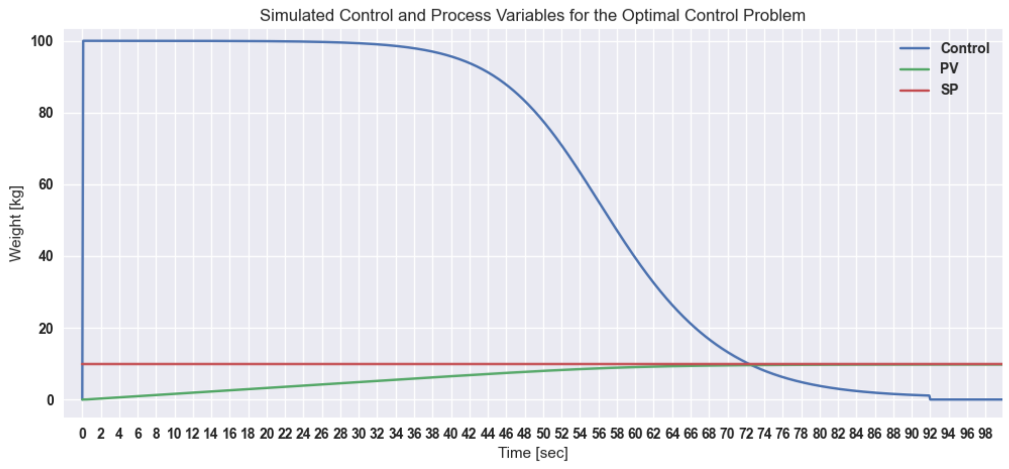


Figure 3.19: Simulated Optimal Slow control, with non-linear weighted error feedback Control

3.7 Fully assembled Control scheme

For the final controller scheme, everything is combined into a single Python program, with the final code included in the appendix together with the corresponding config file. Model estimation, Kalman filtering, Non-linear weighted Multi State-Control, and optimal parameter estimation. Each of these functionalities is contained within a class in a single python script. A second python script is then used to instantiate the classes. With one instance for each of the dosing units. This gives each dosing unit a unique estimated model, which is utilized by the Kalman filter and optimal control estimation. The python program is structured with multithreading, with a single thread handling PLC communication, and one thread for each of the weighing bins in the plant.

Running the fully assembled controller scheme, the estimated model parameters should converge towards an optimal estimate. Converged model parameters should provide an adequate system representation. The Kalman filter and Optimal control estimation should then work without any manual adjustments for each of the dosing units. With the only parameter available to tune being the accuracy/speed of the dosing. The program is run in the background with the initial controller scheme running as usual. This gives the new controller scheme an opportunity to gather data for an initial model estimation.

With an initial model in place, the new controller is tested live, with results seen in figure 3.20 and 3.21 from screw 7 and 18 respectively. The remaining results are included in the appendix. Screw 7 shows being dosed, the initial model estimation results in a dosing run that decreases too abruptly. For the following dosing runs, past dosing data is included, resulting in a dosing run that seemingly converges towards an optimal solution. Screw 18 in 3.21 does not convey the same convergence for optimal control as for screw 7. This could be due to the short dosing duration, which provides a limited amount of data for model estimation. For each of the dosing runs, the screw speeds change smoothly, which cannot be said for the screw speeds in 3.3, 3.4, and 3.5 for the initial Multi State-controller trial, which was carried out without the use of a Kalman filter. It's clear the Kalman filter provides a benefit when used together with the new controller scheme, this is also clearly seen in 3.22, where the last trial for screw 7 is shown.

For the model to be utilized properly, the dosing application needs to adhere to the specified run time of 5 Hz (200 ms). This is verified by logging the running duration for the data thread, as well as the controller thread. The results can be seen in figure 3.24 and 3.23 respectively. It's clear that the current python implementation maintains a consistent execution duration for both threads.

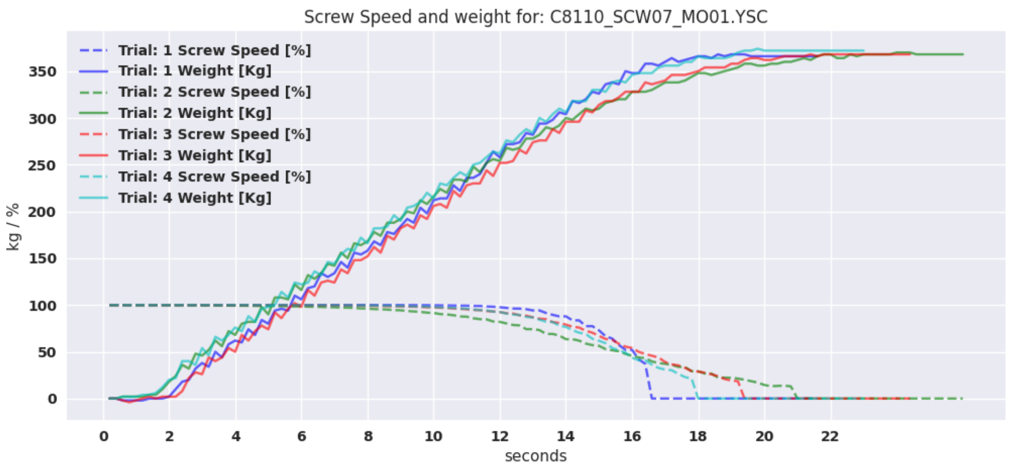


Figure 3.20: Fully assembeled solution tested on screw 7

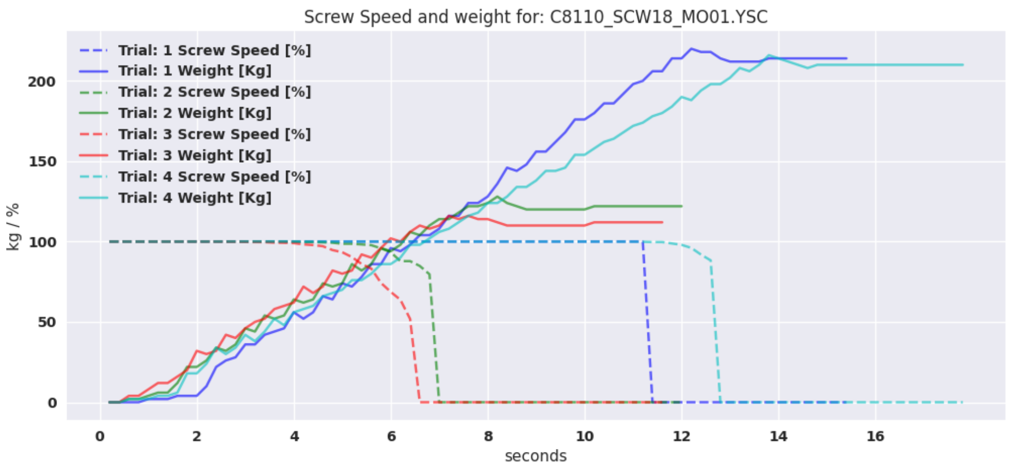


Figure 3.21: Fully assembeled solution tested on screw 18

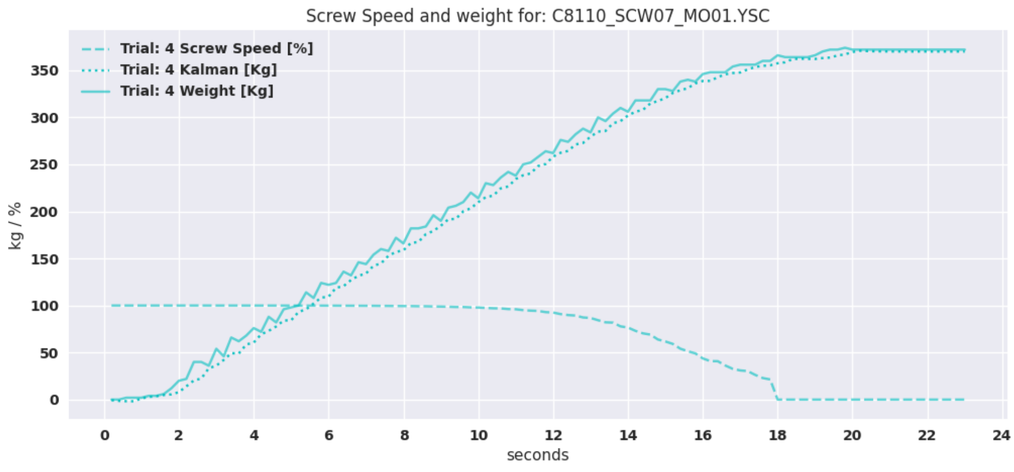


Figure 3.22: Fully assembeled solution Kalman Filter tested on screw 7

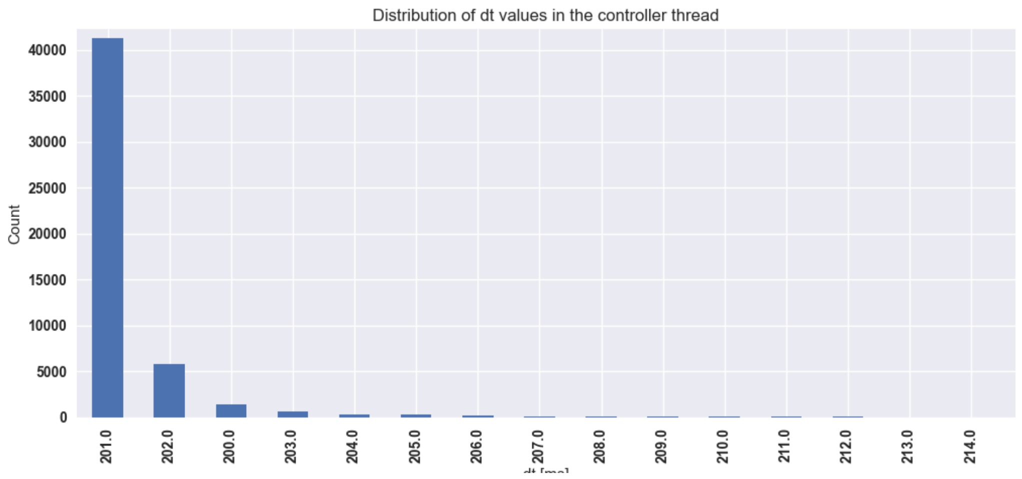


Figure 3.23: Distribution of sampling time for controller thread

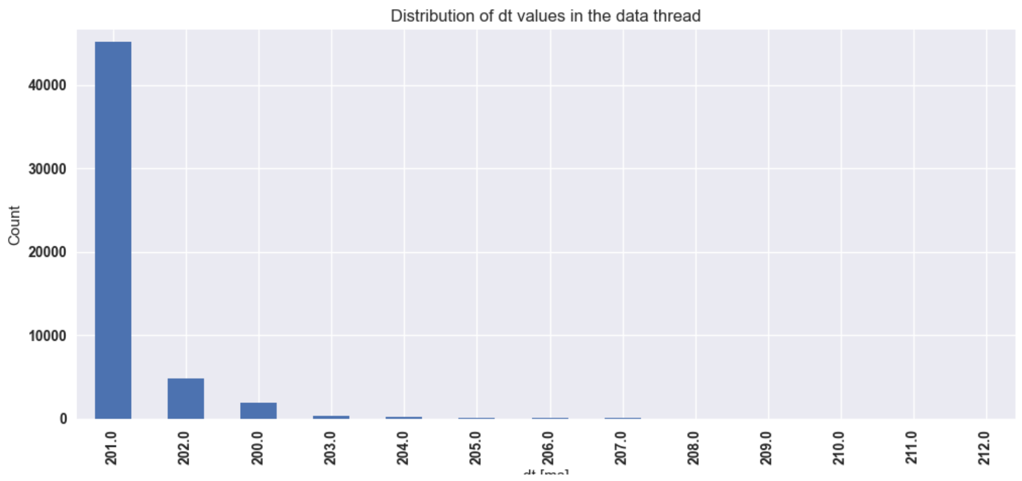


Figure 3.24: Distribution of sampling time for data thread

4 Conclusion

The success of a controller scheme can be determined based of three key performance indicators: the frequency of alarms, the accuracy of the dosing, and the number of control parameters. The final controller scheme has been tested on 7 batches of raw material dosing, making qualitative conclusions difficult for measuring accuracy, as well as for the number of alarms. Still, the final controller scheme addresses each of these issues, and a number of implicit conclusions can be made to determine the success of the final solution.

4.1 Main results

The final controller solution is built in Python and comprises a live model estimator, a Kalman filter, a non-linear weighted error feedback controller, as well as an optimal control parameter estimation routine. The estimated model serves as the foundation for both the Kalman filter and optimal control parameter estimation, making it a weak spot of the entire solution. Without a sufficiently accurate model, the controller does not work. With an accurate model, the Kalman filter is able to filter out oscillations and disturbances, and the optimal control parameters can be found, as shown in Figure 3.20 and 3.21. By minimizing a cost function that penalizes unwanted behavior for the controller, the number of parameters needed to be tuned for each of the dosing units is reduced from 5 to 1 for each of the dosing units. This reduces the number of parameters from around 200 to 37 for all of the dosing units in the French plant. The one parameter that is left, is used to balance the cost of the weight deviating from the setpoint against the changes made to the controller input. As seen in the simulated runs in Figure 3.18 and 3.19, the auto-tuning parameter is used to determine the importance of dosing accuracy, which should ensure a more accurate dosing run where it is desirable.
Through a review of the final solution with BioMar, it became clear that the controller scheme has use cases outside of raw material dosing. With the use of a flow controller, the controller scheme could be used to control oil dosing, as well as steam control. The controller scheme could prove particularly useful for oil dosing where a certain flowrate is desired in order to get an even distribution on the pellets.

4.1.1 Dosing Controller implementation

The dosing controller is implemented using Python. A Python API is used to interface with the PLC of the plant. Running at 200 ms, the Python script manages to respond fast enough and control the dosing. However, using Python to control an industrial process is unconventional, as it introduces additional breakpoints in the process. It is crucial to minimize these breakpoints to maintain the highest possible uptime. The computer hosting the Python script could shut off, restart, or crash. The communication between the Python script and the PLC is also subject to scrutiny, as it employs a public API through a custom Python library, that is not backed by a major industrial vendor. Additionally, using the Python language increases the risk of failure. Python being a highly dynamic language, the chance of experiencing code crashes increases. Alongside considering the reliability of the implemented solution, there is also the challenge of optimizing the computational time. Figure 4.1 shows the delayed start time from 0 seconds for 4 batches of dosing for screw 7. The delay is caused by the computation of the optimal control parameter estimation. With the algorithm running before each of the dosing runs, taking into account the current estimated model as well as the weight setpoint.

Restructuring the code should address the computational delay. By having a separate thread for solving the optimization problem for each weighing bin, the controller thread can run on its own while the optimization algorithm estimates the optimal control parameters for the next silo. With this in place, one could also go one step further in securing the code and transfer the controller to a C application or into the PLC program itself. Moving the controller thread into the PLC would create a more robust solution by not relying on an external application to work. The Python program would still be needed to run the optimization algorithm for optimal control. Splitting it up like this would decrease the dependency on a stable communication path, as well as having the Python application available at all times.

4.2 Dosing Controller Further improvements

Reviewing the final solution, there are a couple of areas of the final dosing scheme that are susceptible to improvements.

For the muti-state feedback controller, the controller could be simplified by using a more traditional feedback controller for the flow control. A PI-controller would remove the need for an exponential function changing the flow setpoint, which in turn would omit the need for aditional heuristics to increase the stability of the controller. In deciding when to stop the screw speed, the final controller uses the estimated model to see if it exceeds the setpoint within a prediction horizon. Different products could have different amounts of

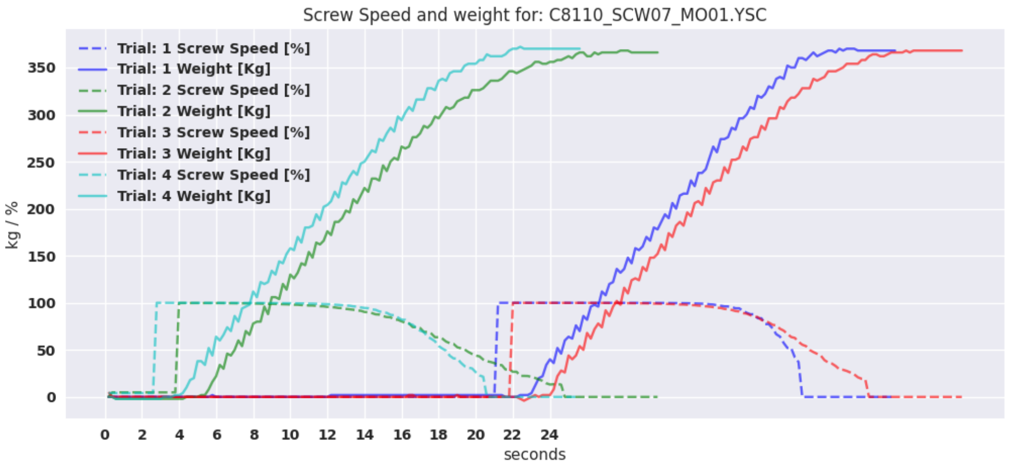


Figure 4.1: Live data for dosing controller running on screw 7

material in-flight once the screw has stopped. The amount of product in-flight could also be dependent on the flow of material, which is not accounted for with the current control scheme. The amount of raw material in-flight is known after a dosing run is finished, this could be used to model the amount in-flight for a given screw speed for each specific raw material.

