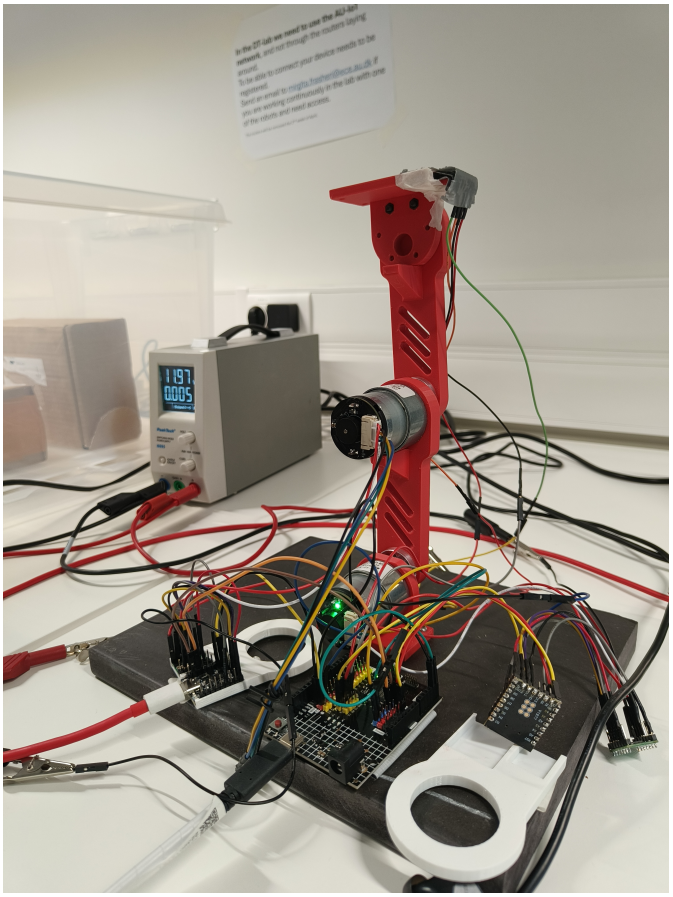
AUTOMATIC TUNING OF MULTIPLE PID MOTOR CONTROLLERS FOR A SELF-BALANCING PLATFORM VIA MACHINE LEARNING



ADRIAN ANTHONY
Aarhus University
Aarhus, Denmark
202205347@POST.AU.DK

SUPERVISOR:
Assoc. Prof. Danish Shaikh
Department of Electrical and Computer Engineering
Aarhus University
Aarhus, Denmark
DANISH.SHAIKH@ECE.AU.DK

Acknowledgement

For researching and developing the outline for the project, a few acknowledgements must be made.

I would like to thank Peter L. Almvig (202006421@post.au.dk) and Simon A. Andreasen (202207041@post.au.dk) for collaborating on the Relay feedback auto-tuning case study [1], and being part of my first auto tuning experience.

Also a big thank you to Christian Porsmose for letting me borrow his copy of *'Adaptive control second edition'* by Åström and Wittenmark [2] for the project.

Lastly, a big thank you to Assoc. Prof. Danish Shaikh for supervising the project and providing the literature for ICO learning.

Abstract

This project studies the feasibility of employing a Hebbian-based machine learning algorithm, input correlation (ICO) learning, for online auto tuning of multiple PID controllers in a self-balancing robotic platform environment. Effectiveness of classic PID auto tuning methods, such as Ziegler-Nichols and gain scheduling, is limited by lengthy system analysis to deduce models or offline calibration. To navigate around these constraints, ICO learning is implemented to dynamically estimate PID-gain parameters by utilizing time correlations between delayed predictive and current reflexive input sensor signals. Algorithm tests are implemented in both offline simulation and online embedded hardware environments, with filtering techniques introduced to improve stability and robustness against disturbances and nonlinear gravitational dynamics. An offline simulation framework is developed using MATLAB, and also an online deployment on an embedded system, controlling a robotic arm with dual motor actuation, is used. The offline simulation tests showed promising results, demonstrating the algorithm's capacity to adapt to nonlinearities and disturbances without requiring prior system modelling, thereby validating ICO learning as a viable method for intelligent control in complex, nonlinear systems running multiple PID controllers. In regards to the results of the embedded system tests, hardware limitations made the results of the specific online implementation questionable. However, the implications also suggest potential for scalable, low-latency, and hardware-efficient PID tuning in real-world robotic applications.

Resumé

Dette projekt undersøger anvendelsen af en Hebbian-baseret machine learning algoritme, input correlation (ICO) læring, til automatisk at tune PID-regulatorer i realtid for en selvbalancerende robotplatform. Effektiviteten af metoder i klassisk kontrolteori, såsom Ziegler-Nichols og gain scheduling, er begrænset af systemanalyser, som udleder systemmodeller, eller af længerevarende kalibreringsprocesser i ikke-realtid. For at undgå problemet er ICO-læring implementeret, således at det dynamisk er i stand til at estimere PID-gain parametre via signalsammenligning i tid mellem et forudsigende signal og et forsinket reflekssignal. Testscenarier af algoritmen er implementeret i både simuleringer i ikke-realtid og indlejrede hardwaremiljøer i realtid, hvor filtreringsteknikker bruges til at styrke stabilitet og robusthed imod systemforstyrrelser og ulineære gravitationsdynamikker. Simulationsrammer i ikke-realtid er blevet udviklet i MATLAB, og det indlejrede hardwaremiljø er et selv-balancerende robotplatformssystem bestående af to motorer. Simuleringstestene viste lovende resultater og demonstrerede algoritmens evne til at tilpasse sig ulineariteter og forstyrrelser uden at kræve forudgående systemmodellering. Dette validerer ICO-læring som en brugbar metode til intelligent styring i komplekse, ulineære systemer, der bruger flere PID-regulatorer. Med hensyn til resultaterne af de indlejrede systemtests, gjorde hardwarebegrænsninger resultaterne fra den specifikke implementering tvivlsomme. Til gengæld antyder implikationerne også potentiale for skalerbar, lav-latens og hardware-effektiv PID-tuning af robotapplikationer i realtid.

Contents

List of Figures

List of Tables

List of Equations

1 Introduction

In the modern world characterized by ever complex systems, the effective control of these systems is becoming increasingly challenging. Within the field of control theory, the method of controlling systems using PID controllers, originally developed in 1911 [3], is still considered a state of the art. PID controllers are crucial for motor control in robotics, drones and 3D printing. This is particularly relevant in complex mechatronic systems composed of multiple independent motors with interdependent control loops, where each motor requires precise feedback control for the system to function effectively. In such systems, identifying individual transfer functions and manually tuning PID controllers for each feedback loop is both time-consuming and error-prone. A global algorithm capable of online tuning across multiple PID controllers would significantly enhance efficiency, reduce setup time, and improve the overall robustness and reliability of the system. This entails deducing a control method that can adapt to highly nonlinear systems and adjust control parameters in real time based on observed system behavior. This presents a significant challenge for classical control algorithms. Instead, a versatile machine learning approach, referred to as intelligent control, can be used for its ability to accommodate real time nonlinear dynamics while minimizing the need for extensive prior modeling. The specific algorithm investigated in this study is Input Correlation (ICO) learning, a biologically inspired algorithm based on Hebbian learning [4], originally developed by Porr and Wörgötter [5]. This algorithm was chosen since it is designed for disturbance adaption, while having a relatively low computational cost compared to that of other intelligent control algorithms. By leveraging the temporal correlation between predictive and delayed reflexive input signals, the system is expected to adaptively modify control gains to ensure stable and responsive performance, even in the presence of dynamic disturbances. The primary aim is to evaluate the feasibility of integrating ICO learning into a control architecture that simultaneously tunes multiple PID controllers on a self-balancing robotic platform. This evaluation will use both MATLAB simulations and an online embedded hardware system as deployment environments.

1.1 Project objective

The objective of the project is to investigate the feasibility of implementing the ICO learning algorithm for the purpose of PID-gain parameter estimation as a method for non-linear adaptive control. The developed algorithm is expected to operate with minimal delay to prevent instability and remain scalable across all PID controllers in a system. If feasibility is asserted in terms of system stability, the deduced ICO-PID algorithm is to be tested on an inverse pendulum robot arm system. Both simulation- and embedded hardware environments are used as canvases for these tests.

1.2 Literature review

This section presents a literature review on various studied measures for auto tuning PID-gain parameters in systems with different environment dynamics. The literature review is separated into two focus groups, the classic control view and the modern control view. Each of these sections provides a concise analysis of the approaches used for auto-tuning within their respective

domains.

1.2.1 The classic control view

Adaptive control and PID auto tuning algorithms are widely researched topics within the field of control theory. Among common practices for controlling a changing environment is the Ziegler-Nichols method for auto tuning PID-gain parameters in linear systems [6], or creating a gain scheduler for system states [7] [8]. Ziegler-Nichols method is often used in the relay feedback method for PID auto tuning [9], where a calibration phase sends a sequence of pulses into the system. This is used to determine the response of the system, and can be run when it is necessary to adapt new PID-gain parameters because of a new/changed environment dynamic. A study of this method for an initially linear system was implemented and tested prior in a case study [1](see Appendix 4). On the contrary, a gain scheduler is a method for separating the nonlinear dynamics into separate state sections, and then based on a lookup table, apply parameters deduced to handle the system for that specific state. These are well known and used methods, and generally well documented by Åström et al. The general challenges for using these methods are often the need of a potential lengthy and regular calibration process, such as with relay feedback, or conducting a lengthy analysis of the system to deduce PID controllers for each system state when, as an example, deducing gain tables.