Overfitting was encountered in the model estimation development process. Instead of addressing the issue by removing complexity, penalized regression could have been used instead [10]. This would ensure that the model doesn't accommodate abrupt changes in the weight difference caused by disturbances and unwanted oscillations. In setting the correct model structure, an ARMAX model could prove beneficial. By including autoregressive states, more of the system dynamics could be included in the model. Additionally, a moving average would attenuate disturbances in the weight derivative even better. With other areas of the plant like oil dosing, possibly making use of the same controller scheme, ARMAX could prove to be a more suitable general model estimation approach. Building upon an ARMAX method, Fourier ARMAX could also prove to be viable. With clear periodic tendencies, which includes unwanted system dynamics, which would then be easier to ignore/dismiss in model estimation. It would also become easier to include longer-term system dynamics, present over several dosing runs.

The Kalman filter could possibly attenuate disturbances even better by including the autoregressive states from the ARMAX filter. The resulting measurement error covariance matrix would then include the relation between consecutive measurement errors. This would make the Kalman gain assign more trust to the prediction once a disturbance has occurred.

In estimating the optimal control parameters, the method is entirely dependent on a sufficiently accurate model. An ILC method could be used to accommodate deviations in the estimated model, based on the combination of MPC and ILC in [11]. After a dosing run, the resulting weight and controller time series data are run through the same performance index function that's used by the optimization algorithm. The optimal performance index is then compared against the actual performance index. Additional weights are then assigned to the components of the performance index. The altered performance index is then going to be used by the optimization algorithm for the next run, which should result in a better match to the actual performance index.

References

[1] Wikipedia contributors, *Manufacturing execution system — Wikipedia, the free encyclopedia*, https://en.wikipedia.org/w/index.php?title=Manufacturing_execution_system&oldid=1199234120, [Online; accessed 29-April-2024], 2024.

[2] K. Åström and T. Hägglund, 'The future of pid control,' *Control Engineering Practice*, vol. 9, no. 11, pp. 1163–1175, 2001, PID Control, ISSN: 0967-0661. DOI: https://doi.org/10.1016/S0967-0661(01)00062-4. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0967066101000624.

[3] A. Besharati Rad, W. L. Lo and K. Tsang, 'Self-tuning pid controller using newton-raphson search method,' *IEEE Transactions on Industrial Electronics*, vol. 44, no. 5, pp. 717–725, 1997. DOI: 10.1109/41.633479.

[4] X. Xu, H. Xie and J. Shi, 'Iterative learning control (ilc) guided reinforcement learning control (rlc) scheme for batch processes,' in *2020 IEEE 9th Data Driven Control and Learning Systems Conference (DDCLS)*, 2020, pp. 241–246. DOI: 10.1109/DDCLS49620.2020.9275065.

[5] 'Model structure detection and parameter estimation,' in *Nonlinear System Identification*. John Wiley & Sons, Ltd, 2013, ch. 3, pp. 61–104, ISBN: 9781118535561. DOI: https://doi.org/10.1002/9781118535561.ch3. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/9781118535561.ch3. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/9781118535561.ch3.

[6] G. Welch and G. Bishop, 'An introduction to the kalman filter,' *Proc. Siggraph Course*, vol. 8, Jan. 2006.

[7] B. Doicin, M. Popescu and C. Patrascioiu, 'Pid controller optimal tuning,' in *2016 8th International Conference on Electronics, Computers and Artificial Intelligence (ECAI)*, 2016, pp. 1–4. DOI: 10.1109/ECAI.2016.7861175.

[8] C. Shannon, 'Communication in the presence of noise,' *Proceedings of the IRE*, vol. 37, no. 1, pp. 10–21, 1949. DOI: 10.1109/JRPROC.1949.232969.

[9] R. R. L. Jr, *Filterpy*, [Online; accessed 16-March-2024], 2022. [Online]. Available: https://github.com/rlabbe/filterpy.

[10] A. Géron, *Hands-On Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. O'Reilly Media, Inc., 2017.

[11] C. Zheng, L. Zhou and F. Li, ‘Two-dimensional model predictive iterative learning control based on just-in-time learning method for batch processes,’ in *2023 IEEE 12th Data Driven Control and Learning Systems Conference (DDCLS)*, 2023, pp. 1353–1358. DOI: 10.1109/DDCLS58216.2023.10166437.

Appendices

4.0.1 Initial Controller

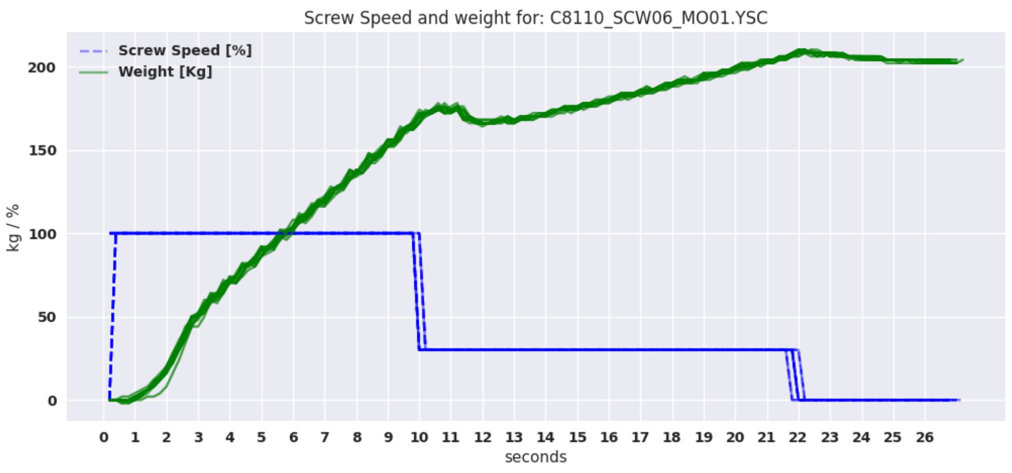


Figure 4.1: Dosing Control for screw 6 with existing control system

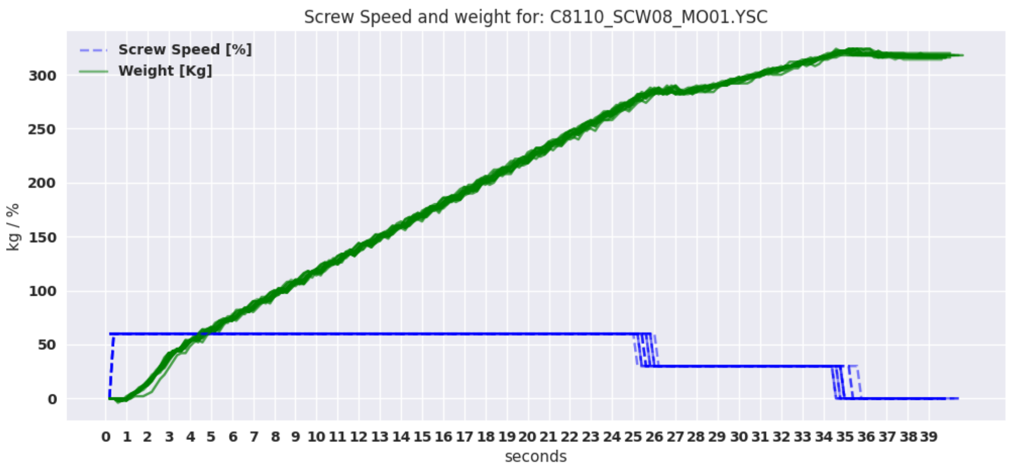


Figure 4.2: Dosing Control for screw 8 with existing control system

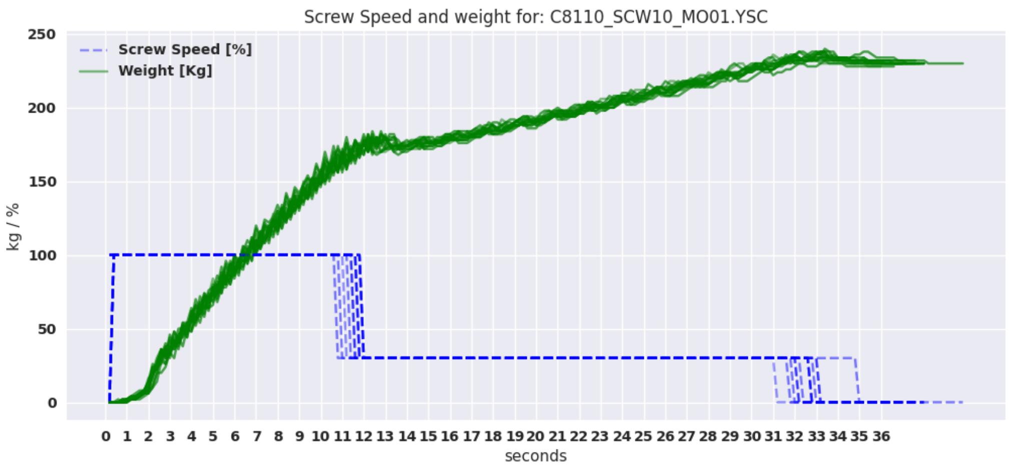


Figure 4.3: Dosing Control for screw 10 with existing control system

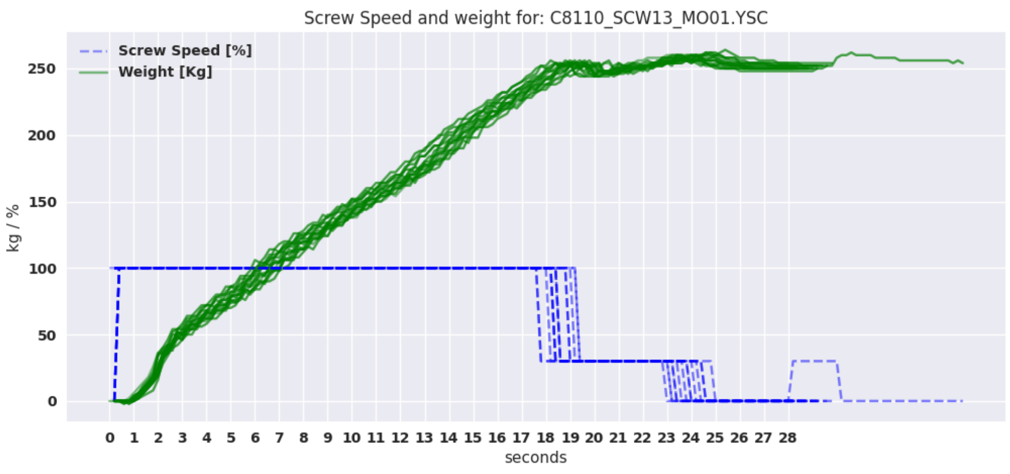


Figure 4.4: Dosing Control for screw 13 with existing control system

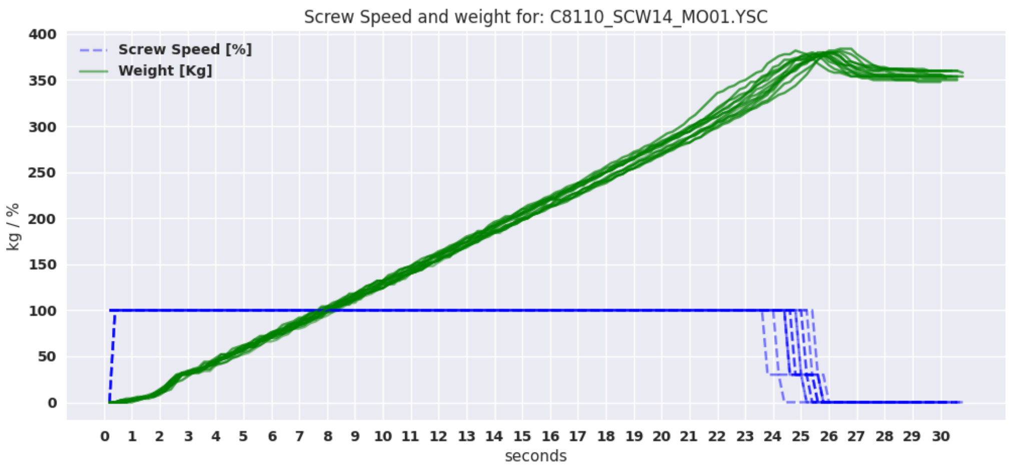
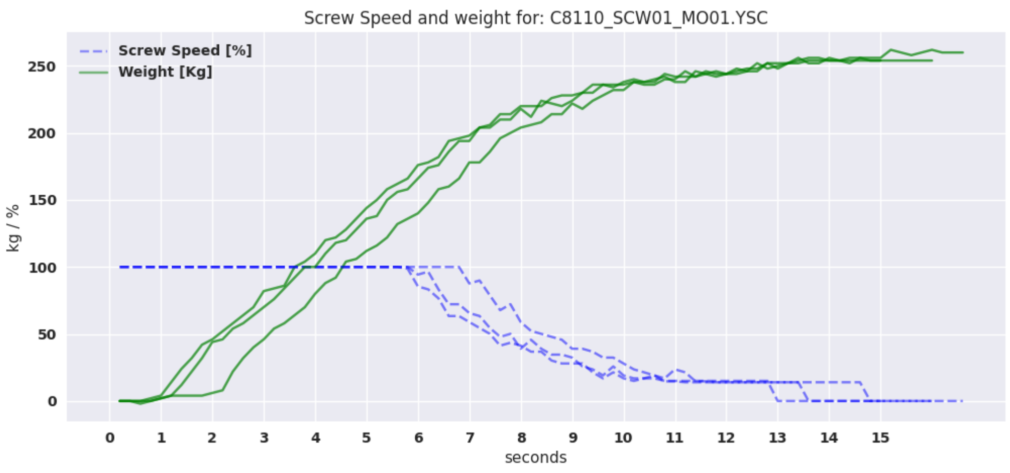


Figure 4.5: Dosing Control for screw 14 with existing control system

4.0.2 P-Controller



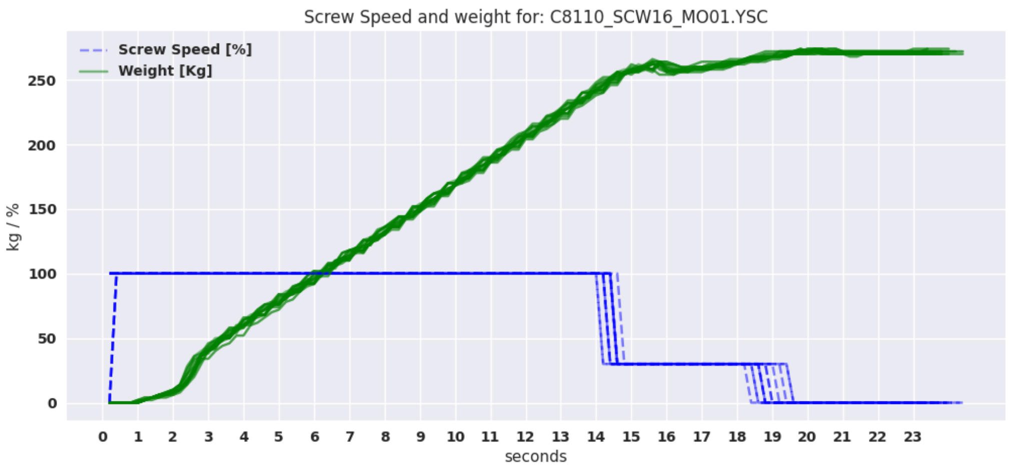


Figure 4.6: Dosing Control for screw 16 with existing control system

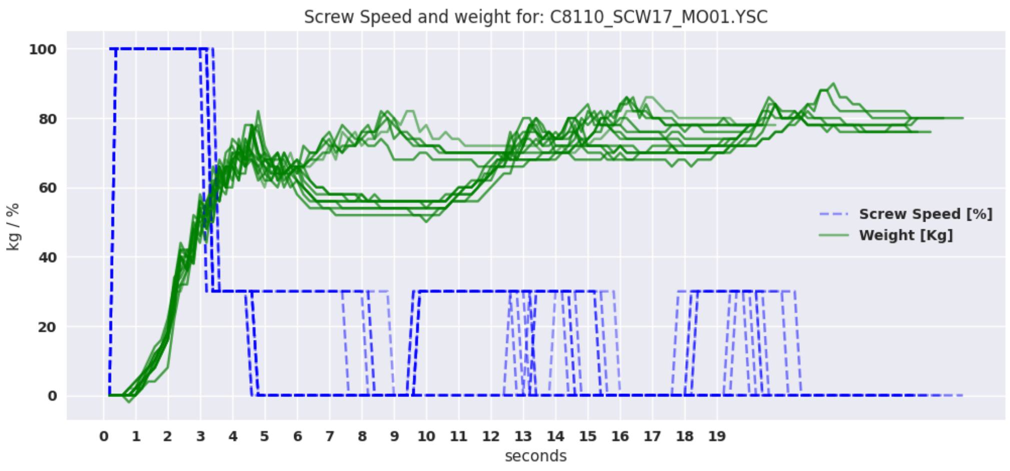


Figure 4.7: Dosing Control for screw 17 with existing control system

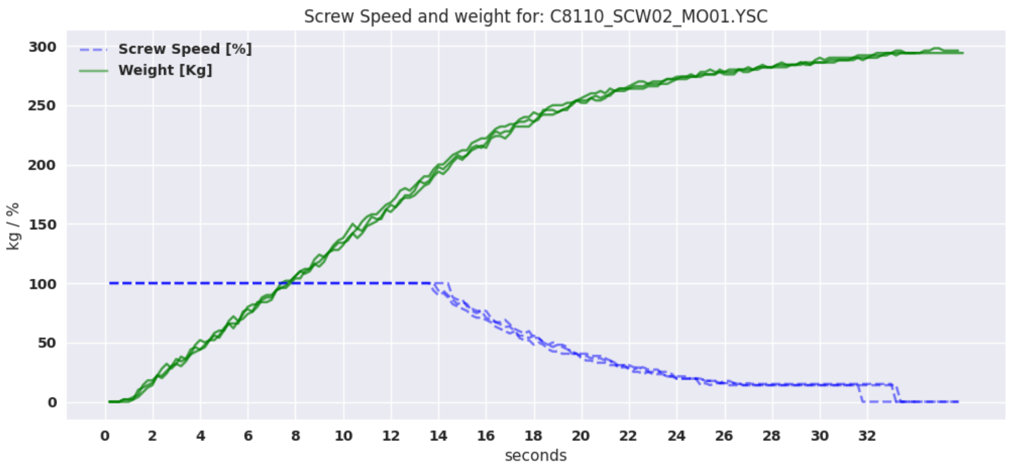
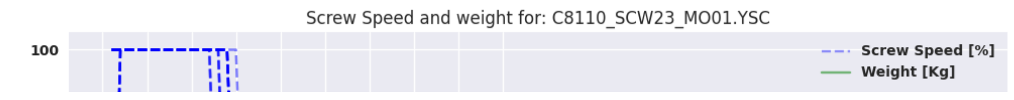


Figure 4.10: P-Control Trial for screw 2



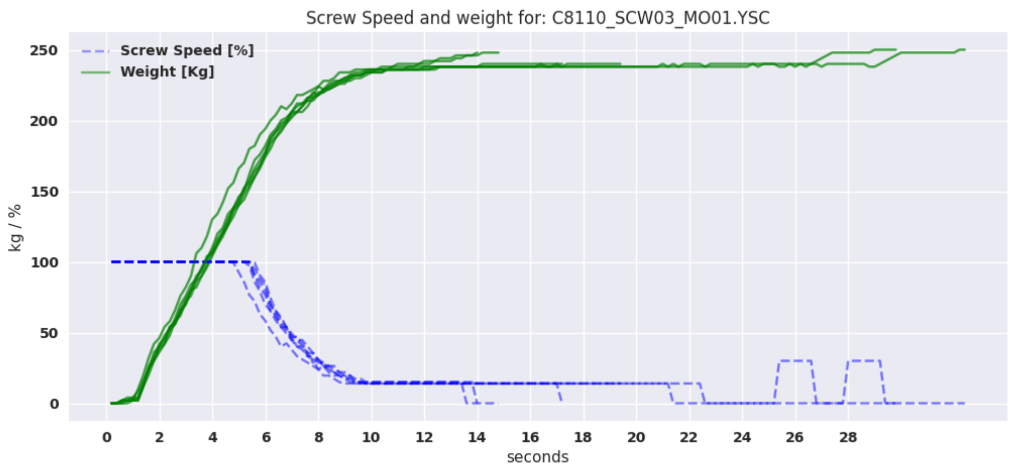


Figure 4.11: P-Control Trial for screw 3

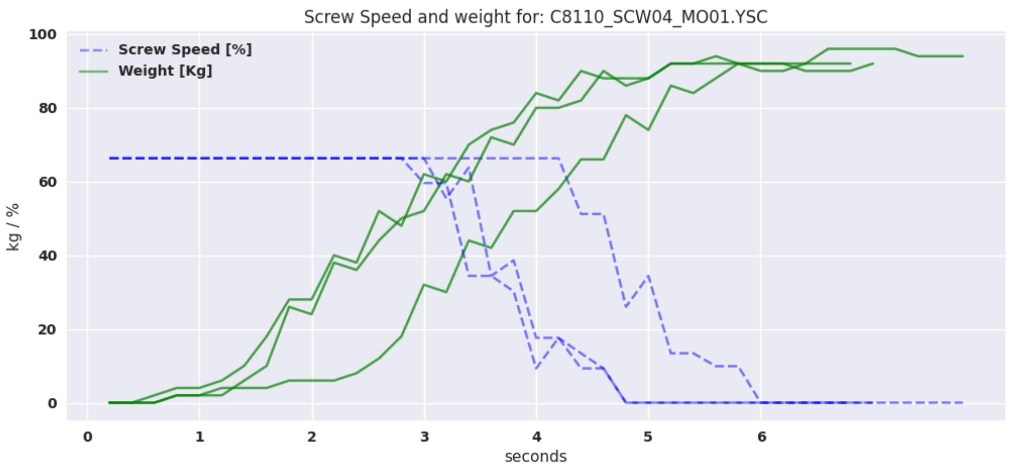


Figure 4.12: P-Control Trial for screw 4

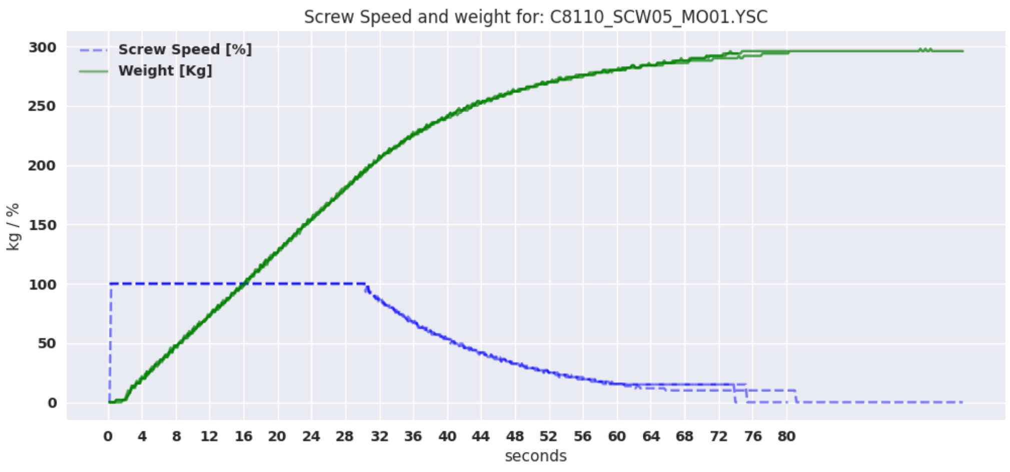


Figure 4.13: P-Control Trial for screw 5

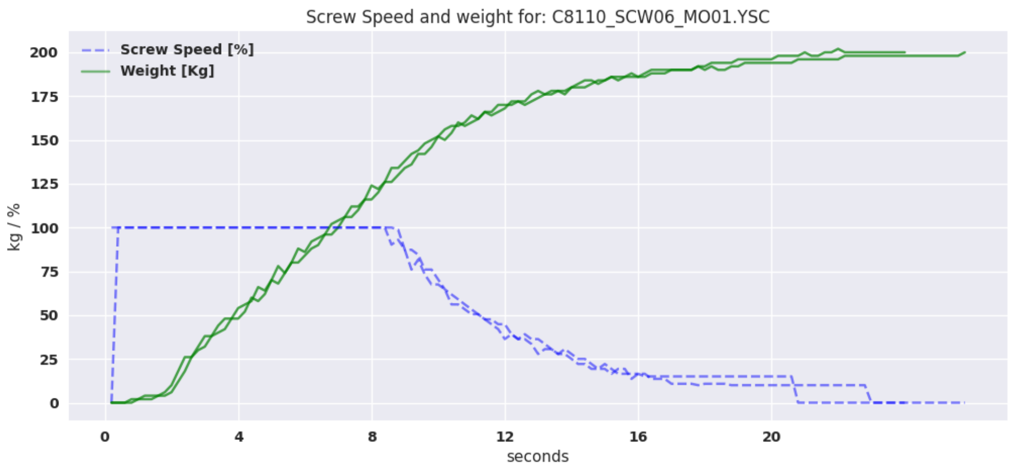


Figure 4.14: P-Control Trial for screw 6

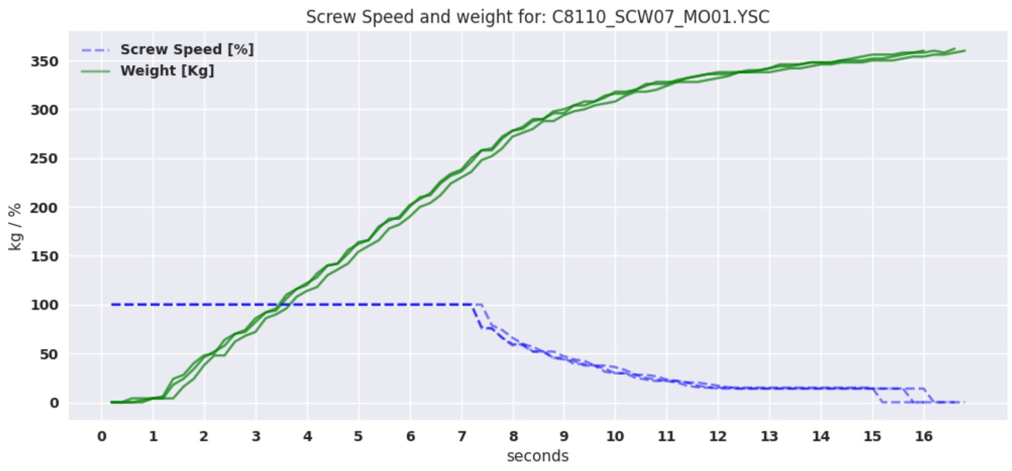
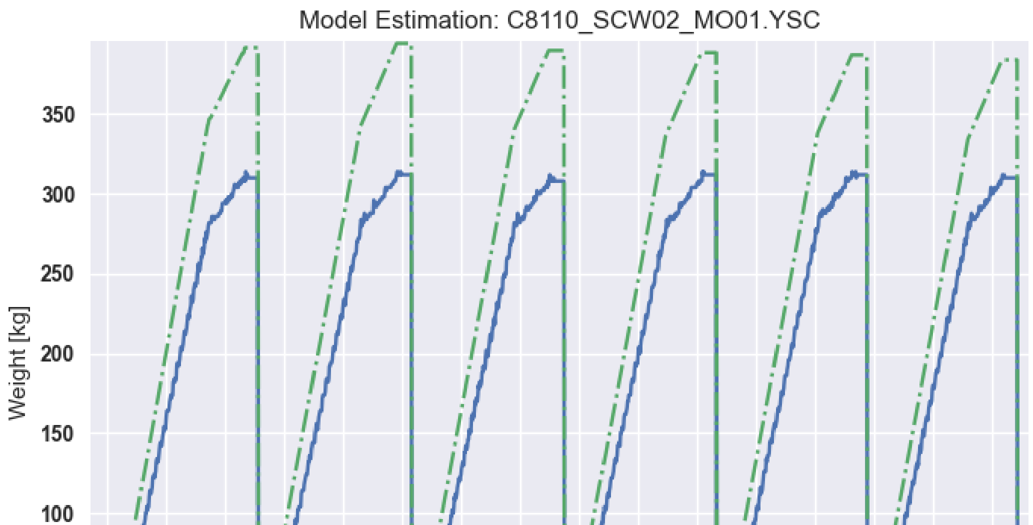