1.2.2 The modern control view

Another possibility for control is defining a non-linear state-space equation for the system, and then conducting an equilibrium analysis around an equilibrium point where a linear state-space representation can be found [2]. For this method, small changes in the state vector can be accounted for when designing a controller for the representation. For discrete systems, this non-linear to linear transformation can have it's stability inferred via Schur-Cohn stability criteria [10] to assert that the stability region isn't violated using the PID control law. Alternatively, one can specify a non-equilibrium operating state to be analyzed for Lyapunov stability [2] [11]. This involves defining a Lyapunov function for the error vector, and then ensure that the energy of this error function decreases over time.
While these methods are valid and well-established for controlling an inverse pendulum robotic arm, their effectiveness may deteriorate under sufficiently large or scaled disturbances, as such perturbations can drive the system outside the predefined stability region for which the controllers were originally designed. Another issue is obtaining the transfer function of the controlled system, which may not be available, or too complicated to deduce. Using an intelligent control approach would provide an alternative to avoid these issues, and create a control algorithm for adapting to nonlinearities in real time. This, in general, involves defining a loss function for the error, and then seek to minimize it. One method for this, outlined by Schoen [12], is using an unsupervised "Neuro Fuzzy system". This specifies using a fuzzy rule set for the hidden layers of a neural network (NN). Using fuzzy logic as part of the neural network would provide a smooth transition between the system states and potentially make disturbance adaption more seamless. The inputs could be angle and acceleration for the robot arm, and then passing it through the NN to estimate a control signal. Backpropagation is then used to update the NN weights with

gradient descend, thus minimizing the loss function. Deploying this configuration with an output vector of size 3x1 which contains the adapted PID-gain parameters could be considered a viable option.
A different popular approach within the field of robotics is using Reinforcement Learning (RL) as a control measure for nonlinear systems [13]. Various methods have been researched and tested for implementing the principles of RL for tuning PID controllers. One variation of this algorithm is the proposed MM-MADRL algorithm by Zhang [14], which combines the monkey swarm algorithm (a swarm intelligence algorithm) with deep reinforcement learning. This uses the monkey algorithm [15] to discover candidate PID-gain parameters, and then passing them through the actor-critic NN's, which updates the RL policy, and minimizes the loss function. This method is configured to both run as 'online' learning, and offline simulation learning. Another version of RL auto tuning is proposed by Dogru [16]. This study is also investigating deploying a RL model for online learning, and uses an actor as environment state observer, and depending on the environment inputs, the actors policy and NN is updated, and an action is taken depending on these variables. However, it still has an initial offline tuning phase to spare computational power on the online system.
While the results of both RL-based studies are promising, they share a common limitation: the computational demands of online implementation. Although the use of multiple agents and deep neural networks yields accurate and optimized PID gain parameters, this often comes at the cost of reduced responsiveness in time-critical control scenarios. This cannot be afforded when the system should be responsive and quick when exposed to impulse disturbances.

2 Methods

This section presents the methodology used in deducing the ICO-PID algorithm with accompanying assumptions and system analysis. The first sections are oriented towards describing the ICO learning algorithm and deducing a representation for the PID control architecture with ICO as the core intelligent control principle. Then, the proceeding sections becomes implementation oriented in terms of discretizations, stability analysis, and deduction of filter configuration models.

2.1 ICO learning

The ICO learning model [5] is a learning algorithm based on Hebbian learning [4], developed as an alternative to the Isotropic Sequence Order (ISO) Learning model [17] for usage in control systems. It is deduced as a measure to solve the exponential growth problem of the Hebbian weight update, which is caused by the autocorrelation in the learning rule as described in Eqn. (1).

$$\Delta\rho_j = \mu u_j f(v) \tag{1}$$

where

$$v = \sum_j \rho_j u_j \tag{2}$$

with update rule

$$\rho_{new} = \rho_{old} + \Delta\rho \tag{3}$$

Here, μ is the learning rate, ρ_j is the $j'th$ Hebbian weight, u_j the $j'th$ signal channel, and f is the response function or kernal function. This gives the autocorrelation as $\Delta\rho_j \propto \mu\rho_j u_j f(u_j)$. Thus, even at steady state, the weight terms are growing exponentially, which can cause computational hazards. The ICO learning algorithm seeks to eliminate disturbances in a system by adapting the weights whithout the risk of exponential weight growth. This is done by only processing the inputs, but at different positions in time, and then computing the derivative of the delayed signal (reflex signal) and correlating it with the current time signal (predictive signal). This effectively eliminates the self-reinforcement/autocorrelation problem from the Hebbian model. The result of this input correlation is then used to update the weights. The justification for implementing a delay in the controller and then also adding a time correlation computation, is the ability of detecting impulse disturbances between the algorithm iterations, and then adapt a weight based on the the derivative change. This is a feature obtained from the predictive input signals that precede disturbances, compared to that of the delayed reflex signal. This adapted weight will then force likeness between the signals. When the system is no longer experiencing significant changes in the reflex signal, as an example after a disturbance has been counteracted, the weight update Δw approaches zero, indicating convergence and successful disturbance adaption. Therefore, for ICO learning to be effective, the predictive signal channels must encode behavior that is related to the disturbances imposed on the system. This allows the learning rule to extract a meaningful correlation between the time-separated input dynamics, enabling the controller to act preemptively and reduce future reflex responses.

2.2 Applying ICO learning in control architecture

This section introduces how the learning algorithm can be used in a control architecture. When conducting this analysis with the intention of deployment on hardware, it is vital to identify the available sensors for system feedback. It must be inferred that the inputs to ICO learning is implementation realistic, while also becoming a stable closed loop system. This section therefore also focuses on establishing a realistic mathematical interpretation of the system that can be analyzed for stability.
Once the stability of the closed loop PID system can be inferred based on available sensors the specifics on how the PID-gain parameters should be estimated can be explored, which is done in section 2.6. For the analysis in the following sections, the reflex input will be based on the level error for setpoint $R = 0$ and the predictive input the position of the robot arm as acquired from the inertial measurement unit (IMU). To ensure system stability, all system variables must be available, unless stability is verified through numerical methods. This is often not the case in practical scenarios where the ICO-PID control algorithm is intended to be deployed. A key advantage of the algorithm is that it does not require prior knowledge of the system parameters; instead, it can adapt to the environment and automatically tune the system. It is however a benefit that if the parameters and transfer functions are available, then a model for stability proof has been deduced in this project.

2.2.1 ICO learning mechanism

The output of ICO learning is a weighted summation output, where the weights w_j are updated as part of the learning process using input correlation in time. The correlation is deduced by comparing the derivative of the previous input error $\frac{du_0}{dt}$ with the current sensor value X_j to generate a predictive response. With the learning rate μ, this gives the differential equation for the learning mechanism as

$$\Delta w_j = \mu u_j \otimes \frac{du_0}{dt}, \qquad j \neq 0 \tag{4}$$

And the subsequent update rule as

$$w_{new} = w_{old} + \Delta w \tag{5}$$

Where u_j and u_0 are the filtered input (predictive) and error (reflex) signals respectively.

$$u_0 = h_0(t) * x_0(t) \tag{6}$$
$$u_j = h_j(t) * x_j(t) \tag{7}$$

The weight w_0 is not updated, and is chosen at initialization as a bias. If the input signals are in the same domain and noiseless, the filters can be set such that $h_j = 1$.
Hence, the output equation can be described as the summation

$$u = \sum_{j=0}^{N} w_j u_j \tag{8}$$

For added insight on this structure, an alternate interpretation of the ICO learning mechanism, conducted in a brief study, is available in Appendix 1.
To implement this in a more traditional control architecture, it is transformed with the Laplace transform such that it can be inspected with known control methods and made easier to handle. This gives the transformed update Eqns. (4) & (5) as

$$sw_j = \mu U_j^- sU_0, \qquad j \neq 0 \tag{9}$$

$$w_{j_{new}} = w_{j_{old}} + sw_j \tag{10}$$

and the filtered signals (6) & (7) as

$$U_0 = H_0(s)X_0(s) \tag{11}$$

$$U_j = H_j(s)X_j(s) \tag{12}$$

Note that the expressions for time reversal will henceforth be denoted as $X(-s) = X^-$ and $X(z^{-1}) = X^-$ to shorten the correlation expressions. Combining this into the simplest version with $N = 1$ predictive channels, the ICO learning algorithm will be of the form as seen in Figure 1.

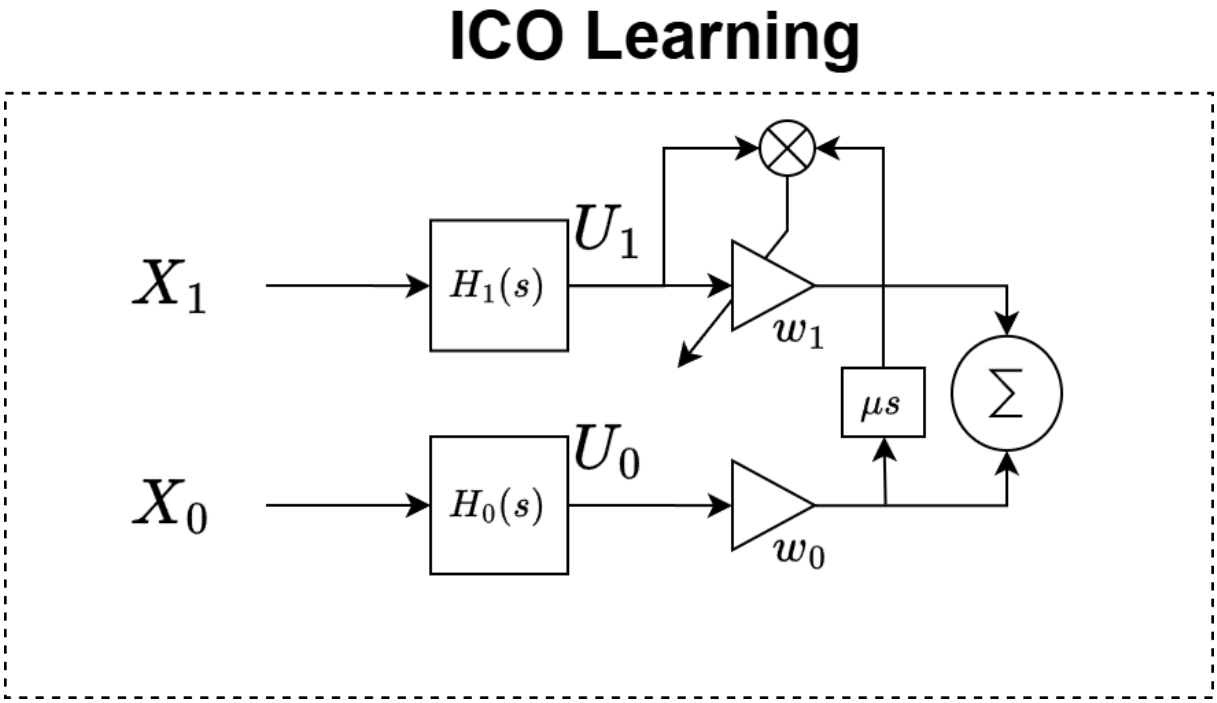


Fig. 1: Simple ICO learning in Laplace domain.

This gives a framework for a controller implementation.