Figure 4.15: P-Control Trial for screw 7

4.0.3 OLS Validation



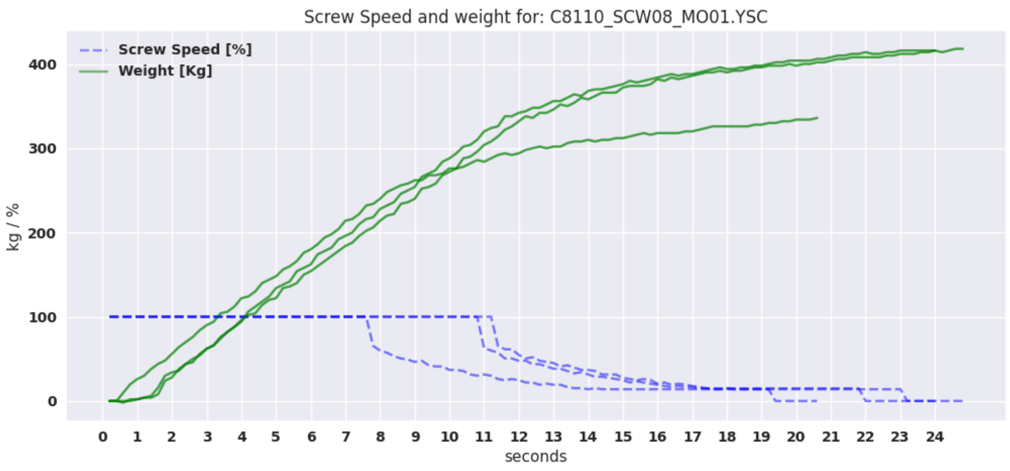


Figure 4.16: P-Control Trial for screw 8

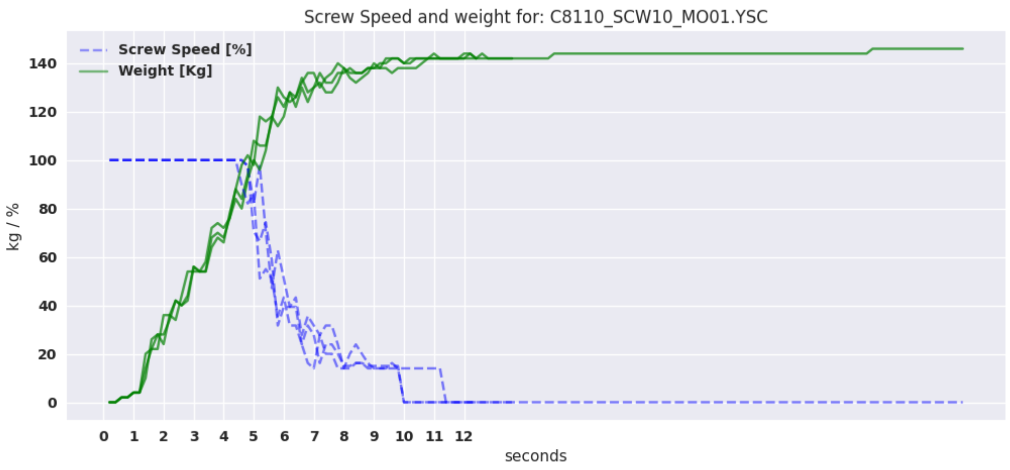


Figure 4.17: P-Control Trial for screw 10

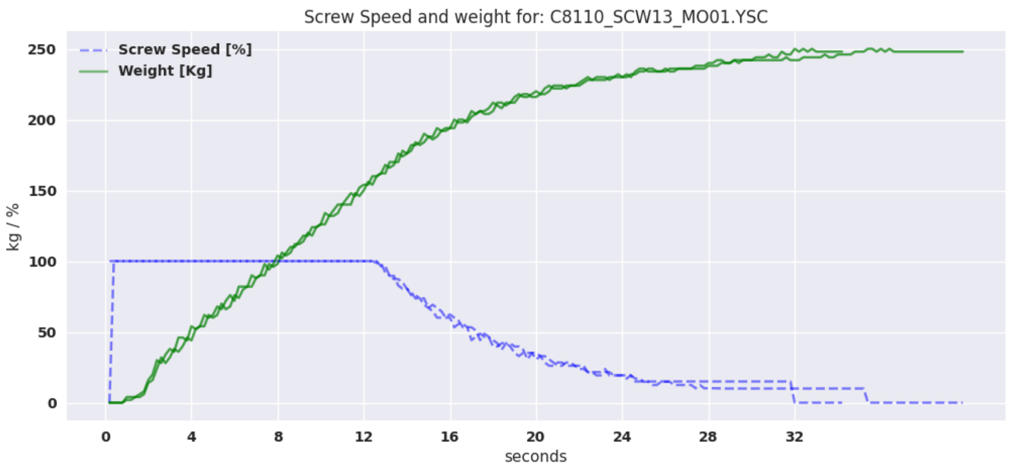


Figure 4.18: P-Control Trial for screw 13

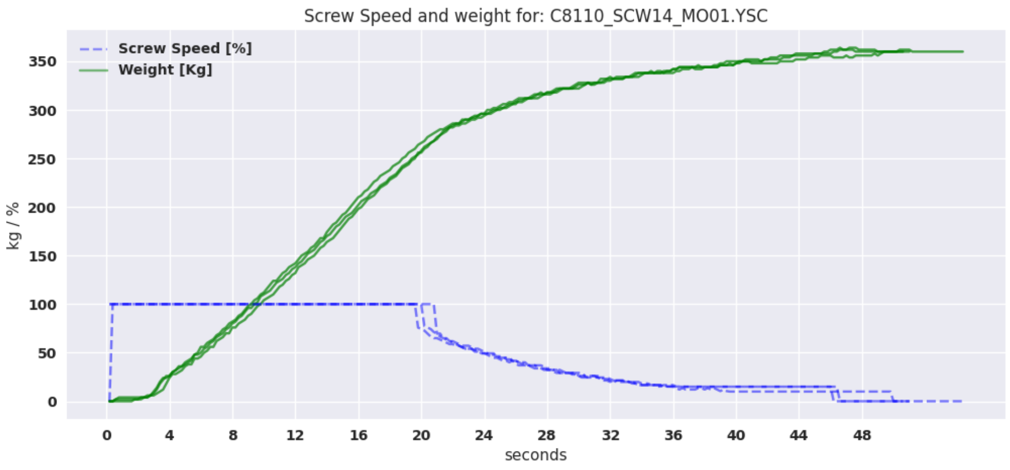
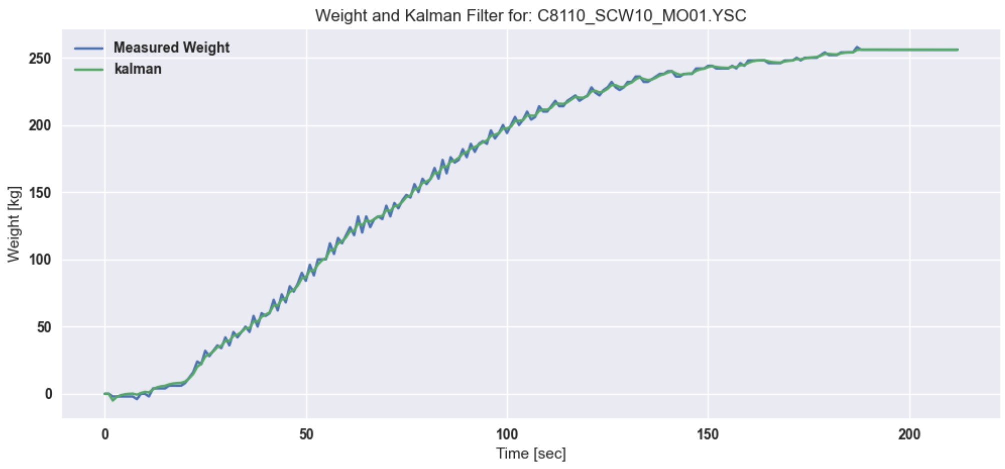


Figure 4.19: P-Control Trial for screw 14

4.0.4 Kalman filter



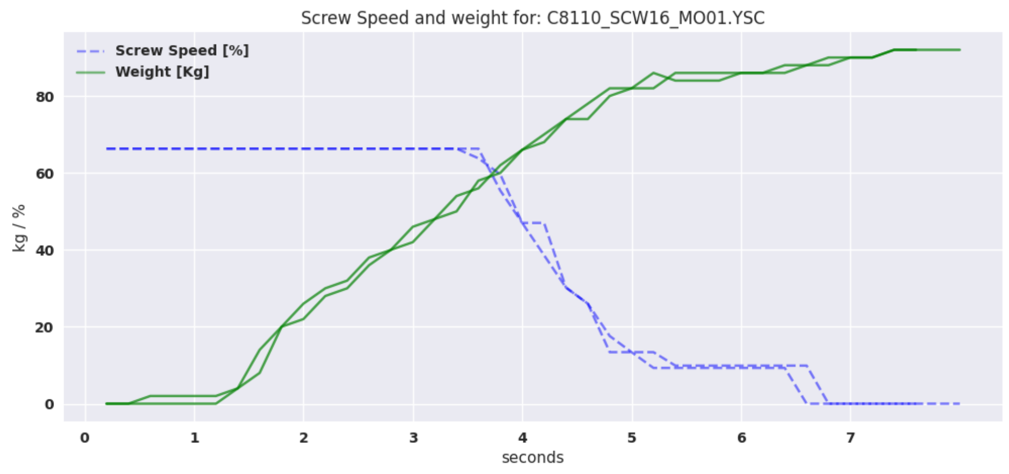


Figure 4.20: P-Control Trial for screw 16

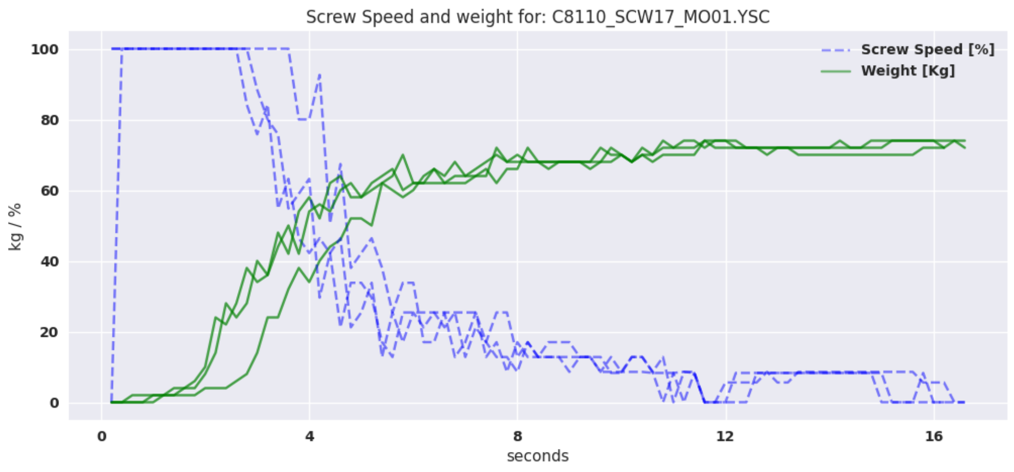


Figure 4.21: P-Control Trial for screw 17

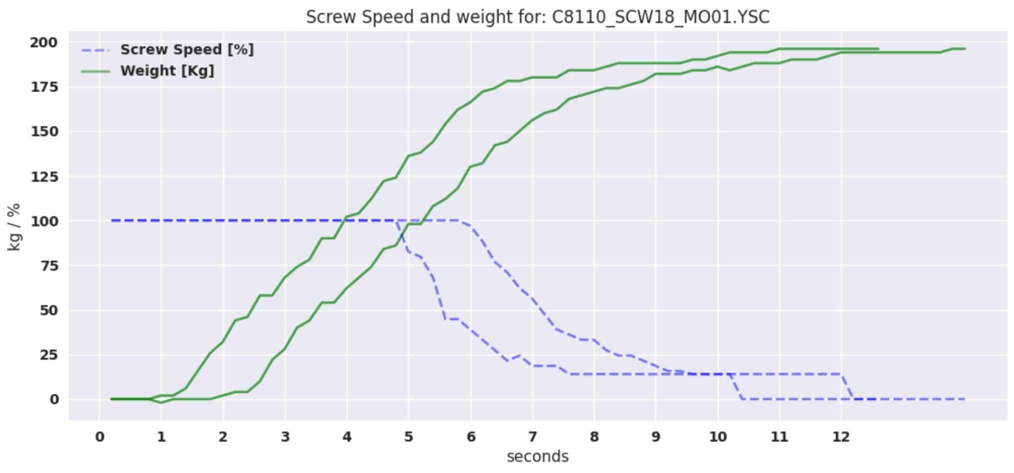


Figure 4.22: P-Control Trial for screw 18

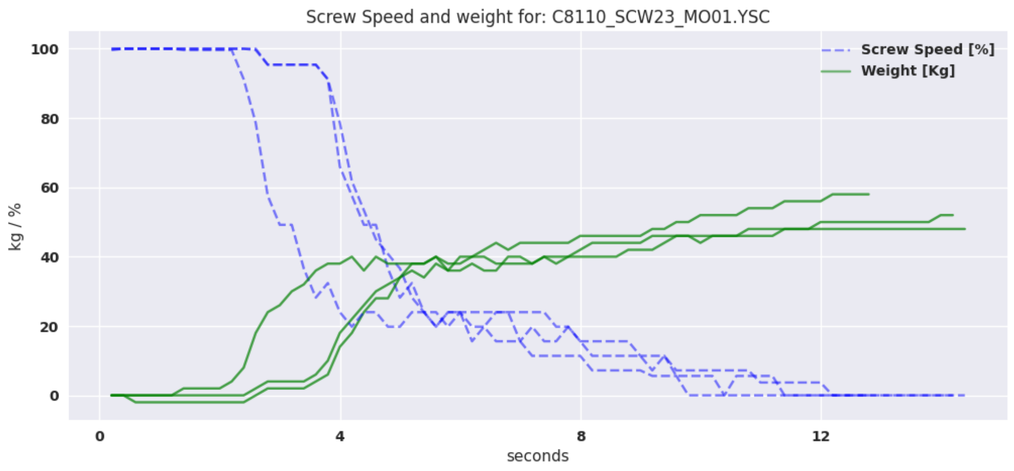
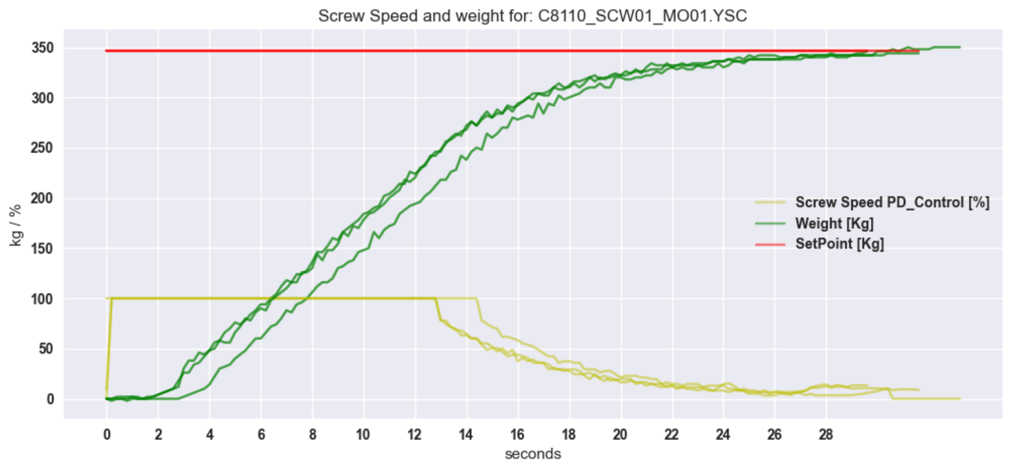


Figure 4.23: P-Control Trial for screw 23

4.0.5 PD-Controller



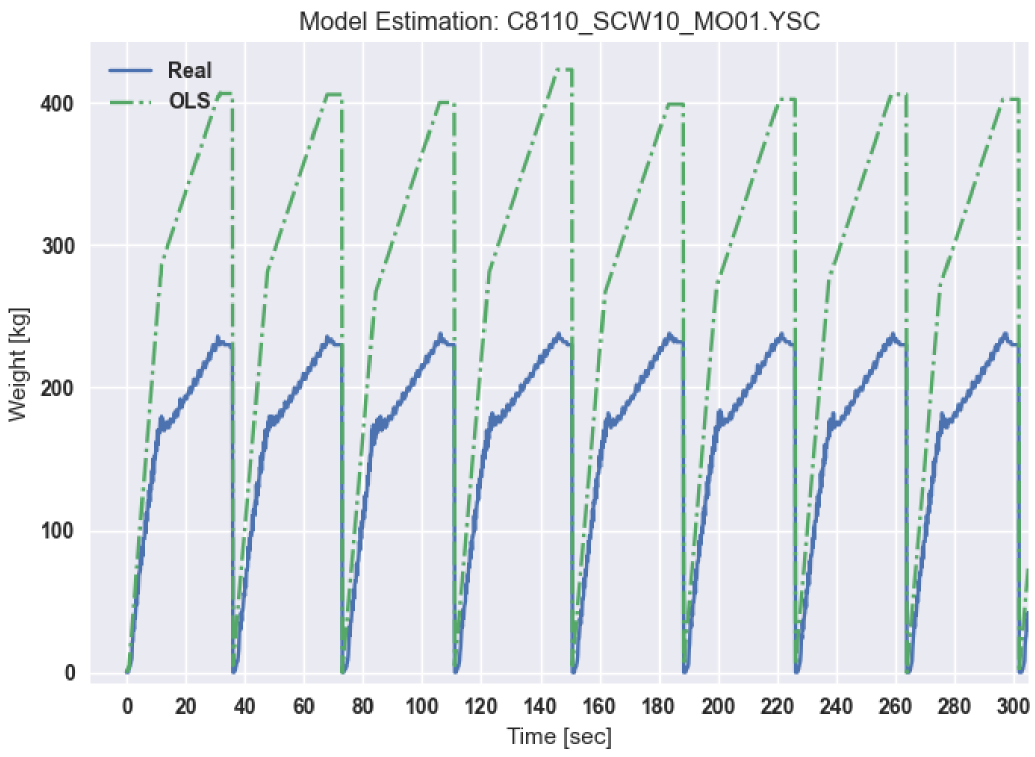


Figure 4.25: OLS-Validation for screw 10

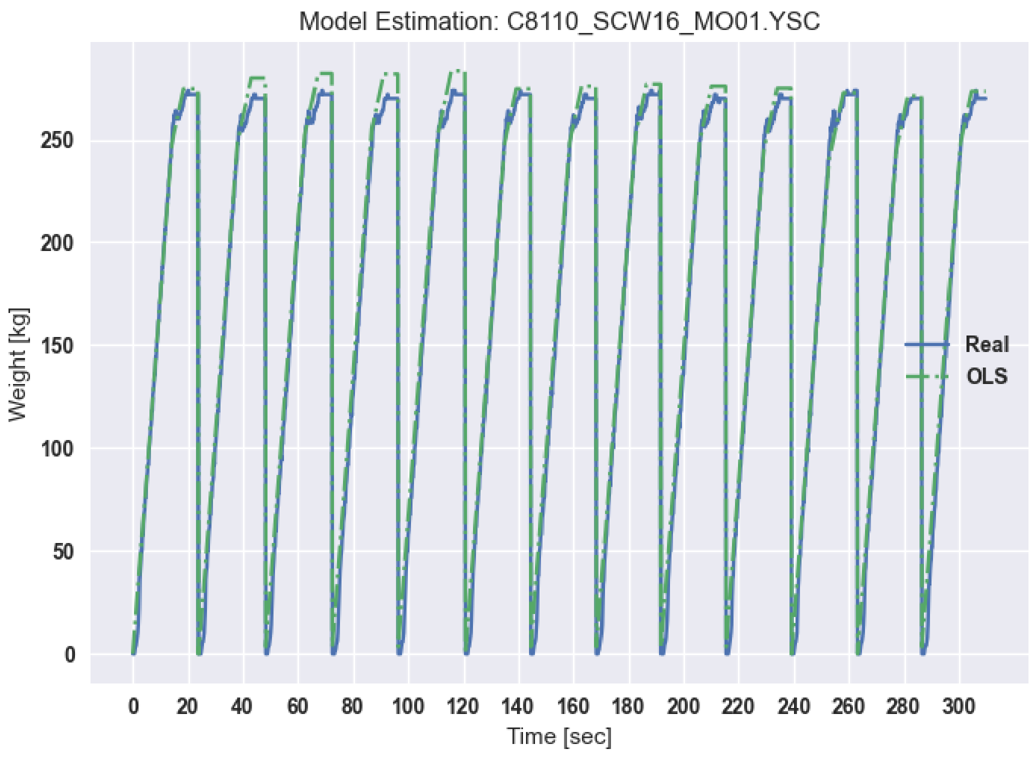


Figure 4.26: OLS-Validation for screw 16

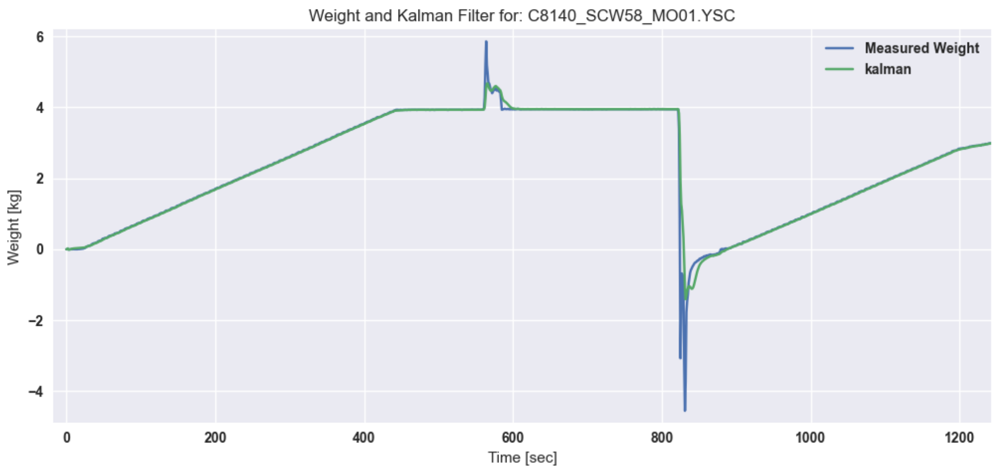


Figure 4.28: Kalman filter for screw 58

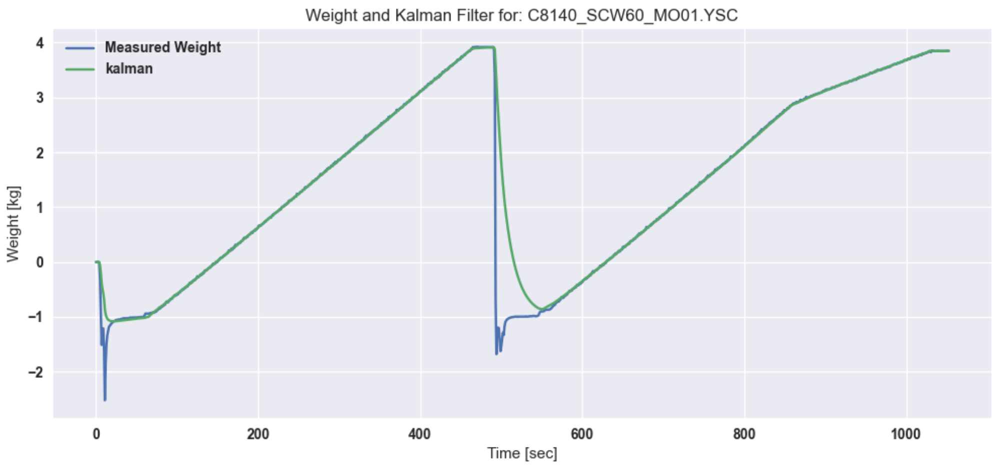
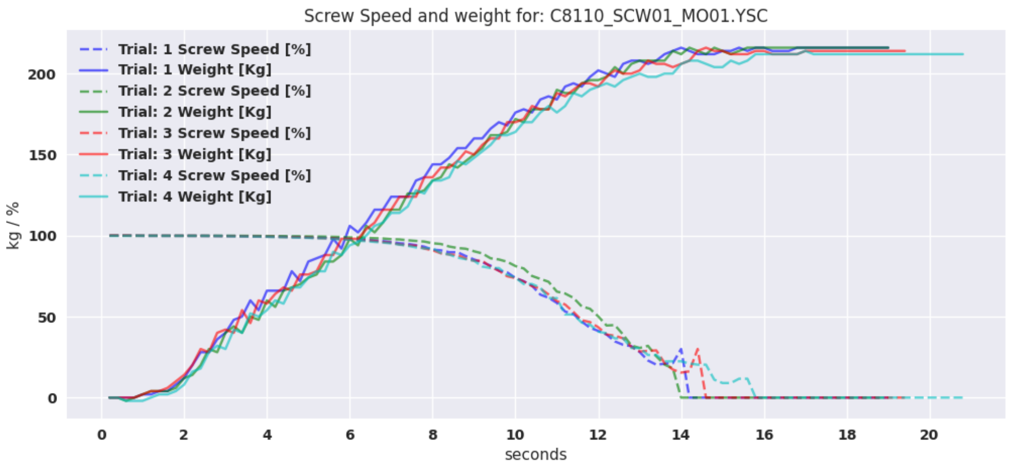