2.2.2 ICO learning for control signal estimation

Applying ICO learning for control signal estimation is a useful tool, which is specifically useful if the system being controlled is complex and with non-nonlinearities. To use the ICO learning as a control mechanism, it should be implemented in a feedback system with a delay to create the predictive dynamic of the weight update for if disturbances are registered. X_0 becomes the delayed error signal and X_1 the predictive signal. The block diagram for this can be seen in Figure 2.

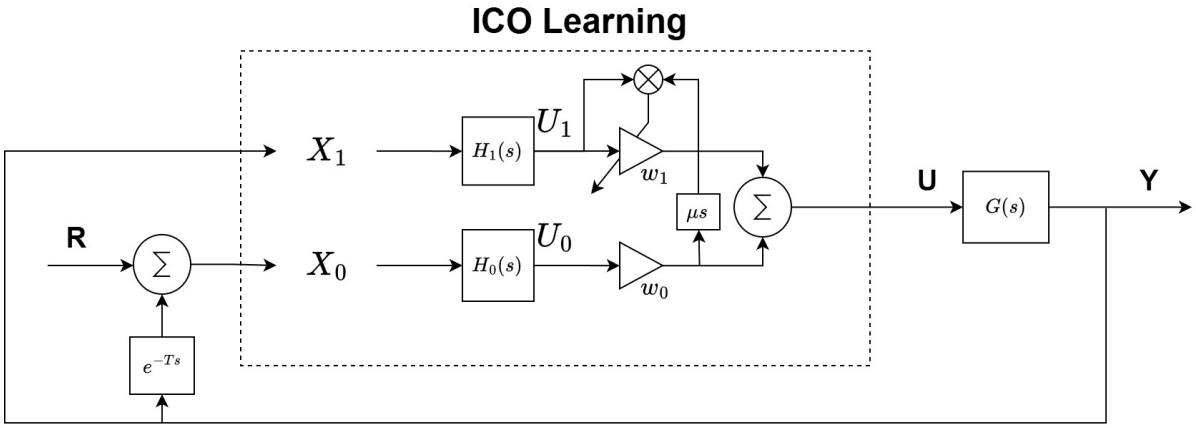


Fig. 2: ICO learning deployed for control of input estimation without disturbance.

Here, in the context of this project, the transfer function $G(s)$ is the linear dynamics of the motor, the sensor signal Y will be the observed IMU/encoder angle of the robot arm, and the reference R the desired angle of the arm. This gives the initial input error as

$$X_0 = Y_{IMU} e^{-Ts} - R \tag{13}$$

$$= G_{motor}(X_0 H_0 w_0) e^{-Ts} - R \tag{14}$$

Where T is the imposed delay on the error signal. In the context of the project, for the robot arm to be level, R is set to 0. Thus

$$X_0 = G_{motor}(X_0 H_0 w_0) e^{-Ts} \tag{15}$$

where solving for X_0 yields

$$X_0 = \frac{0}{1 - G_{motor} H_0 w_0 e^{-Ts}} \tag{16}$$

This yields $X_0 = 0$, resulting in an error of zero. Hence, if no input is applied to the system, it maintains its current state without deviation. If the predictive signal U_1 is added to the summation, assuming that $w_1 \neq 0$, such that

$$X_0 = G_{motor}(X_0 H_0 w_0 + G_{motor} U H_1 w_1) e^{-Ts} \tag{17}$$

where $X_1 = G_{motor}U$. The expression for X_0 becomes

$$X_0 = \frac{G_{motor}G_{motor}UH_1w_1e^{-Ts}}{1 - G_{motor}H_0w_0e^{-Ts}} \tag{18}$$

The expanded error equation can then be combined with the equation of the weight change for the Laplace domain. The initial Laplace transform of the change equation is given in Eqn. (9). Combining that with Eqns. (11) and (18) yields

$$sw_1 = \mu H_1^- G_{motor}^- U^- s \frac{G_{motor}G_{motor}UH_1w_1e^{-Ts}}{1 - G_{motor}H_0w_0e^{-Ts}} H_0 \tag{19}$$

Here, there is an infinite amount of solutions, which means there isn't any inferred learning of the w_1 weight.

2.2.3 ICO learning in a system with disturbance

The Disturbance in the system in the case of the robot arm comes from factors such as imposed impulse disturbances, the torque imposed on the arm by gravity, or even unpredictable design problems with the motor. For the system to recognize these disturbances in this project, they have to be measured through the IMU/encoder sensors, as there is no optimal way of sensing independent system disturbances and simulating them filtered through the environment, such as done by Bernd Porr and Florentin Wörgötter [5]. This configuration can be seen in Figure 3.

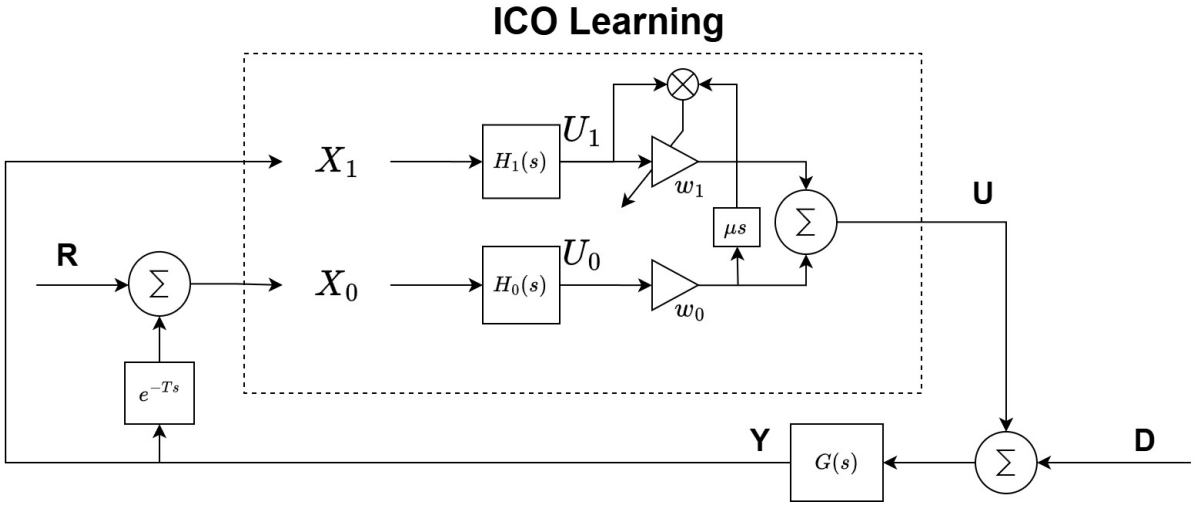


Fig. 3: ICO learning deployed for control signal estimation with disturbance.

When introducing the disturbance into Eqn. (15) it becomes

$$X_0 = G_{motor}(X_0H_0w_0 + D)e^{-Ts} \tag{20}$$

and when solving for X_0, the expression from Eqn. (16) changes such that

$$X_0 = \frac{DG_{motor}e^{-Ts}}{1 - G_{motor}H_0w_0e^{-Ts}} \tag{21}$$

Which allows introduction of F [5] as filter response to $D = 1$

$$F = X_0H_0 \tag{22}$$

$$= \frac{G_{motor}e^{-Ts}}{1 - G_{motor}H_0w_0e^{-Ts}}H_0 \tag{23}$$

When introducing the disturbance into Eqn. (17) it becomes

$$X_0 = G_{motor}(X_0H_0w_0 + G_{motor}(U + D)H_1w_1 + D)e^{-Ts} \tag{24}$$

This updates Eqn. (21)

$$X_0 = \frac{G_{motor}(G_{motor}(U + D)H_1w_1 + D)e^{-Ts}}{1 - G_{motor}H_0w_0e^{-Ts}} \tag{25}$$

which gives a new update equation for weight w_1 in Eqn. (19).

$$sw_1 = \mu H_1^- G_{motor}^-(U^- + D^-)s\frac{G_{motor}(G_{motor}(U + D)H_1w_1 + D)e^{-Ts}H_0}{1 - G_{motor}H_0w_0e^{-Ts}} \tag{26}$$

$$\begin{aligned} &= \mu H_1^- G_{motor}^-(U^- + D^-)sFG_{motor}(U + D)H_1w_1 \\ &+ \mu H_1^- G_{motor}^-(U^- + D^-)sFD \end{aligned} \tag{27}$$

Solving for w_1 yields

$$w_1 = \frac{\mu H_1^- G_{motor}^-(U^- + D^-)FD}{1 - \mu H_1^- G_{motor}^-(U^- + D^-)FG_{motor}(U + D)H_1} \tag{28}$$

$$= W_{1O} \tag{29}$$

This will be denoted as W_{1O} for the learned parameter in ordinary ICO learning. It's apparent the effect the disturbance D has on the weight change, where if $D = 0$, the expression mounts to zero. Eqn. (28) can be analyzed for stability, which is done by Bernd Porr and Florentin Wörgötter [5] in the z-domain. Here, the stability of the transfer function is desired and can be inferred via pole-zero analysis (PZ-analysis), assuming it's for a system where the transfer function G_{motor} and disturbance D is known. Otherwise, continuous time bounded-input bounded-output (BIBO) stability can be inferred. Substituting Eqn. (29) into the output summation, it yields.

$$U = W_{1O}X_1H_1 + w_0X_0H_0 \tag{30}$$

Which if the filter-bank H_j is stable, and W_{1O} is stable, then the output U is stable.

2.2.4 Expanding ICO learning for control signal estimation

By expanding the ICO learning to accommodate more inputs, and by extension, more predictive pathways, the potential for more accurate control parameter estimation is available. This expansion can be in the form of different approaches

- Based on single sensor output Y, augment the k'th input with the k'th filter, to ensure domain comparability.
- Introduce more environmental sensors which yields system output in different domains ($\hat{Y}$) and apply filters. This configuration can be seen in Figure 4.
- Pass the disturbance through the environment in isolation as a direct disturbance learning approach, such as done by Bernd Porr and Florentin Wörgötter [5].
- Utilize a verity of filter banks to introduce temporal adaptiveness in the learning process [17].

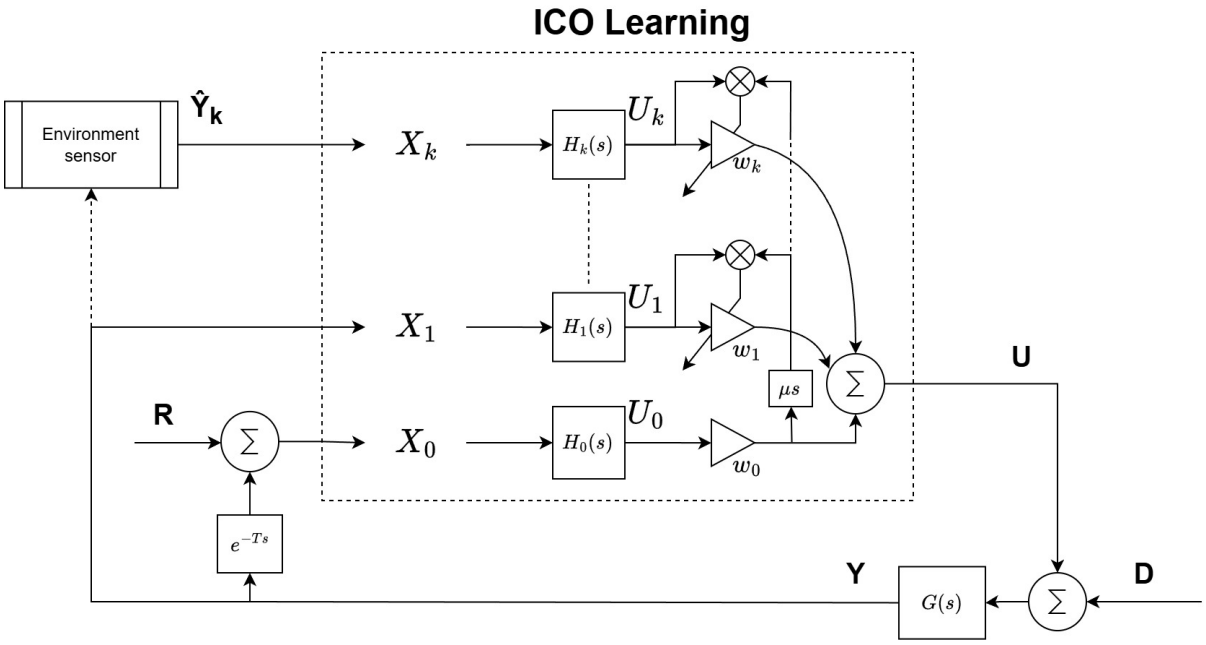


Fig. 4: Expanded ICO learning deployed for control of input estimation with more expanded environmental sensor capabilities.

With this new configuration, Eqn. (21) is rewritten to include the summation of the rest of the N predictive signals.

$$X_0 = \frac{G_{motor}\left(G_{motor}(U+D)\sum_{k=1}^{N} w_k H_k + D\right)e^{-Ts}}{1 - G_{motor}H_0 w_0 e^{-Ts}} \tag{31}$$

This in turn augments Eqn. (26) such that

$$sw_j = \mu H_j^- G_{motor}^- (U^- + D^-) s \frac{G_{motor}\left(G_{motor}(U+D)\sum_{k=1}^{N} w_k H_k + D\right) e^{-Ts} H_0}{1 - G_{motor} H_0 w_0 e^{-Ts}} \tag{32}$$

$$= \mu H_j^- G_{motor}^- (U^- + D^-) sF G_{motor}(U+D) \sum_{k=1}^{N} w_k H_k + \mu H_j sFD \tag{33}$$

Which, when solving for w_j yields

$$w_j = \mu H_j^- G_{motor}^- (U^- + D^-) F \frac{\left(G_{motor}(U+D)\sum_{k=1,k\neq j}^{N} w_k H_k + D\right)}{1 - \mu H_j^- G_{motor}^- (U^- + D^-) F G_{motor}(U+D) H_j} \tag{34}$$

$$= W_{jE} \tag{35}$$

and is denoted as W_{jE} from here on as the correlated parameter w_j in the expanded algorithm. As with Eqn. (28), Eqn. (34) can be evaluated for stability, assuming knowledge about the system it is implemented in is available, or via methods such as BIBO stability.

2.3 Estimating PID parameters via ICO learning

Up until this section, the ICO learning has been deployed for direct control signal estimation. In this section, deploying the algorithm as an adaptive control method for the PID parameters K_p, K_d, K_i is examined. The classic PID controller to be adaptively tuned is seen in Figure 5. This has the potential to provide more accurate control because of the deployment of three ICO models pr. controller. The configurations of the individual model and how they interact with the other models within the controller is therefore something to investigate to ensure convergence, settling time, and required discrete computation

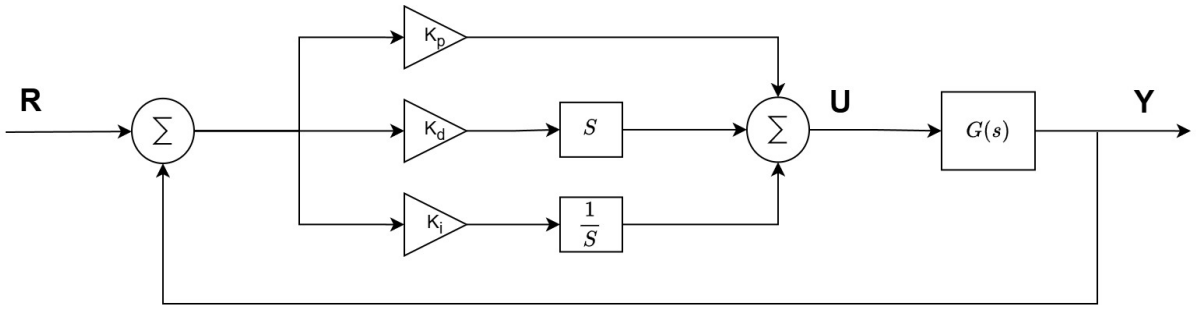


Fig. 5: Classic PID controller blockdiagram representation.

2.3.1 Parallel PID parameter estimation

This configuration for deployment would be a parallel configuration, where each ICO learning model would be deployed like in Figure 3, where they have two pathways U_0, U_1, but instead of

the outputs being the control signal, they would replace the previous PID parameters. This would then be on the form of Figure 6.

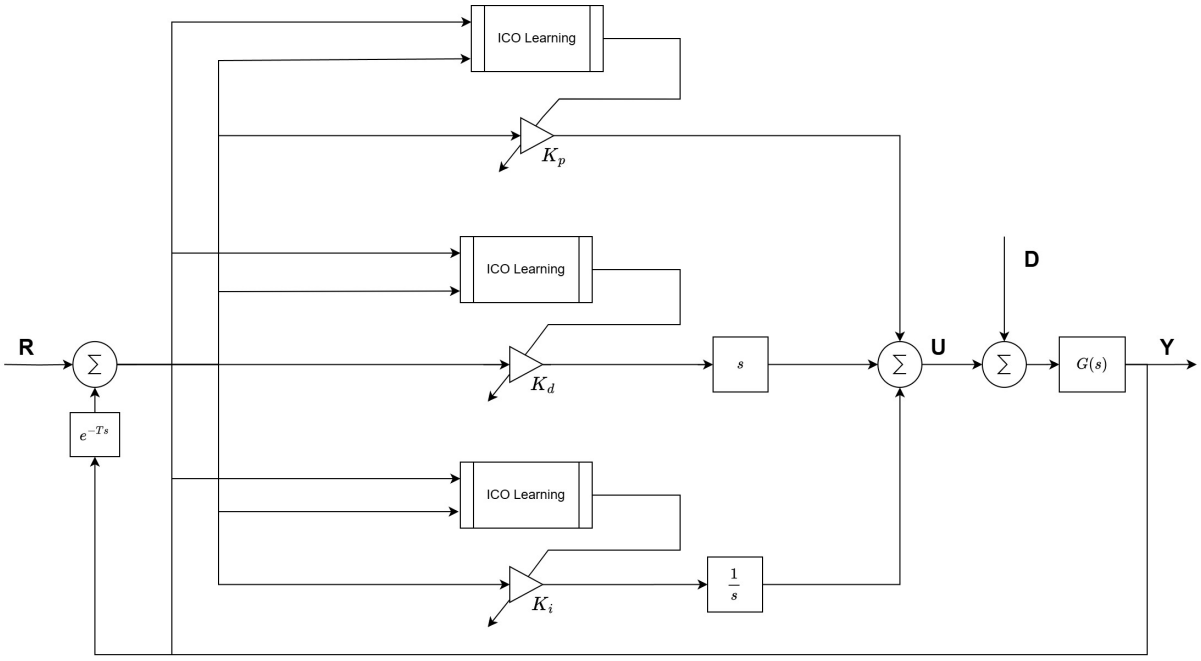


Fig. 6: ICO implementation for learning PID parameters in a parallel configuration.

This would yield the new individual adapted parameters as on the form of output Eqn. (28). The control output of the PID to a given disturbance D would therefore be

$$U = \left(L + Ls + L\frac{1}{s}\right)(Ye^{-Ts} - R) \tag{36}$$

where the learned parameter(s) L is given by.

$$L = W_{1O}YH_1 + w_0(Ye^{-Ts} - R)H_0 \tag{37}$$

with an updated interpretation of the initial filter response $F \in W_{1O}$ to accommodate the new control loop structure

$$F = \frac{G_{motor}e^{-Ts}}{1 - G_{motor}H_0\left(w_{0K_p} + w_{0K_i}\frac{1}{s} + w_{0K_d}s\right)e^{-Ts}}H_0 \tag{38}$$

When inspected, this leads to an interesting dynamic where if the same model is deployed, with the same filters and bias weight w_0, the input correlation will estimate the same parameter value for all three PID parameters because of identical input. While it is possible to define a specific value for all three parameters, it would be a benefit to have the extra degrees of freedom. Deploying different filters for each model could be a possible solution, since they would accommodate

signal behavior of the system to ensure reliable input. Testing with different initial biases would be one approach to accommodate this dynamic. Alternatively, a different input from an environment sensor can be used if possible from a practical perspective, such as the input for the predictive channels being of the form of the PID parameter estimated, I.e, the K_i model as an example, would have $X_{1ki} = \frac{1}{s}E$. This is explored more in Section 2.6.