Figure 4.29: Kalman filter for screw 60

4.0.6 Fully assembled solution



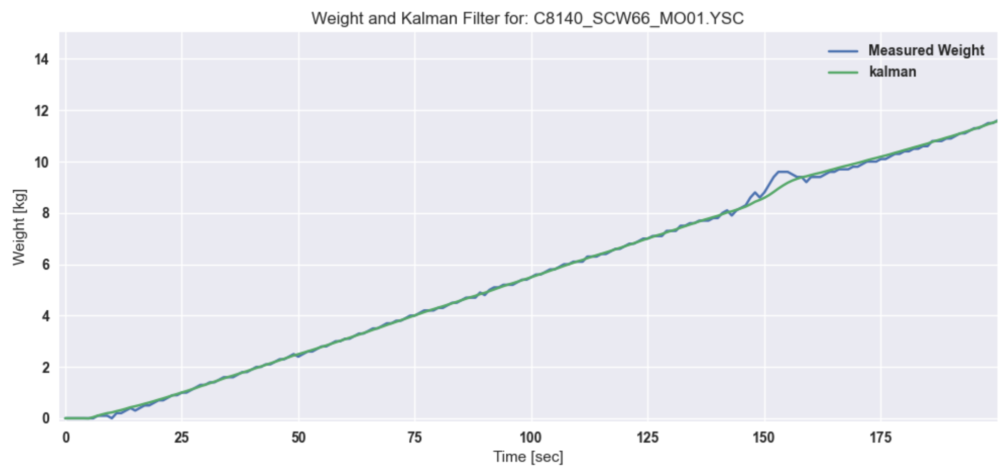


Figure 4.30: Kalman filter for screw 66

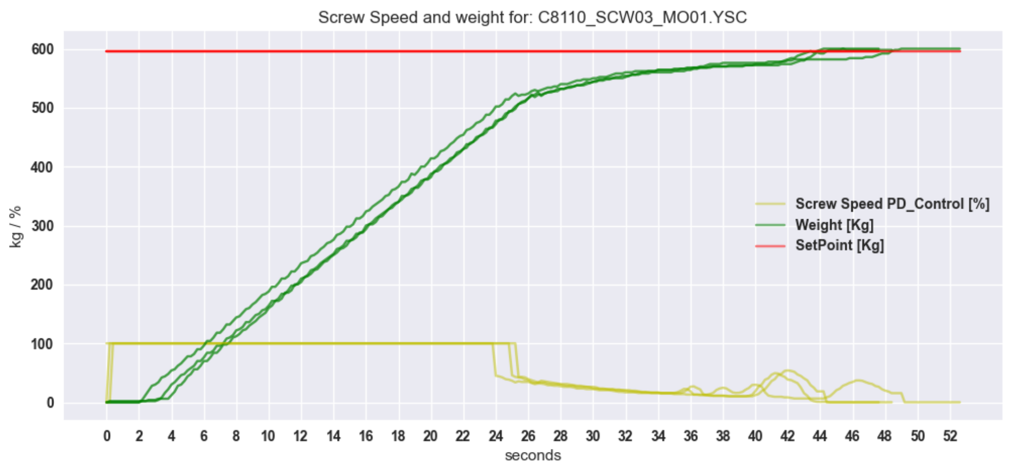


Figure 4.32: PD-Control for screw 3

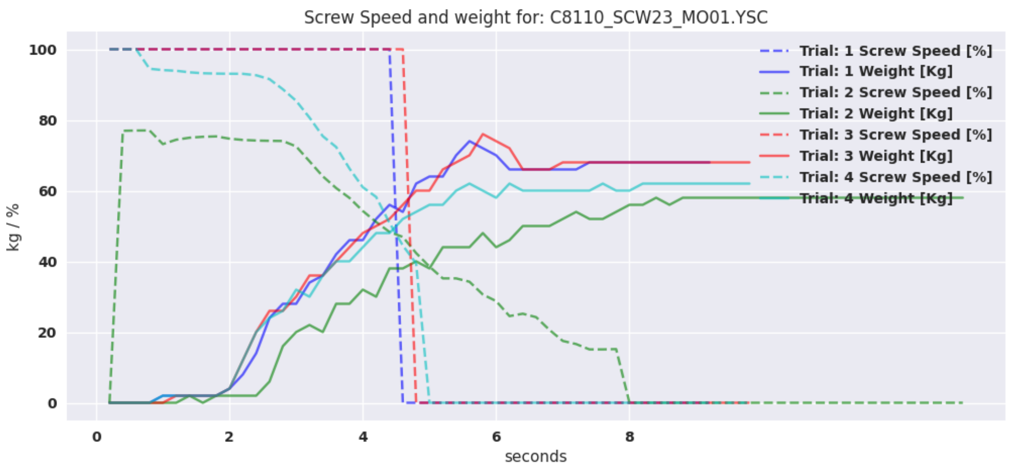


Figure 4.45: Fully assembled solution tested on screw 23

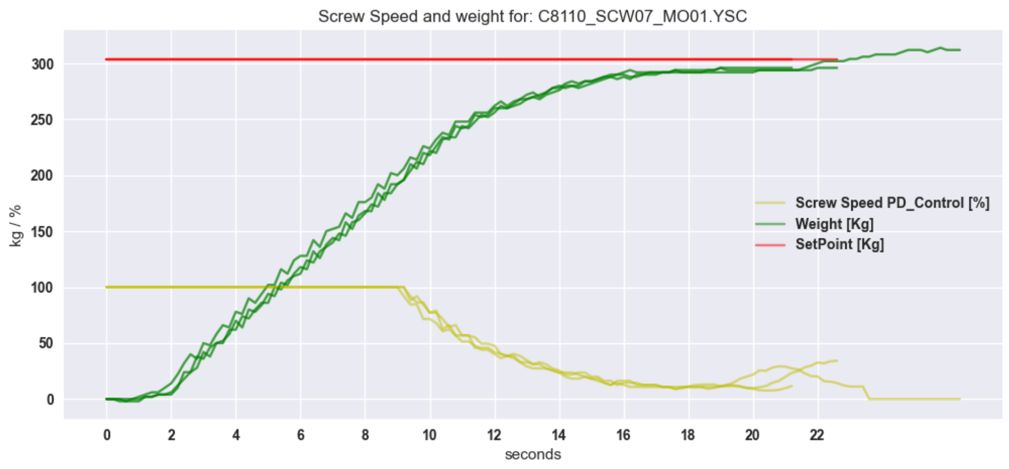


Figure 4.33: PD-Control for screw 7

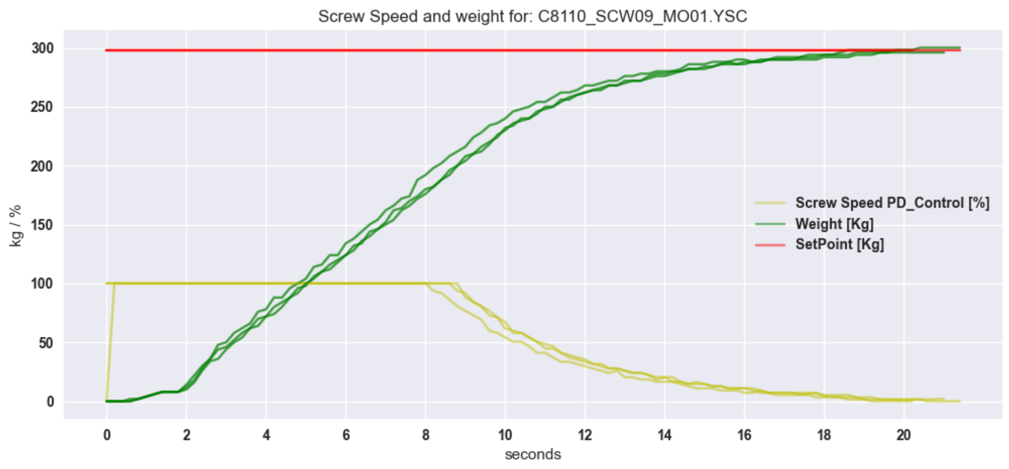


Figure 4.34: PD-Control for screw 9

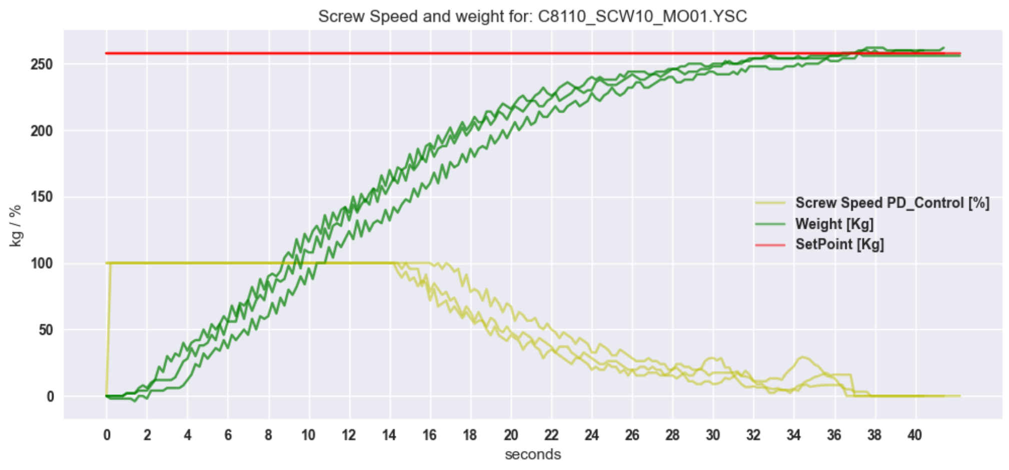


Figure 4.35: PD-Control for screw 10

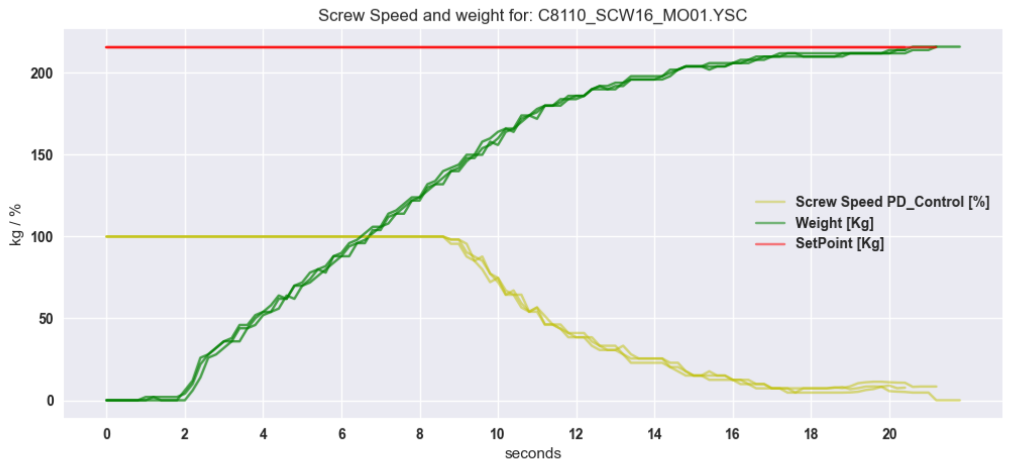


Figure 4.36: PD-Control for screw 16

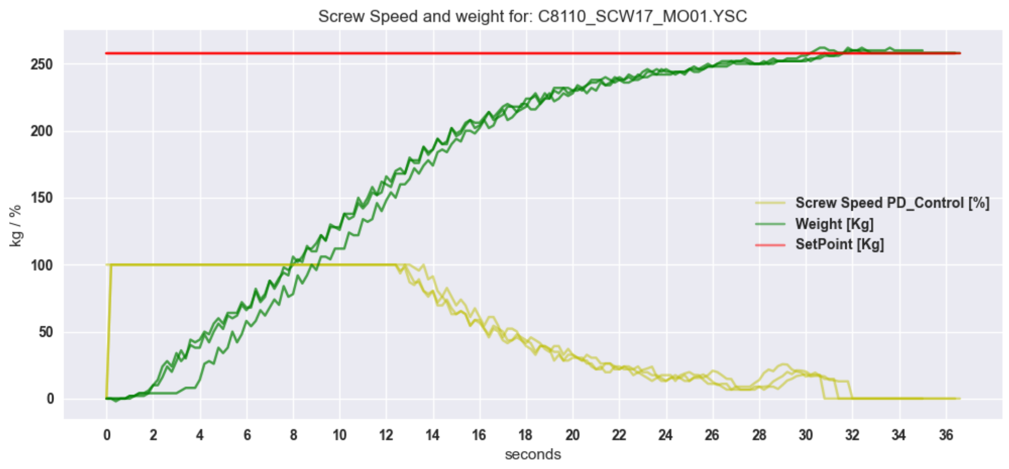


Figure 4.37: PD-Control for screw 17

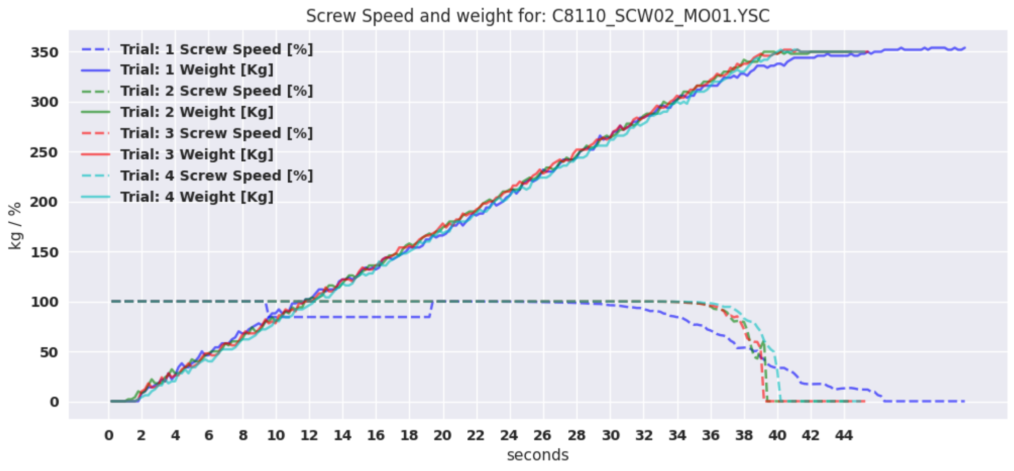


Figure 4.39: Fully assembled solution tested on screw 2

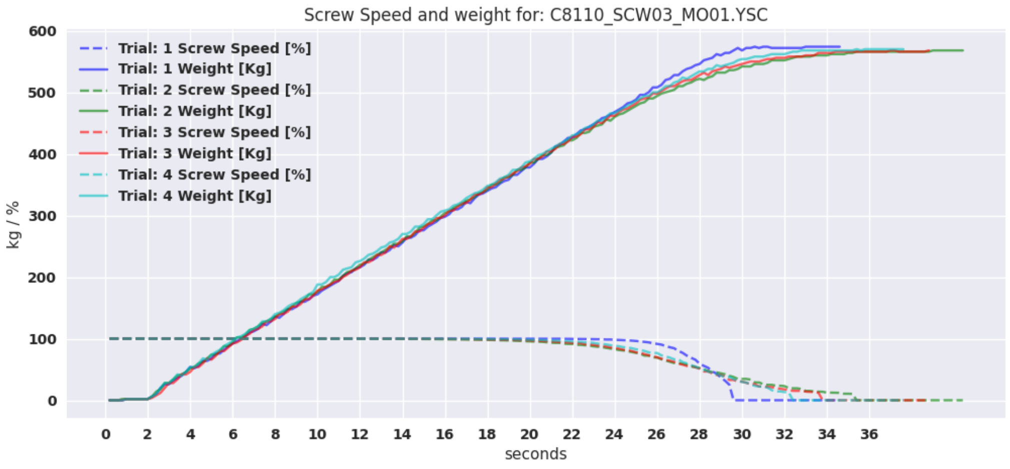


Figure 4.40: Fully assembled solution tested on screw 3

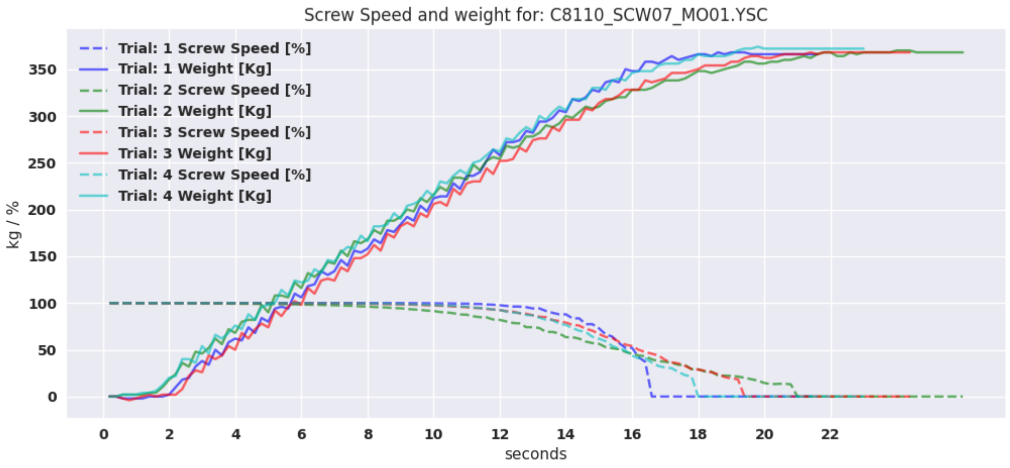


Figure 4.41: Fully assembled solution tested on screw 7

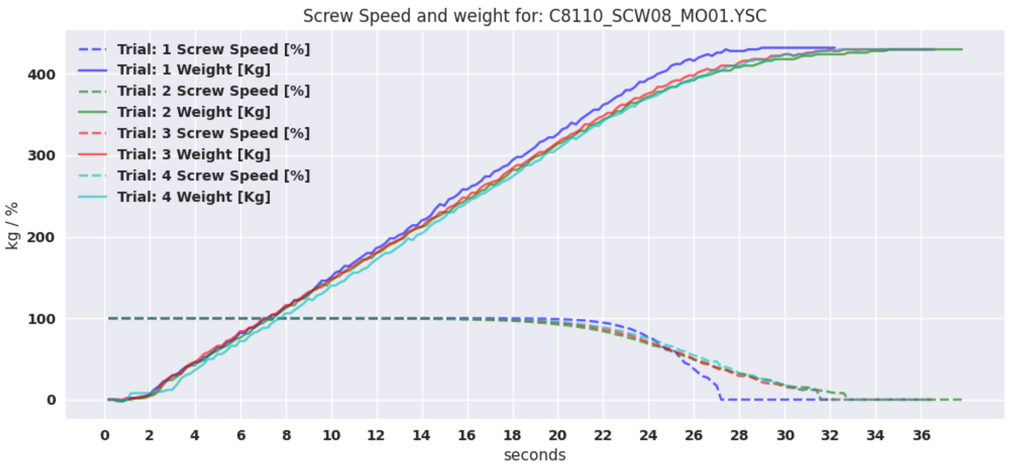


Figure 4.42: Fully assembled solution tested on screw 8

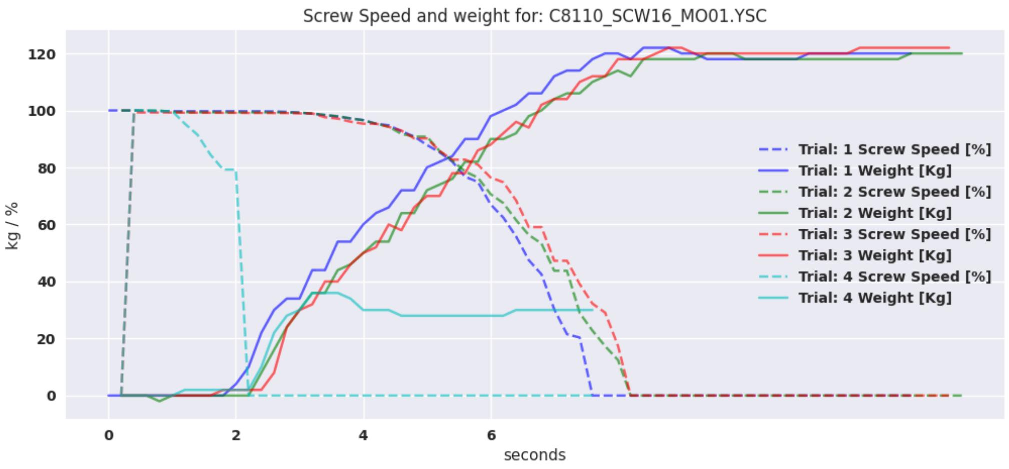


Figure 4.43: Fully assembled solution tested on screw 16

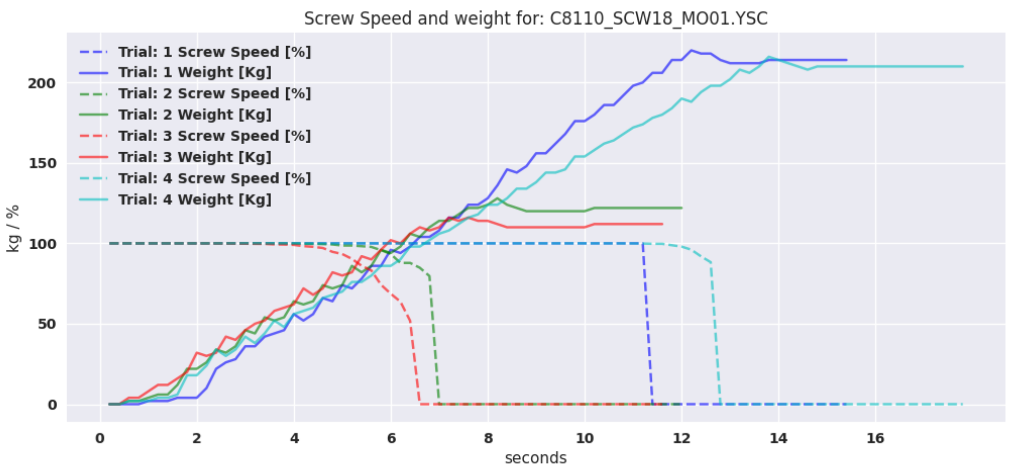


Figure 4.44: Fully assembled solution tested on screw 18

Main-file Python

```
    import pylogix
import keyboard
import json
from ROLS import ModelBasedFilter, data, MPC
from PD_Controller import pd_control
import time
import collections
import sys
import cProfile


def loop(data,units, rols, dt_sp, control):
    #Initialise variables
    ID_valOld = 0
    ScrewTag = ''

    #Initialise timing variables
    StartTime = time.time()
    dt_actual = 0
    dt_start = time.time()
    timespan = 260000 #Three days
    oldScrew = ""
    x = 0
    y = 0
    InitialWeight = 0



    print("Starting reading...")

    while True:

        unit_instance = 0
        for unit, rol in zip(units, rols):
            Silo = unit['Silo']
            WeightTag = unit['Weight']
            WeightDiffTag = unit['WeightDiff']
```

```
SpTag = unit['SP']


unit['PV']['SiloID'] = data.getSilo(Silo, unit)
#Initialise variables for the current unit
ID_val = unit['PV']['SiloID']
ID_valOld = unit['PV']['SiloIDOld']
ScrewTag = unit['PV']['ScrewTag']
InitialWeight = unit['PV']['InitialWeight']
n = unit['PV']['n']


#Update the model
if ID_val != ID_valOld and ID_val != 0:
        rol[n].update(unit_instance, n)
        rol[n].initialised = False



#Assigns the correct screw tag based on the ID value
#print("ID_val: ", ID_val)
if ID_val != 0 and ID_val != ID_valOld :
    unit['PV']['n'] = 0
    #Position of screw in list
    for i in range (0, len(unit['Screws']) ):
        if unit['Screws'][i]['SiloID'] == ID_val:

            ScrewTag = unit['Screws'][i-1]['ScrewName']
            unit['PV']['ScrewTag'] = ScrewTag
            break
    n = i
    print("n: ", n)
    unit['PV']['n'] = n






    if ID_valOld != 0:
        InitialWeight = 0
        unit['PV']['InitialWeight'] = 0
        #print("ID_val: ", n)
    ID_valOld = ID_val
    unit['PV']['SiloIDOld'] = ID_val
    # Assign the correct screw tag based on SIlo id for screw


    #print("ScrewTag: ", ScrewTag)

if ID_val == 0:
    ScrewTag = ''
    unit['PV']['ScrewTag'] = ''


#Read the process data

#print("ScrewTag: ", unit['PV']['ScrewTag'])
if ScrewTag != '':
        speed, Wt, WtDiff, SP = data.ReadProcessData(ScrewTag, WeightTag,

        if Wt is None:
            Wt = 0
        if speed is None:
            speed = 0

        #Substract the initial weight
        if InitialWeight == 0:
            InitialWeight = Wt
            unit['PV']['InitialWeight'] = InitialWeight
        Wt = Wt - InitialWeight

        #Initialise the model if it is not already initialised
        if rol[n].initialised == False:
            rol[n].initialise()



        if rol[n].initialised == True:
            rol[n].speed = speed
            rol[n].weight = Wt
            rol[n].weight_diff = WtDiff
            rol[n].sp = SP
```

```
                if SP is not None and SP > 0 and rol[n].Optimized == False:
                    #cProfile.runctx(f'rol[{n}].OptimalControl()', globals(),
                    rol[n].OptimalControl()


                rol[n].dataPrep(Wt, WtDiff, speed) #Create model structure
                if SP is not None and SP > 0:
                    cv = rol[n].PDControl(state = rol[n].state, pGain= rol[n]
                rol[n].kalman()

                #Write the state to the PLC
                data.write(unit['KalmanWeight'], rol[n].state[0] + InitialWei



    elif data.datasource == 'PLC' and ScrewTag == '' and WeightTag != '':
        Wt = data.read(WeightTag)

        if Wt is not None:
            data.write(unit['KalmanWeight'], Wt) #Write the weight to the PLC


    unit_instance += 1


#Delays to ensure that the loop runs at a constant rate
dt_actual = (time.time() - dt_start)*1000

if dt_actual < dt_sp:
    time.sleep((dt_sp-dt_actual)/1000)

#print("dt: ", time.time() - dt_start)
dt_actual = (time.time() - dt_start)*1000
dt_start = time.time()



if time.time() - StartTime > timespan:
    break
    # except KeyboardInterrupt:
    #     print("Keyboard interrupt!")
    #     exit()
    # except Exception as e:
    #     print("Error in control loop")
    #     print(e)
    #     print("Error on line {}".format(sys.exc_info()[-1].tb_lineno))
    #     exit()





if __name__ == "__main__":

    #load json config file with tags
    with open('C:\\Users\\isak.skeie\Desktop\\MasterEscapade\\config.json', 'r') as j
        config = json.load(json_file)


    #number of screws
    units = config['DosingUnits']
    n_units = len(config['DosingUnits'])
    rols = [[None] * len(unit['Screws']) for unit in units]
    dt_sp = 2 # Sought after time interval in milliseconds

    for i in range(n_units):
        for n in range(len(units[i]['Screws'])):
            rols[i][n] = ModelBasedFilter(units[i]['Screws'][n], units[i])

    data = data('CSV')
    control = pd_control()


    try:
        loop(data, units,rols, dt_sp, control) #Need to pass all units when onccce is
        pass
    except KeyboardInterrupt:
        print("Keyboard interrupt!")
```