2.3.2 Cascade PID parameter estimation

Another way of ensuring different outputs for the three ICO learning models and also introducing interaction between them is to deploy them in a cascade format, where the input of the previous ICO model is used for the computation of the next parameter. This means deploying the original ICO learning seen in Figure 3 and the expanded one in Figure 4 within the same PID controller. This means if the models are initialized with the same bias weights w_0 the the outputs won't be identical because of the knowledge about the prior parameters estimated in the loop instance. One possible deployment of this theory can be seen in Figure 7. In that configuration, the control outputs will be composed as

$$K_p = W_{1O}YH_1 + w_0(Ye^{-Ts} - R)H_0 \tag{39}$$

$$K_i = W_{2E}K_dH_{2K_i} + W_{1E}YH_1 + w_0(Ye^{-Ts} - R)H_0 \tag{40}$$

$$K_d = W_{2E}K_pH_{2K_d} + W_{1E}YH_1 + w_0(Ye^{-Ts} - R)H_0 \tag{41}$$

Where $N = 2$ in the expanded ICO model (K_d,K_i) and $N = 1$ in the first ICO model in the cascade order (K_p). Here, as in the previous PID-model $F \in W_{jE}$ is modified in accordance as in Eqn. (38). While the H_1 filters can be identical across the three models the H_2 filter in the K_d and K_i models should differ as they should address the format difference between the IMU/encoder error input of X_1 and the estimated previous parameter such that error differentiation can be preformed on a meaningful function. It is then up for discussion of how to extract meaningful information for correlation with the error signal. This can then be analyzed for stability of each parameter.

2.3.3 PID parameters estimation for system with multiple PID controllers

Once a configuration for the individual PID-controller is decided upon, it is to be deployed in the control architecture of the robot arm.
The decided upon solution for this study is the cascade configuration [18]. The model in Figure 6 is chosen for PID-parameter estimation and deployed in the system. This yields the block diagram seen in Figure 8. This configuration aligns with the error function seen in Appendix 3, Eqn. (86). When configured in this manner, a sensor is required for the feedback such that these dynamics can be compared to that of the 1st PID-controller output. This would therefore be actuated by the encoder for the specified motor.

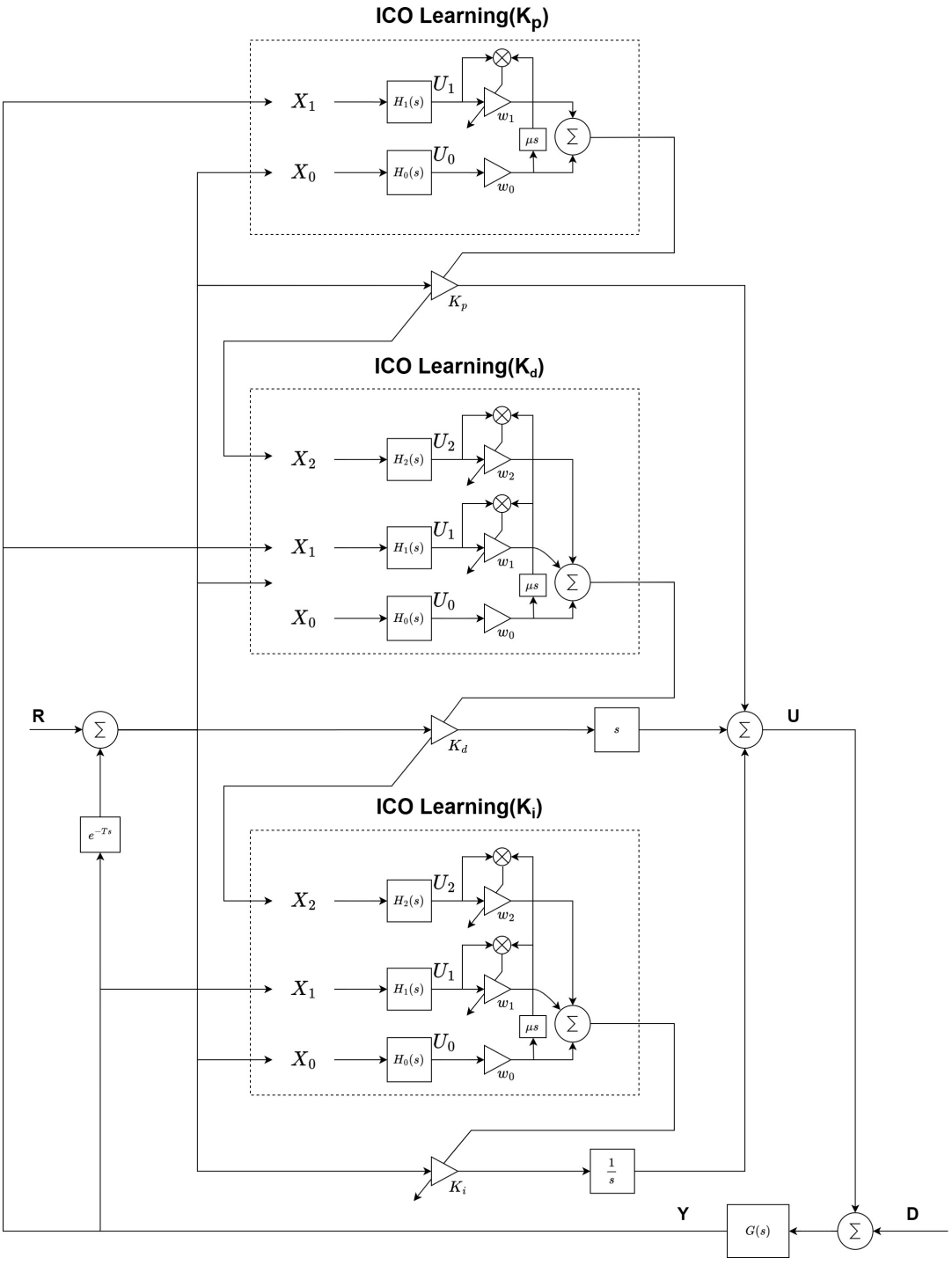


Fig. 7: ICO implementation for learning PID parameters in a cascade configuration.

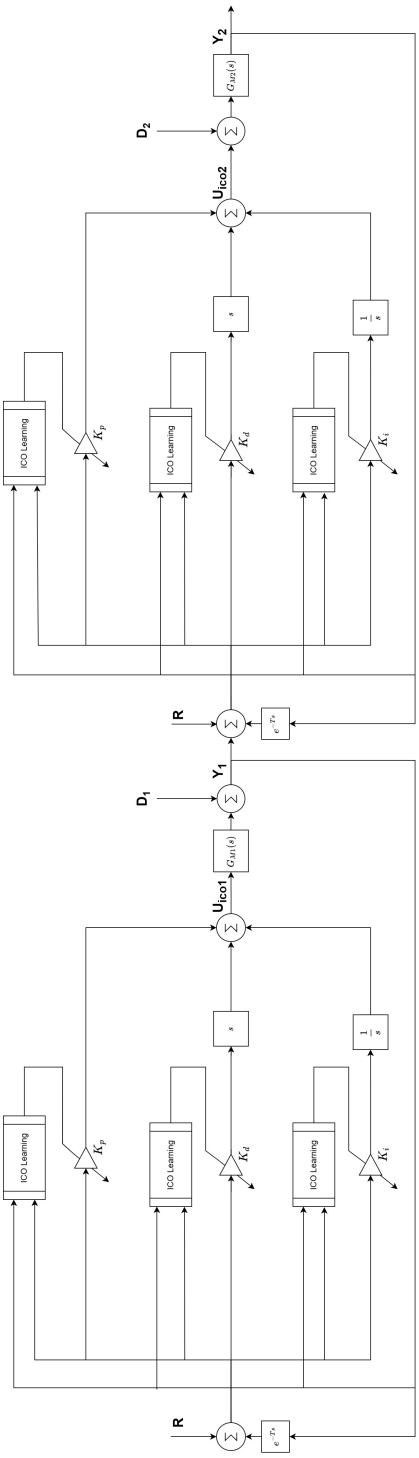


Fig. 8: Double PID controller in cascade configuration with ICO learning for robot arm.

To digitally deploy this architecture, the system is converted from Laplace domain into Z-domain to deduce a digital interpretation of it.
Once converted, it would also open up the possibility for stability analyzes for the digital implementation, where the sampling rate of the micro-controllers is taken into account.

2.4 Discrete ICO-PID learning

When the system is subject to a disturbance D, the system should be stable such that it doesn't diverge. I.e. the filter response of the system should be stable when subject to input. In the context of the ICO learning algorithm, this can be checked by inspecting the output Y_{IMU} to assert that it converges towards a final value for $t->\infty$, or it can be deduced by the learning correlation. If the weights w_j converges to a specific value, then it implies that the system has found the best possible weight such that the signal channels are correlating, and by extension, has accommodated for the disturbances registered in the system. This can be inspected in the continuous Laplace domain, but since the control system is to be deployed on a digital micro controller and implemented with code, it would make sense to transform the algorithm from Laplace to the discrete z-domain and conduct the stability analysis there.

2.4.1 Laplace to z-domain transformation

Converting the algorithms from continuous time to discrete time means considering the sampling frequency of the system. Bernd Porr and Florentin Wörgötter [5] uses a variety of different sampling rates depending on the system it is deployed on, and a thorough system analysis of the most dominant dynamics present. In the case of this project, all the information about the system isn't available, as G_{motor} and the total disturbance D isn't easily identifiable for this system, hence also why the approach of using adaptive learning for system control. This is also a topic for discussion.
There is a lot of different conventions for converting Laplace to z-domain depending on the requirements in regards to metrics such as accuracy and computational load. These methods include ZOH and Tustin transformation to name some common ones. However, in this project, forward Euler is used. This is done because of the easier computational load and also because this is what is used by Bernd Porr and Florentin Wörgötter [5]. This gives the continuous to discrete transformation as

$$\mathcal{Z}\{s\} = \frac{z-1}{T} \tag{42}$$

$$\mathcal{Z}\left\{\frac{1}{s}\right\} = \frac{T}{z-1} \tag{43}$$

Where T is the sampling interval for the transformation. When implementing the system in real time, the usage of backward Euler method is a requirement such that the system is not dependent on future values.

2.4.2 The digital PID-controller

The first step in implementing the algorithm is to deduce the discrete time version of the PID controller, which when applying the transformation from Eqns. (42) and (43) gives the discrete control output as

$$U(z) = \left(K_p + K_d \frac{z-1}{T} + K_i \frac{zT}{z-1}\right) E(z) \tag{44}$$

If substituting $T = 1$ for ease of deduction, and simulating the steps taken by Bernd Porr and Florentin Wörgötter, the algorithm becomes

$$U(z) = \left(K_p + K_d(z-1) + K_i \frac{z}{z-1}\right) E(z) \tag{45}$$

The computational cost in this version is relatively low compared to other implementations. Another version that uses the backward Euler method [19] and using an incremental approach has the form.

$$\begin{aligned} U(z) = \Bigg[K_p \left(\frac{1-z^{-1}}{1-z^{-1}}\right) + \frac{K_p h}{T_i} \left(\frac{1}{1-z^{-1}}\right) + \\ \frac{K_p T_d}{h} \left(\frac{1-2z^{-1}+z^{-2}}{1-z^{-1}}\right)\Bigg] E(z) \end{aligned} \tag{46}$$

Which has greater accuracy yet higher computational cost as well. The original version of this implementation was also intended to include low-pass filtering for noise reduction and initial error reduction for the derivative.
When conducting the inverse z-transform for obtaining the difference equation for the code implementation [20], The following difference equation is obtained

$$u[n] = \mathcal{Z}^{-1}\left\{\left(K_p + K_d(z-1) + K_i \frac{1}{z-1}\right) E(z)\right\} \tag{47}$$

$$= K_p e[n] + K_d(e[n+1] - e[n]) + K_i \sum_{k=-\infty}^{n} e[k] \tag{48}$$

or via backwards Euler

$$u[n] = K_p e[n] + K_d(e[n] - e[n-1]) + K_i \sum_{k=-\infty}^{n} e[k] \tag{49}$$

2.4.3 Transforming ICO learning into digital a interpretation

When the general framework for the discrete PID-controller is decided upon, the next step is to transform the ICO learning algorithm such that it can be incorporated into the discrete PID model for parameter learning. Therefore, for Transforming simple ICO learning, which is used in

the parallel PID parameters estimation approach, the learned parameter(s) L is first transformed such that

$$\mathcal{Z}\{U\} = W_{1O} Y H_1 + w_0 (Y z^{-T} - R) H_0 \tag{50}$$

Where the discrete representation on the correlated parameter $W_1 O$ is

$$\mathcal{Z}\{W_{1O}\} = \frac{\mu H_1^- G_{motor}^- (U^- + D^-) F D}{1 - \mu H_1^- G_{motor}^- (U^- + D^-) F G_{motor} (U + D) H_1} \tag{51}$$

with

$$F = \frac{G_{motor} z^{-T}}{1 - G_{motor} H_0 w_0 z^{-T}} H_0 \tag{52}$$

The block diagram for this can be seen in Figure 9.

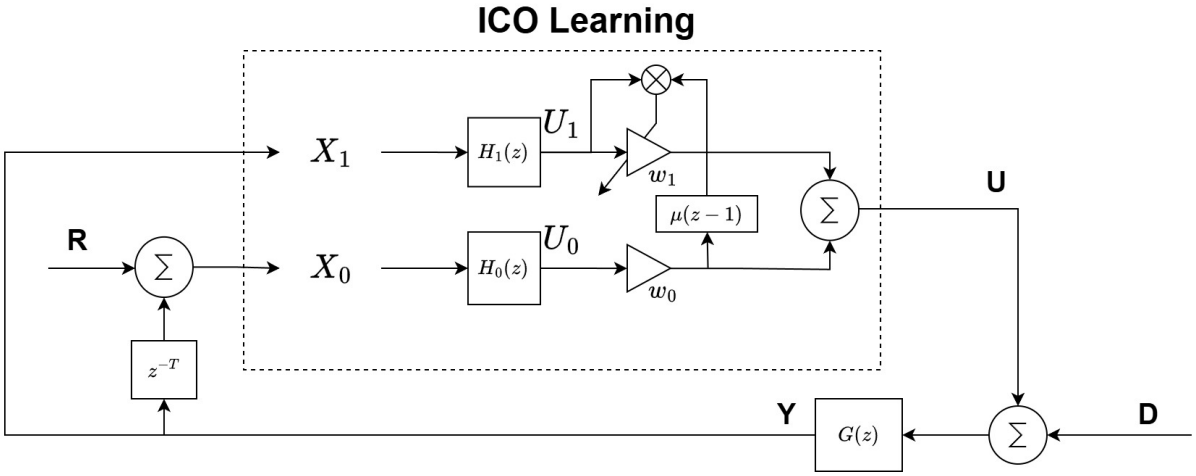


Fig. 9: z-domain interpretation of the ICO learning

2.4.4 Transforming expanded ICO learning into a digital interpretation

When transforming the expanded version, the procedure for transforming simple ICO learning can be more or less replicated.

$$\mathcal{Z}\{U\} = Y \sum_{j=1}^{N} W_{jE} H_j + w_0 (Y z^{-T} - R) H_0 \tag{53}$$

and with the correlated weights for the expanded algorithm as

$$W_{jE} = \mu H_j^- G_{motor}^- (U^- + D^-) F \frac{\left(G_{motor} (U + D) \sum_{k=1, k \neq j}^{N} w_k H_k + D \right)}{1 - \mu H_j^- G_{motor}^- (U^- + D^-) F G_{motor} (U + D) H_j} \tag{54}$$

with the same initial filter function F as Eqn. (52). This version can be seen in Figure 10.

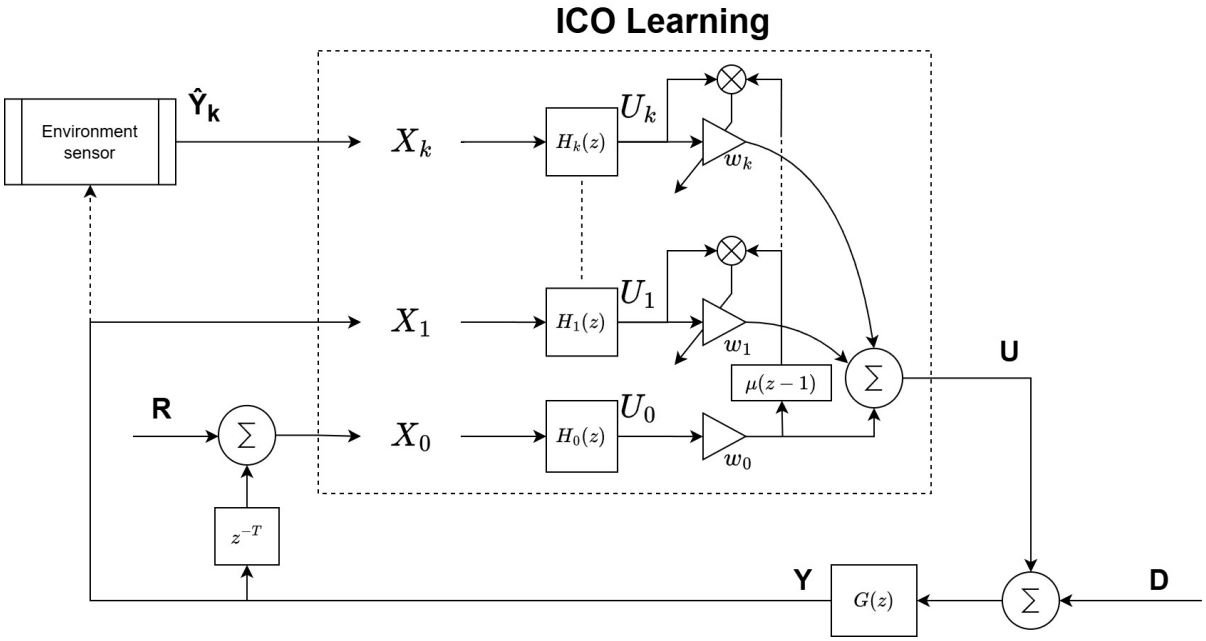


Fig. 10: z-domain interpretation of the expanded ICO learning.

2.4.5 Discrete parallel model

With the discrete output expressions deduced in the previous subsection for both simple and expanded ICO learning, it can be combined with the discrete PID-controller for a total discrete version of the control system. To determine the discrete time version of Eqn. (36). This is done by first transforming the learned parameter(s) L in Eqn. (37), and combining it with the discrete PID from Eqn. (45). This yields.

$$\mathcal{Z}\{L\} = W_{1O} Y H_1 + w_0 (Y z^{-T} - R) H_0 \tag{55}$$

with the control output as

$$U = \left(L + L(z-1) + L\frac{z}{z-1} \right) (Y z^{-T} - R) \tag{56}$$

2.4.6 Discrete cascade model

The discrete cascade model is deduced by transforming Eqns. (39),(40), and (41)

$$K_p = W_{1O} Y H_1 + w_0 (Y z^{-T} - R) H_0 \tag{57}$$

$$\begin{aligned} K_i = {} & W_{2E} K_d H_{2K_i} \\ & + W_{1E} Y H_1 \\ & + w_0 (Y z^{-T} - R) H_0 \end{aligned} \tag{58}$$

$$\begin{aligned} K_d = {} & W_{2E} K_p H_{2K_d} \\ & + W_{1E} Y H_1 \\ & + w_0 (Y z^{-T} - R) H_0 \end{aligned} \tag{59}$$

2.5 Proof of stability

To determine the stability of the learning process, the weight change response to a disturbance in Eqns. (51) & (54) should be investigated.

2.5.1 Stability of digital control systems

A discrete-time system with transfer function

$$H(z) = \frac{A(z)}{B(z)} \tag{60}$$

can be asserted to be stable via traditional pole-zero analysis if it can be identified as linear time invariant (LTI) [21]. However, in Eqns. (51) & (54), because the disturbance D is part of both the numerator $A(z)$ and denominator $B(z)$, they cannot be considered linear digital systems, but nonlinear time-invariant systems (NLTI). Thus, if a numerical solution isn't used, other means of stability inference must be used.

2.5.2 Stability in nonlinear time invariant systems

A discrete-time system is stable if its impulse response $h[n]$ satisfies the summation criterion:

$$\sum_{n=-\infty}^{\infty} |h[n]| < \infty \tag{61}$$

This ensures that the system is BIBO stable. This is true for all discrete systems, and can be used to infer stability within the equations specified in Section 2.5.1. This is the same approach used by Porr and Wörgötter [22], yet still relevant in the context of the altered sensor model. If the system is assumed to be real valued and causal, they can be analysed as seen in the following paragraphs.