Class-File Python

```
        import pandas as pd
import pylogix
import time
import os
import numpy as np
import collections
import keyboard
from filterpy.kalman import KalmanFilter
from filterpy.common import Q_discrete_white_noise
import sys
import statsmodels.api as sm
import json
import matplotlib.pyplot as plt
from scipy.signal import lfilter_zi, lfilter, butter
import importlib
from scipy.optimize import minimize
from scipy.optimize import Bounds
from filterpy.common import Saver
import matplotlib
from math import tanh, exp



plt.style.use('seaborn-v0_8')  # Use ggplot style
plt.style.use('seaborn')
font = {
        'weight' : 'bold',
        'size'   : 26}

matplotlib.rc('font', **font)
plt.rcParams['figure.figsize'] = [12, 5]  # width, height in inches


class ModelBasedFilter: #Kalman filter and Recursive Least Squares algorithm
    def __init__(self, screw, unit):

        #Initialise the model parameters
```

```
self.theta = np.array(screw['Theta'])
self.unit = unit
self.screw = screw
self.dt = 0.2
self.lag = 2 #Lag for variables
self.window = 10000
self.lambda_ = 0.8 #Forgetting factor for theta
self.A = np.array([
            [1,1],
            [0,0]])

self.B = lambda theta: np.array([[0,0,0,0,0,0],
                                 [theta[0], theta[1], theta[2],
                                  theta[3],theta[4], theta[5]]])

self.n_states = 2
self.n_inputs = 6


self.weight = 0
self.weight_diff = 0


self.U = collections.deque([0]*self.lag,maxlen=self.n_inputs)  #Speed inputs
for i in range(self.n_inputs):
    self.U.append(0)
self.Y = np.zeros(1)



self.X_kalman = np.zeros((self.n_states,1))
self.Y_kalman = np.zeros(2)


self._y = np.zeros(1) #single Y vector
self.x_Window = collections.deque( maxlen=self.window) #Window of X vectors
self.y_Window = collections.deque(maxlen=self.window) #Window of y vectors

self.initialised = False


self.state = np.zeros((self.n_states)) #Initial state
self.sp = 1
self.cv = 0
self.cv_array = np.zeros(1)
self.U_array = np.zeros((1,self.n_inputs))


#Initialze control
self.U_ini = 0  #Initial control input
self.error = 0  #Reference - Weight
self.Weight_init = 0   #Initial weight, to be subtracted
self.initiate = False   #Bit to initiate control variables
self.ControlPoint  = 0 #Point where proportional control starts
self.pGain  =   0     #Proportional gain

self.FlowSP = 0 #Flow setpoint
self.dGain = 0 #Derivative gain
self.di_cv = 0 #Derivative control accumulator


self.Optimized = False

#Optimal control
self.Qe = 1 #Weighting matrix for the error
self.Pu = 1 #Weighting matrix for the control inputs
self.mpc = MPC(self.state, [self.U_ini, self.ControlPoint, self.pGain, self.d
self.Optimized = False


#Kalman filter
self.kf = KalmanFilter(dim_x=self.n_states, dim_z=2)
self.kf.x = np.array([0, 0.])
self.kf.F = self.A
self.kf.H = np.array([[1.,0.], [0., 1.]])
self.kf.R = np.array(unit["MeasurementNoise"])
self.kf.Q = np.array(unit["ProcessNoise"])

self.kf.P = np.array([[1,    1],
                [1, 1] ])
self.s = Saver(self.kf)

# Parameters for the filter
```

```
        self.order = 3       # order of the filter
        self.fs = 5      # sample rate, Hz
        self.cutoff = 1.5  # desired cutoff frequency of the filter, Hz
        self.a, self.b = self.butter_lowpass()

     #Function that adds the speed and weight to the model
    def dataPrep(self, weight, weight_diff, speed):
        try:

            #Create X and Y vector
            x = np.zeros((self.n_states,1))
            x[:,0] = [weight, weight_diff]
            self.U.appendleft(speed)
            self.U.appendleft(speed**(3/2))
            self.U.appendleft(speed**(1/2))


            self.Y = np.append(self.Y, weight)
            self.cv_array = np.append(self.cv_array, self.cv)

            self.U_array = np.vstack((self.U_array, self.U))

            #Assign current state
            self._y = [weight, weight_diff]
        except Exception as e:
            print("Error in dataPrep")
            print(e)
            print("Error on line {}".format(sys.exc_info()[-1].tb_lineno))
            exit()


    def initialise(self):

        self.U = collections.deque([0]*self.lag,maxlen=self.n_inputs)  #Speed inputs
        for i in range(self.n_inputs):
            self.U.append(0)

        self.Y = np.zeros(1)
        self.speed = 0
        self.weight = 0

        self.X_kalman = np.zeros((self.n_states,1))
        self.Y_kalman = np.zeros(1)


        self.initialised = True
        self.state = np.zeros((self.n_states))
        self.cv_array = np.zeros(1)
        self.U_array = np.zeros((1,self.n_inputs))
        self.di_cv = 0
        self.Optimized = False

        self.kf.x = np.array([0, 0.])
        self.kf.P = np.array([[1,    0.],
                        [0., 1] ])
        #self.control.initiateControl(self.sp, self.weight)


    def kalman(self):

        y = self._y
        self.kf.predict(u= self.U, B = self.B(self.theta))


        #if abs(self.kf.x_prior[0] - self._y[0]) > self.kf.R[0,0]*2:
        #    y[0] = self.kf.x_prior[0]


        #Keeps the filter from diverging when disturbances are too large
        if abs(self.kf.x_prior[1] - self._y[1]) > self.kf.R[1,1]:
            y[1] = self.kf.x_prior[1]




        self.kf.update(y)
        self.Y_kalman = np.append(self.Y_kalman, self.kf.x[0])
        self.state = self.kf.x

        self.s.save()  # save the current state

    def update(self,unit_instance, n):
        y_batch = self.Y
        #Plot the measured weight
        plt.plot(y_batch, label='Measured Weight')
        self.s.to_array()

        try:
            # Update the model parameters using the window least squares algorithm
            y_old = y_batch[0]
            for x, y in zip(self.U_array, y_batch):
                y_diff = y - y_old
                self.x_Window.appendleft(x)
                self.y_Window.appendleft(y_diff)
                y_old = y

            # Calculate the OLS estimates
            x = np.array(self.x_Window)
            y = np.array(self.y_Window)


            lambda_ = 1000  # You can adjust this value as needed
            theta = None
            try:
                if len(x) > 100:
                    I = np.eye(np.shape(x)[1])
                    theta = np.linalg.inv(x.T @ x + lambda_ * I) @ x.T @ y



            except Exception as e:
                print("Error in OLS update: ", e)
                print("Error on line {}".format(sys.exc_info()[-1].tb_lineno))


            if theta is not None:
                if len(theta) == self.n_inputs:
                    self.theta = theta

            tril = np.tril(np.ones((len(y_batch), len(y_batch))))
            ydot_hat = self.U_array @ self.theta
            y_hat = tril @ ydot_hat
            #plt.plot(y_hat, label='OLS prediction')
            # Write the updated config JSON file
            with open('config.json', 'r') as json_file:
                config_data = json.load(json_file)
            # Update specific keys in the config data

            config_data['DosingUnits'][unit_instance]['Screws'][n]['Theta'] = self.th

            with open('config.json', 'w') as json_file:
                json.dump(config_data, json_file, indent=4)

            plt.title("Weight and Kalman Filter for: " + self.screw['ScrewName'] )
            plt.plot(self.Y_kalman, label='kalman')

            ticks = np.arange(0, len(y_hat)//5, 1)



            #plt.xticks(np.arange(0,len(y_hat)-1, 5), (ticks).astype(int))
            plt.xlabel('Time [sec]')
            plt.ylabel('Weight [kg]')
            plt.grid()
            plt.legend()
            #if len(y) > 10:
                #plt.show()


            plt.grid()



        except Exception as e:
            print("Error in ROLS update")
            print(e)
            print("Error on line {}".format(sys.exc_info()[-1].tb_lineno))
            plt.close()
            #self.comm.Close() need to close data connection
            plt.close()
            #exit()



    def PDControl(self, state, pGain, dGain, FlowSP):
```

```
#Ensures correct shape of state array
if state.shape == (self.n_states, 1):
    state = state.reshape((self.n_states,))



weight = state[0]
#weight_diff = self.butter_lowpass_filter(state[1]/self.dt)
weight_diff = state[1]
#print("Filtered weight diff: ", self.weight_diff, "Old weight diff: ", self.

error  = self.sp - weight

if self.sp > 0 and error/self.sp < 5:
    FlowSP = FlowSP*exp(error/self.sp)

if weight / self.sp < 0.5:
    self.di_cv = 0

self.di_cv += (1 - weight_diff/FlowSP)*dGain
if self.di_cv > 1: self.di_cv = 1
if self.di_cv < 0: self.di_cv = 0

cv = (tanh((error/self.sp)*pGain) + self.di_cv) * 100
if cv > 100: cv = 100


#Stops contorl when input and weight diff reaches ref in n dt

if weight/self.sp > 0.95:

    for i in range(1, 5):
        state = self.A @ state + self.B(self.theta) @ self.U
        if state[0] >= self.sp:
            cv = 0
            break

return cv
```

```
    def OptimalControl(self):

        A = self.A
        B = self.B(self.theta)
        self.di_cv = 0
        self.pGain, self.FlowSP, self.dGain = self.mpc.minimize(self.sp, A, B, self.p
        self.state = np.zeros((self.n_states,1))
        self.Optimized = True

        # Function to design a Butterworth lowpass filter
    def butter_lowpass(self):
        nyq = 0.5 * self.fs
        normal_cutoff = self.cutoff / nyq
        b, a = butter(self.order, normal_cutoff, btype='low', analog=False)
        return a, b

    # Function to apply the filter
    def butter_lowpass_filter(self, data):

        if not hasattr(self, 'zi'):  # If initial conditions don't exist, create them
            self.zi = lfilter_zi(self.b, self.a) * data
        y, self.zi = lfilter(self.b, self.a, [data], zi=self.zi)

        return y[0]


class data:
    def __init__(self, datasource):
        self.datasource = datasource



        if datasource == 'PLC':
            comm = pylogix.PLC()
            comm.IPAddress = '172.25.0.10'
            comm.ProcessorSlot = 0
            self.comm = comm
        elif datasource == 'CSV':
            self.comm = None
            self.data = pd.read_csv('C:\\Users\\isak.skeie\\Desktop\\MasterEscapade\\
    def getSilo(self, silo, unit):

        self.unit = unit
        if self.datasource == 'PLC':

            silo =  self.comm.Read(silo).Value

        elif self.datasource == 'CSV':
            silo = 0
            CurrentScrew = self.data.iloc[0]['TagName']
            #Get the SiloID from the CSV file
            for screw in self.unit['Screws']:
                if screw['ScrewName'] == CurrentScrew:
                    silo  = screw['SiloID']
        return silo


    def ReadProcessData(self, ScrewTag, WeightTag, WeightDiffTag, SpTag):
        if self.datasource == 'PLC':
            speed       = self.comm.Read(tag=ScrewTag).Value
            Wt          = self.comm.Read(tag = WeightTag).Value
            WtDiff      = self.comm.Read(tag = WeightDiffTag).Value
            SP          = self.comm.Read(tag = SpTag).Value

        elif self.datasource == 'CSV':

            speed = self.data.iloc[0]['Speed']
            Wt = self.data.iloc[0]['Weight']
            SP = self.data.iloc[0]['SP']
            WtDiff = self.data.iloc[0]['WeightDiff']
            self.OldData = self.data.iloc[0]['Weight']

            if WtDiff > 10 or WtDiff < -5:
                WtDiff = 0
            #WtDiff = self.butter_lowpass_filter(WtDiff)

            #remove the first row
            self.data = self.data.iloc[1:]


        return speed, Wt, WtDiff, SP
    def read(self, tag):
        if self.datasource == 'PLC':
            return self.comm.Read(tag).Value
        elif self.datasource == 'CSV':
            return data[tag]
        else:
            return 0

    def write(self, tag, value):
        if self.datasource == 'PLC':
            self.comm.Write(tag, value)
        elif self.datasource == 'CSV':
            pass





#
# Single run MPC
#
class MPC:


    def __init__(self, state_ini_values, u_ini, dt, horizon, Qe, Pu, ControlFunc):

        self.Np = horizon #prediction horizon
        self.dt = dt
        self.state = state_ini_values
        self.u_ini = u_ini
        self.Qe = Qe
        self.Pu = Pu
        self.Pc = np.zeros(self.Np)
        self.ControlFunc = ControlFunc
        self.delta_cv = [0]
        self.di_cv = 0

        self.lag = 2 #Lag for variables
        self.n_inputs = 3
        self.max_rate = 0
        self.U = collections.deque([0],maxlen=self.n_inputs*self.lag)  #Speed inputs
        for i in range(self.n_inputs*self.lag):
            self.U.append(0)



    def objective(self, u_ini):


        #In this function we:
        #define functions for objective and constraints

        I = np.eye(self.Np)

        if self.Ref != 0:
            Qe = np.eye(self.Np)*self.Qe / self.Ref  #weighting matrix for the error
            Pu = np.eye(self.Np)*self.Pu / self.Ref   #weighting matrix for the contr


        Ref = self.Ref


        #Reset the control inputs
        for i in range(self.n_inputs*self.lag):
            self.U.append(0)

        #define the objective function
        J = lambda Pc, delta_u: np.trace((Ref-Pc).T*Qe*(Ref-Pc)) + np.trace(delta_u.T

        Pc = np.zeros(self.Np)

        self.Pc = Pc
        #print("State :", self.state_ini_values)
        #print("Control :", u_ini)
        J_final = 0
        self.delta_cv = [0]
        self.last_cv = 60


        for i in range(self.Np-1):
            weight = self.costFunc(u_ini)[0]
            Pc[i] = weight






        #J_final = np.sum(J(Pc, self.delta_cv))
        self.delta_cv[0:4] = 0 #removes cost in initialization
        self.delta_cv = abs(self.delta_cv)



        J_final = np.sum(((1-Pc/Ref)*10)**2)*self.Qe  + np.sum(abs(self.delta_cv**2))

        print("Qe: ", np.sum(((1-Pc/Ref)*10)**2)*self.Qe , "Pu: ", np.sum(abs(self.de
        if Pc.any() > self.Ref:
            J_final *= 10000


        #print("U_ini: ", u_ini)
        self.state = np.zeros(2)
        return J_final

    def costFunc(self, u):

        # Reverse normalization
        ranges = [(b[1] - b[0]) for b in self.bounds]
        # Normalized values
        u = [u[i] * ranges[i] + self.bounds[i][0] for i in range(3)]



        pGain = u[0]
        FlowSP =  u[1] * self.max_rate
        dGain =  u[2] * self.dt



        cv = self.ControlFunc(state = self.state, pGain= pGain,dGain=dGain,FlowSP=Flo

        #Appending Control Inputs
        self.U.appendleft(cv)
        self.U.appendleft(cv**(3/2))
        self.U.appendleft(cv**(1/2))


        delta_cv = cv - self.last_cv


        if cv > 40: delta_cv = 0
        #Limits rate of change in control input
        if delta_cv > 10:
            delta_cv = 10

        self.delta_cv = np.append(self.delta_cv,delta_cv)
        self.last_cv = cv
        self.state = self.A @ self.state + self.B @ np.array(self.U)




        return self.state

    def simulate(self, u):
        state = [0,0]
        cv_array = [0]
        sp = self.Ref
        pv = [0]
        self.di_cv = 0
        cv = 0
        for i in range(self.n_inputs*self.lag):
            self.U.append(0)


        for i in range(self.Np):
            state = self.A @ state + self.B @ np.array(self.U)

            cv = self.ControlFunc(state = state,pGain= u[0],dGain=u[2],FlowSP=u[1])
            delta_cv = cv - self.last_cv
            #Limiting the rate of change in control input
            #if delta_cv > 25: cv = self.last_cv + 25
            #if delta_cv < -25: cv = self.last_cv - 25
            #Appending Control Inputs
            self.U.appendleft(cv)
            self.U.appendleft(cv**(3/2))
            self.U.appendleft(cv**(1/2))

            cv_array = np.append(cv_array, cv)
            pv = np.append(pv, state[0])
            sp = np.append(sp, self.Ref)
            self.last_cv = cv

        plt.plot(cv_array, label='Control')
        plt.plot(pv, label='PV')
        plt.plot(sp, label='SP')
        plt.title("Simulated Control and Process Variables for the Optimal Control Pr
        plt.xlabel('Time [sec]')
        plt.ylabel('Weight [kg]')
        ticks = np.arange(0, len(cv_array)//10, 2)
        plt.xticks(np.arange(0,len(cv_array)-1, 20), (ticks).astype(int))

        plt.legend()
        plt.show()



    def minimize(self, ref, A, B, pGain,dGain,FlowSP):

        self.Ref = ref
        self.A = A
        self.B = B


        self.bounds = [(1, 20), (0.3, 1), (0.005, 0.02)]

        # Normalize bounds to [0, 1]
        self.normalized_bounds = [(0, 1) for _ in self.bounds]
        u_0 = np.zeros((3))

        if pGain == 0:#pGain
            u_0[0] = 0.5
        else:
            u_0[0] = pGain
```

```
if FlowSP == 0: #FlowSP
    u_0[1] = 0.5
else:
    u_0[1] = FlowSP

if dGain == 0:#dGain
    u_0[2] = 0.5
else:
    u_0[2] = dGain

self.di_cv = 0





#Find the maximum rate of change in weight
for i in range(self.lag):
    self.U.append(100)
    self.U.append(100**(3/2))
    self.U.append(100**(1/2))
self.max_rate = (self.A @ self.state + self.B @ np.array(self.U))[1]

#u_final = minimize(self.objective, u_0, bounds=self.normalized_bounds, metho
u_final = minimize(self.objective, u_0, bounds=self.normalized_bounds, method

# Reverse normalization
ranges = [(b[1] - b[0]) for b in self.bounds]
# Normalized values
print("Unscaled U: ", u_final.x)
u = [u_final.x[i] * ranges[i] + self.bounds[i][0] for i in range(3)]

u[1] *= self.max_rate
u[2] *= self.dt
print("U_final: ", u)
self.simulate(u)
return u
```