Stability of ordinary ICO learning

Eqn. (51) has the numerator and denominator A and B respectively as

$$A = \mu H_1^- G_{motor}^-(U^- + D^-)FD \tag{62}$$
$$B = 1 - \mu H_1^- G_{motor}^-(U^- + D^-)FG_{motor}(U + D)H_1 \tag{63}$$

When inspecting if A is bounded, it is relevant to explore the filter H_0 in F and H_1 used, which in this case is a decaying low-pass Poisson finite input response (FIR) filter, which is described in detail in Section 2.6.4. This gives that for $n > N$, $h[n] = 0$, where N is the filter length, and thus making the response bound. Since this filter format is used for both filters in the filter bank, it will hold for both parts of the summation in A.
The denominator B, if amusing it is a real causal system, can be described as

$$B = 1 - \mu F|G_{motor}(U + D)H_1|^2|_{z=e^{j\omega}} \tag{64}$$

when evaluated on the unit circle to obtain the frequency response [21] [5]. If $\exists\omega \in [0, 2\pi)$ such that the denominator $|B(e^{j\omega})| \to 0$, the system will have a pole on the unit circle, and thus not be stable. As stated Section 2.2, it is not totally possible to deduce stability for the systems, because of the lack of information, which becomes apparent at this step. Since the information is not known about G_{motor}, it cannot be deduced if a pole is added during the learning process. However, a criteria can be set up, which can be tuned to ensure a stable learning process. If a small enough learning rate μ can be tuned, then the system can be forced such that $\max\left(\mu F|G_{motor}(U + D)H_1|^2\right) < 1|_{z=e^{j\omega}}$, and thus ensuring $1 - \mu F|G_{motor}(U + D)H_1|^2 > 0|_{z=e^{j\omega}}$.

Stability of expanded ICO learning

The same procedure can be used for Eqn. (54) as with Eqn. (51). The main difference in the weight change in this scenario is the weighted summation multiplied in the numerator A, and then the respective j'th filter response of the filter bank in the denominator B.
The response of A will in this scenario still be bound because of the filtered system response F and predictive filter H_j being a dominant part of the correlation process, ensuring the response decay for $n \to N$. For ensuring no additional poles are added in B, the same method of tuning μ can be used. However, it may result in not as efficient learning for some of the predictive pathways, because of the use of differing filters. This would be the case for, as an example, the cascade PID configuration in Figure 7, where H_2 could differ from H_0 and H_1.

2.6 Algorithm implementation

Using the framework outlined in the previous sections, the theoretical ICO learning algorithms deduced for the PID control structure can be revised and implemented in a practical manner correlating with the system dynamics of a robot arm with multiple joints. This includes configuring the algorithm not just such that it is stable, but also converges to an error of zero.

2.6.1 Practical interpretation of ICO learning

In the ICO learning method introduced by Porr and Wörgötter [5] they explore the usage of the algorithm for disturbance adaption, where the set point is set to zero. This means that the correlated signals are expressed as

$$\Delta w_j = \mu \left(y \otimes \frac{d(y-r)}{dt} \right) \tag{65}$$

if filtering isn't applied. While this does adapt to disturbances imposed on the system, if a non-zero reference is used, then the system will not necessarily adapt towards an error of zero, and thus having X_1 become non-zero. The signals will have to be identical by nature to assert convergence towards the reference signal. In this case, both channels will have to be an error signal, because that is the system dynamic the ICO learning is meant to control adaptively.

2.6.2 Predictive channels for PID parameter estimation

The respective PID-parameters represents gain values for specific dynamic features in the system. The proportional dynamic (K_p), the integral dynamic (K_i), and the derivative dynamic (K_d). The individual models should thus be adapting based on these specific dynamics, and the predictive channels should therefore reflect that in the inputs X_j. If this is applied to the parallel model described in section 2.3.1, the predictive signals will be of the form

$$X_{1Kp} = E \tag{66}$$

$$X_{1Ki} = E \frac{1}{1-z^{-1}} \tag{67}$$

$$X_{1Kd} = E(1-z^{-1}) \tag{68}$$

when using backwards approach. The resulting block diagram will be of the form as seen in figure 11. Combining this with the block diagram outlined in Figure 8, the complete digital architecture is arrived at, which is in Figure 12. This configuration also asserts that when $E \to 0$, the inferred learning decreases.

2.6.3 Filtering

As will become apparent in section 3, there is a need for filtering of the derivative change in the algorithm to ensure convergence. For general implementation of filtering, there is a few design decisions to be made because of the nature of the online system. The system should be computationally optimized at all time to reduce delays in the algorithm process. This means that after the convolution of signals has been done, a single scalar value should be passed on from the process, instead of the entire convolution output of the signal response of the filter.
The scalar signal chosen from the resulting convolution output should be chosen such that it contains as much relevant information as possible. A signal buffer is utilized to remember the signals from the previous M iterations. This is done such that even though the current iteration receives a single discrete scalar input from the system, the memory retention gives the filtering process the ability to operate on old signals as well. Thus every iteration updates the buffer with

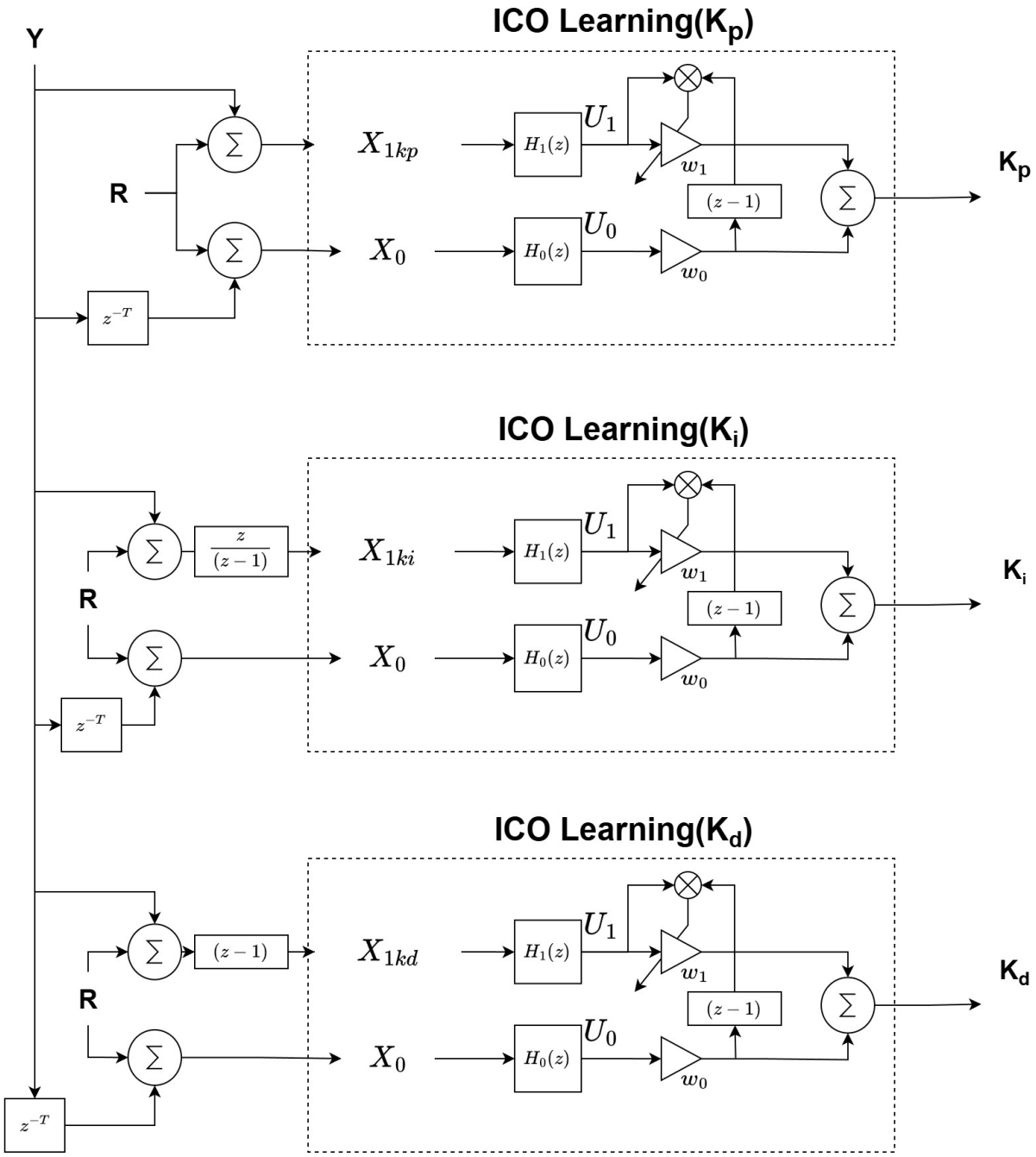


Fig. 11: Blockdiagram of error based ICO learning for PID gain values.

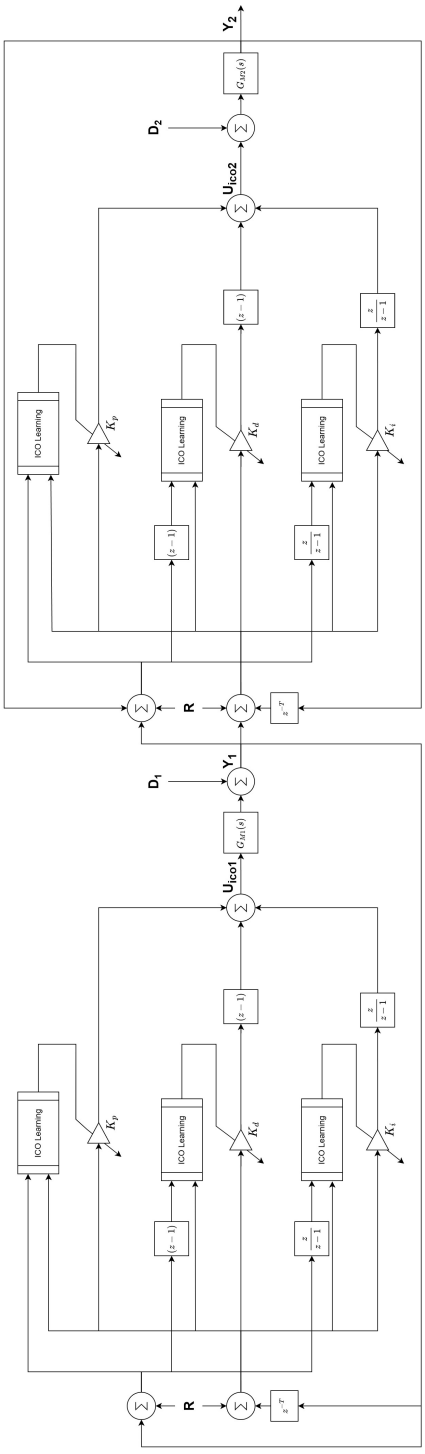


Fig. 12: Block diagram of the cascading digital PID controllers used for the robot arm.

the newest signal scalar at the end of the buffer. The filter used has a length as well, which is given as L. The output of the general linear time-invariant (LTI) filter system is therefore given as

$$Y(z) = X(z)H(z) \tag{69}$$

$$\mathcal{Z}\{y[n]\} = \mathcal{Z}\left\{\sum_{k=0}^{L-1} x[k] \cdot h[n-k]\right\}, \quad n = 0,1,\ldots,L+M-1 \tag{70}$$

Which specifies the full convolution of the system. In the resulting discrete output y, it is hard to visualize which value to pick as the scalar output. If the center value is chosen, then the newest input isn't dominating the response behavior, which is preferred since this is the current reality of the system which should be addressed. If the the last entry is chosen, then the previous samples aren't evaluated, which is still a required dynamic. The solution to this is instead of conducting a full convolution, only the center L indices are calculated, since this is where most of the buffer signals are evaluated at the same time. For this to be effective, it is required that $M = L$, such that no samples are left out of the center range of the filter and no zero padding happens. Then when choosing the last index of the output y, as the scalar, all the previous entries has influence on the signal. Because of the Poisson filtering used, which is further detailed in Section 2.6.4, the dominance of the newest input is favored. This essentially chooses the index $\left\lfloor \frac{(M-1)}{2} + L - 1 \right\rfloor$ from the full convolution. From this point, the specific method for how filtering should be implemented in the ICO-PID algorithm should be inspected. To do this, there are two chosen options.