Config file for final Implementation

```
{
"Watchdog": "C8112_DosingControlWatchDog.Reset",
"DosingUnits": [
    {
        "UnitName": "C8112_WB01_unit",
        "Weight": "C8112_WB01_WT01_Data.CYCIN.Gross",
        "WeightDiff": "C8112_WB01_WT01_DervLPF.Out",
        "KalmanWeight": "C8112_WB01_KalmanData",
        "Silo": "C8112_WB01_Unit.Y_SiloActive",
        "BatchID": "C8112_WB01_Unit.ID",
        "SP": "C8112_WB01_Unit.Y_QuantSilo",
        "MeasurementNoise": [
            [
                9,
                0.01
            ],
            [
                0.01,
                1
            ]
        ],
        "ProcessNoise": [
            [
                4,
                1
            ],
            [
                1,
                1
            ]
        ],
        "PV": {
            "SiloID": 0.0,
            "SiloIDOld": 0.0,
            "n": 0,
            "ScrewTag": "",
            "InitialWeight": 0.0
        },
        "Screws": [
            {
```

```
        "ScrewName": "C8110_SCW01_M001.YSC",
        "SiloID": 1,
        "Theta": [
            -0.002495102439514986,
            -0.0015776501650419752,
            0.011825477724756468,
            0.0023498452779486863,
            -0.0009487540601615006,
            0.052136755766226714
        ]
    },
    {
        "ScrewName": "C8110_SCW02_M001.YSC",
        "SiloID": 2,
        "Theta": [
            0.1707405019705703,
            0.029926863270879324,
            -0.3269583453032653,
            0.015182018425026242,
            -0.06437450457371391,
            0.0006820000000000001
        ]
    },
    {
        "ScrewName": "C8110_SCW03_M001.YSC",
        "SiloID": 1.5,
        "Theta": [
            0.009206768957260447,
            -0.004068979160261311,
            0.014925534378022928,
            0.010387653069092785,
            0.004268838962954644,
            0.021264853160628247
        ]
    },
    {
        "ScrewName": "C8110_SCW04_M001.YSC",
        "SiloID": 4,
        "Theta": [
            0.1707405019705703,
            0.029926863270879324,
            -0.3269583453032653,
            0.015182018425026242,
            -0.06437450457371391,
            0.0006820000000000001
        ]
    },
    {
        "ScrewName": "C8110_SCW05_M001.YSC",
        "SiloID": 5,
        "Theta": [
            0.1707405019705703,
            0.029926863270879324,
            -0.3269583453032653,
            0.015182018425026242,
            -0.06437450457371391,
            0.0006820000000000001
        ]
    },
    {
        "ScrewName": "C8110_SCW06_M001.YSC",
        "SiloID": 6,
        "Theta": [
            0.1707405019705703,
            0.029926863270879324,
            -0.3269583453032653,
            0.015182018425026242,
            -0.06437450457371391,
            0.0006820000000000001
        ]
    },
    {
        "ScrewName": "C8110_SCW07_M001.YSC",
        "SiloID": 7,
        "Theta": [
            -0.003216650045072024,
            -0.004030285388960714,
            0.03033389337047867,
            -0.00475521758702603,
            0.0021903011489740726,
            0.03518977832092959
        ]
    },
    {
        "ScrewName": "C8110_SCW08_MO01.YSC",
        "SiloID": 8,
        "Theta": [
            0.1707405019705703,
            0.029926863270879324,
            -0.3269583453032653,
            0.015182018425026242,
            -0.06437450457371391,
            0.0006820000000000001
        ]
    },
    {
        "ScrewName": "C8110_SCW09_MO01.YSC",
        "SiloID": 9,
        "Theta": [
            0.00864614802126853,
            -0.004877638394895037,
            0.040336181696511195,
            0.008357038867047759,
            0.0015027863779251902,
            0.04074715212125624
        ]
    },
    {
        "ScrewName": "C8110_SCW10_MO01.YSC",
        "SiloID": 10,
        "Theta": [
            0.0160377236204651,
            -0.01171384014939594,
            0.044927914510319515,
            -0.00942593455057121,
            0.012021058066144931,
            -0.02775360835095592
        ]
    },
    {
        "ScrewName": "C8110_SCW13_MO01.YSC",
        "SiloID": 13,
        "Theta": [
            0.1707405019705703,
            0.029926863270879324,
            -0.3269583453032653,
            0.015182018425026242,
            -0.06437450457371391,
            0.0006820000000000001
        ]
    },
    {
        "ScrewName": "C8110_SCW14_MO01.YSC",
        "SiloID": 14,
        "Theta": [
            0.1707405019705703,
            0.029926863270879324,
            -0.3269583453032653,
            0.015182018425026242,
            -0.06437450457371391,
            0.0006820000000000001
        ]
    },
    {
        "ScrewName": "C8110_SCW15_MO01.YSC",
        "SiloID": 15,
        "Theta": [
            0.1707405019705703,
            0.029926863270879324,
            -0.3269583453032653,
            0.015182018425026242,
            -0.06437450457371391,
            0.0006820000000000001
        ]
    },
    {
        "ScrewName": "C8110_SCW16_MO01.YSC",
        "SiloID": 16,
        "Theta": [
            0.005892569880441699,
            -0.006322069971621551,
            0.042335066190921186,
            -0.0020497930216585086,
            0.004031305395606848,
            0.012554267613444067
        ]
    },
    {
        "ScrewName": "C8110_SCW17_M001.YSC",
        "SiloID": 17,
        "Theta": [
            0.010276930434759586,
            -0.010192472217480751,
            0.043057718266496216,
            -0.007787469751224188,
            0.010573572287628596,
            -0.019604459257330974
        ]
    },
    {
        "ScrewName": "C8110_SCW18_M001.YSC",
        "SiloID": 18,
        "Theta": [
            0.1707405019705703,
            0.029926863270879324,
            -0.3269583453032653,
            0.015182018425026242,
            -0.06437450457371391,
            0.0006820000000000001
        ]
    },
    {
        "ScrewName": "C8110_SCW19_M001.YSC",
        "SiloID": 19,
        "Theta": [
            0.1707405019705703,
            0.029926863270879324,
            -0.3269583453032653,
            0.015182018425026242,
            -0.06437450457371391,
            0.0006820000000000001
        ]
    },
    {
        "ScrewName": "C8110_SCW20_M001.YSC",
        "SiloID": 20,
        "Theta": [
            0.1707405019705703,
            0.029926863270879324,
            -0.3269583453032653,
                0.015182018425026242,
                -0.06437450457371391,
                0.0006820000000000001
            ]
        },
        {
            "ScrewName": "C8110_SCW23_M001.YSC",
            "SiloID": 23,
            "Theta": [
                0.1707405019705703,
                0.029926863270879324,
                -0.3269583453032653,
                0.015182018425026242,
                -0.06437450457371391,
                0.0006820000000000001
            ]
        },
        {
            "ScrewName": "C8110_SCW24_M001.YSC",
            "SiloID": 24,
            "Theta": [
                0.1707405019705703,
                0.029926863270879324,
                -0.3269583453032653,
                0.015182018425026242,
                -0.06437450457371391,
                0.0006820000000000001
            ]
        },
        {
            "ScrewName": "C8110_SCW25_M001.YSC",
            "SiloID": 25,
            "Theta": [
                0.1707405019705703,
                0.029926863270879324,
                -0.3269583453032653,
                0.015182018425026242,
                -0.06437450457371391,
                0.0006820000000000001
            ]
        }
]
```

```
    },
    {
        "UnitName": "C8142_WB01_Unit",
        "Weight": "C8142_WB01_WT01.Y",
        "WeightDiff": "C8142_WB01_WT01_DervLPF.Out",
        "Silo": "C8142_WB01_Unit.Y_SiloActive",
        "BatchID": "C8142_WB01_Unit.ID",
        "KalmanWeight": "C8142_WB01_KalmanData",
        "SP": "C8142_WB01_Unit.Y_QuantSilo",
        "MeasurementNoise": [
            [
                1,
                0.005
            ],
            [
                0.005,
                4
            ]
        ],
        "ProcessNoise": [
            [
                0.002,
                0.001
            ],
            [
                0.001,
                1e-05
            ]
        ],
        "PV": {
            "SiloID": 0.0,
            "SiloIDOld": 0.0,
            "n": 0,
            "ScrewTag": "",
            "InitialWeight": 0.0
        },
        "Screws": [
            {
                "ScrewName": "C8140_SCW55_MO01.YSC",
                "SiloID": 55,
                "Theta": [
                    0.1707405019705703,
            0.029926863270879324,
            -0.3269583453032653,
            0.015182018425026242,
            -0.06437450457371391,
            0.0006820000000000001
        ]
    },
    {
        "ScrewName": "C8140_SCW56_MO01.YSC",
        "SiloID": 56,
        "Theta": [
            0.1707405019705703,
            0.029926863270879324,
            -0.3269583453032653,
            0.015182018425026242,
            -0.06437450457371391,
            0.0006820000000000001
        ]
    },
    {
        "ScrewName": "C8140_SCW57_MO01.YSC",
        "SiloID": 57,
        "Theta": [
            5.765903503975951e-08,
            2.3063613991865945e-06,
            3.646677564747566e-07,
            5.321900440917621e-07,
            2.1287601761641725e-05,
            3.365865374506283e-06
        ]
    },
    {
        "ScrewName": "C8140_SCW58_MO01.YSC",
        "SiloID": 58,
        "Theta": [
            3.892094670101508e-06,
            -6.03058248084032e-06,
            1.2962750579353528e-05,
            3.187932150215732e-05,
            3.693764658587133e-06,
            0.00010978742851907179
        ]
        },
        {
            "ScrewName": "C8140_SCW59_MO01.YSC",
            "SiloID": 59,
            "Theta": [
                -1.3577206938101738e-07,
                -4.073162081683406e-06,
                -7.436542508323886e-07,
                8.108612440616349e-07,
                2.4325837321509696e-05,
                4.441269943733859e-06
            ]
        },
        {
            "ScrewName": "C8140_SCW60_MO01.YSC",
            "SiloID": 60,
            "Theta": [
                3.0762950168735665e-06,
                -6.075512833564203e-06,
                9.744274198303647e-06,
                5.9158673522543156e-05,
                1.3438422892799903e-06,
                0.0001959237907113205
            ]
        }
    ]
},
{
    "UnitName": "C8142_WB02_Unit",
    "Weight": "C8142_WB02_WT01.Y",
    "WeightDiff": "C8142_WB02_WT01_DervLPF.Out",
    "Silo": "C8142_WB02_Unit.Y_SiloActive",
    "BatchID": "C8142_WB02_Unit.ID",
    "KalmanWeight": "C8142_WB02_KalmanData",
    "SP": "C8142_WB02_Unit.Y_QuantSilo",
    "MeasurementNoise": [
        [
            2,
            0.01
        ],
        [
            0.01,
            1
        ]
    ],
    "ProcessNoise": [
        [
            0.02,
            0.001
        ],
        [
            0.001,
            0.0001
        ]
    ],
    "PV": {
        "SiloID": 0.0,
        "SiloIDOld": 0.0,
        "n": 0,
        "ScrewTag": "",
        "InitialWeight": 0.0
    },
    "Screws": [
        {
            "ScrewName": "C8140_SCW61_MO01.YSC",
            "SiloID": 61,
            "Theta": [
                0.1707405019705703,
                0.029926863270879324,
                -0.3269583453032653,
                0.015182018425026242,
                -0.06437450457371391,
                0.0006820000000000001
            ]
        },
        {
            "ScrewName": "C8140_SCW62_MO01.YSC",
            "SiloID": 62,
            "Theta": [
                7.408707706045982e-08,
                -3.970099324001589e-05,
                -2.302937092276856e-06,
                0.0005386634584901175,
                -0.00010549195799374131,
            0.0018994567606023672
        ]
    },
    {
        "ScrewName": "C8140_SCW63_M001.YSC",
        "SiloID": 63,
        "Theta": [
            0.1707405019705703,
            0.029926863270879324,
            -0.3269583453032653,
            0.015182018425026242,
            -0.06437450457371391,
            0.0006820000000000001
        ]
    },
    {
        "ScrewName": "C8140_SCW64_M001.YSC",
        "SiloID": 64,
        "Theta": [
            0.1707405019705703,
            0.029926863270879324,
            -0.3269583453032653,
            0.015182018425026242,
            -0.06437450457371391,
            0.0006820000000000001
        ]
    },
    {
        "ScrewName": "C8140_SCW65_M001.YSC",
        "SiloID": 65,
        "Theta": [
            0.1707405019705703,
            0.029926863270879324,
            -0.3269583453032653,
            0.015182018425026242,
            -0.06437450457371391,
            0.0006820000000000001
        ]
    },
    {
        "ScrewName": "C8140_SCW66_M001.YSC",
        "SiloID": 66,
                    "Theta": [
                        0.0003736169056408029,
                        -0.0001429996494434228,
                        0.0013129511462195483,
                        0.00023158570287318367,
                        -1.6339659774816358e-05,
                        0.0008185014571852631
                    ]
                },
                {
                    "ScrewName": "C8140_SCW67_M001.YSC",
                    "SiloID": 67,
                    "Theta": [
                        0.1707405019705703,
                        0.029926863270879324,
                        -0.3269583453032653,
                        0.015182018425026242,
                        -0.06437450457371391,
                        0.0006820000000000001
                    ]
                },
                {
                    "ScrewName": "C8140_SCW68_M001.YSC",
                    "SiloID": 68,
                    "Theta": [
                        0.1707405019705703,
                        0.029926863270879324,
                        -0.3269583453032653,
                        0.015182018425026242,
                        -0.06437450457371391,
                        0.0006820000000000001
                    ]
                }
            ]
        }
    ]
}
```