Option 1 Filtering of derivative learning:

Filtering the behavior of the predictive channel (X_1) in the ICO learning model that determines the K_d gain value, such that the update of this dynamic in PID controller is smoothened and damped. The delayed error (X_0) is also damped with the same filter in all the models. The block diagram for this can be seen in Figure 13.

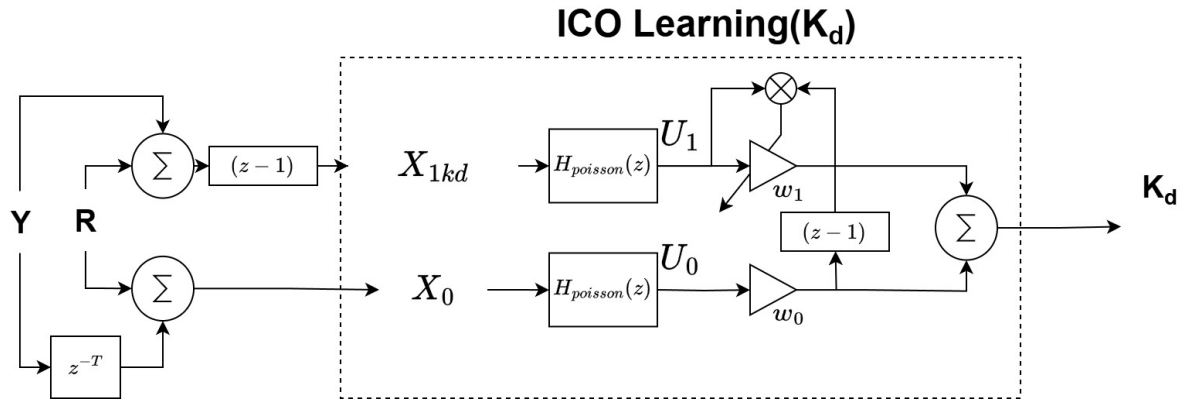


Fig. 13: Filtering of Derivative Model (K_d)

Which gives the system output as

$$\begin{aligned} K_d = w_0 \left(\sum_{k=0}^{L-1} x_0[k] \cdot h_{poisson} \left[\frac{(M-1)}{2} + L - 1 - k \right] \right) \\ + w_1 \left(\sum_{k=0}^{L-1} x_1[k] \cdot h_{poisson} \left[\frac{(M-1)}{2} + L - 1 - k \right] \right) \end{aligned} \tag{71}$$

and the update rule as relatively unaltered

$$\Delta w_1 = \mu \left(u_1[n] \frac{u_0[n] - u_0[n-1]}{dt} \right) \tag{72}$$

When deducing this option while keeping in mind the BIBO stability criteria from Section 2.5.2, here the classic filter format is followed. When using decaying low pass Poisson filters for both channels, the output is bound within the criteria. The pseudocode for this version of the ICO-PID can be seen in Algorithm 1.

Algorithm 1 Filtering of derivative learning

Require: Initial weights w_0, w_{1_kp}, w_{1_ki}, w_{1_kd}; learning rate μ; Poisson kernel $h_{poisson}[\,]$ (or alternate LTI filter) of length L; time step dt

Initialize previous values: $X_0[-1] \leftarrow 0$, $error[-1] \leftarrow 0$

Initialize buffer: $\mathscr{B}_{X_0}, \mathscr{B}_{X_{1\_kd}} \leftarrow \emptyset$

for each time step n **do**

Compute system error (for the robot arm, this is a cumulative joint angles error: $error[n] \leftarrow target - \sum_i joint\_angle_i$)

Compute $X_0[n] \leftarrow error[n-1]$

Compute integral $integral[n] \leftarrow \sum_{k=-\infty}^{n} error[n] \cdot dt$

Compute raw derivative: $\Delta error[n] \leftarrow \frac{(error[n] - error[n-1])}{dt}$

Compute $ICO\_derivative[n] \leftarrow \frac{(X_0[n] - X_0[n-1])}{dt}$

Append $X_0[n]$ to buffer $\mathscr{B}_{X_0}$ (keep latest L elements)

Compute filtered X_0:

$$\hat{X}_0[n] \leftarrow \sum_{k=0}^{L-1} \mathscr{B}_{X_0}[k] \cdot h_{poisson}\left[\frac{(M-1)}{2} + L - 1 - k\right]$$

(implemented as convolution over buffer $\mathscr{B}_{X_0}$)

Compute $X_{1\_kd}[n] \leftarrow error[n] \cdot \Delta error[n]$

Append $X_{1\_kd}[n]$ to buffer $\mathscr{B}_{X_{1\_kd}}$ (keep latest L elements)

Compute filtered X_{1_kd}:

$$\hat{X}_{1\_kd} \leftarrow \sum_{k=0}^{L-1} \mathscr{B}_{X_{1\_kd}}[k] \cdot h_{poisson}\left[\frac{(M-1)}{2} + L - 1 - k\right]$$

(implemented as convolution over buffer $\mathscr{B}_{X_{1\_kd}}$)

Update weights:

$$\begin{aligned}
X_{1_kp}[n] &\leftarrow error[n] \\
\Delta w_{1_kp} &\leftarrow \mu \cdot X_{1_kp}[n] \cdot ICO_derivative[n] \\
w_{1_kp} &\leftarrow w_{1_kp} + \Delta w_{1_kp}
\end{aligned}$$

$$\begin{aligned}
X_{1_ki}[n] &\leftarrow error[n] \cdot integral[n] \\
\Delta w_{1_ki} &\leftarrow \mu \cdot X_{1_ki}[n] \cdot ICO_derivative[n] \\
w_{1_ki} &\leftarrow w_{1_ki} + \Delta w_{1_ki}
\end{aligned}$$

$$\begin{aligned}
\Delta w_{1_kd} &\leftarrow \mu \cdot \hat{X}_{1_kd} \cdot ICO_derivative[n] \\
w_{1_kd} &\leftarrow w_{1_kd} + \Delta w_{1_kd}
\end{aligned}$$

Algorithm 1 Filtering of derivative learning (continued)

15: Compute PID-gain parameters:

$$K_p \leftarrow w_0 \cdot \hat{X}_0[n] + w_{1\_kp} \cdot X_{1\_kp}[n]$$
$$K_i \leftarrow w_0 \cdot \hat{X}_0[n] + w_{1\_ki} \cdot X_{1\_ki}[n]$$
$$K_d \leftarrow w_0 \cdot \hat{X}_0[n] + w_{1\_kd} \cdot \hat{X}_{1\_kd}[n]$$

16: **end for**

Option 2 Filtering of derivative dynamic in ICO learning:

This second method could be considered rather unorthodox compared to the general filtering principles laid out by Porr and Wörgötter, [22], where instead of filtering the input signals to ICO learning, a general smoothening mechanic is introduced before the correlation learning, which also includes the bias weight (w_0). This means that filtering isnt applied for the output summation of the ICO channels, but becomes an integrated smoothening mechanic for the weight update of the ICO model. The blockdiagram for this ICO configuration can be seen on Figure 14.

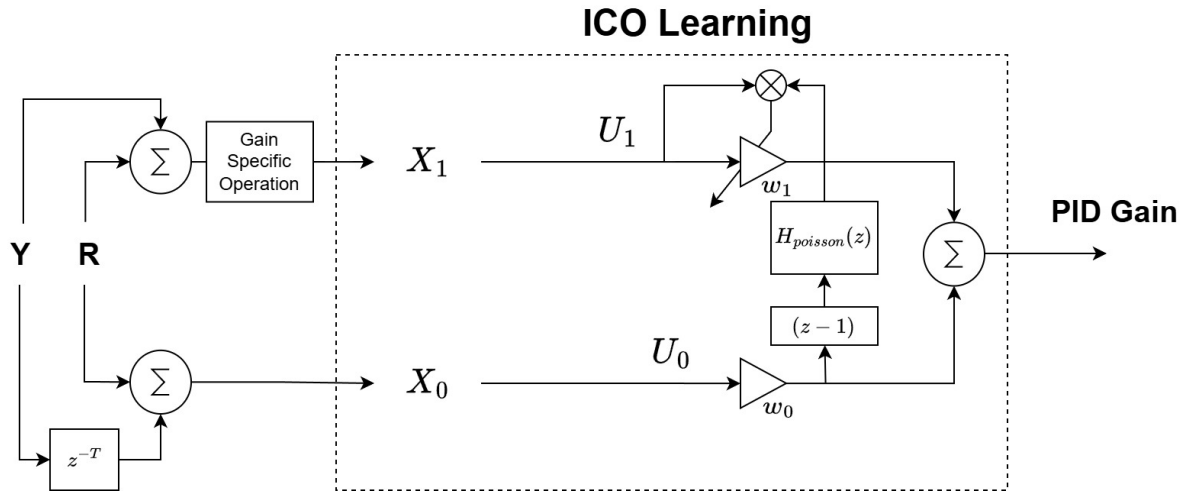


Fig. 14: Filtering of ICO Derivative Dynamic

Which gives the system output as an unfiltered operation

$$PID\_gain = w_0 x_0[n] + w_1 x_1[n] \tag{73}$$

however, the general update rule has been altered such that

$$\Delta w_1 = \mu \left(u_1[n] \left(\sum_{k=0}^{L-1} \frac{u_0[k] - u_0[k-1]}{dt} \cdot h_{poisson} \left[\frac{(M-1)}{2} + L - 1 - k \right] \right) \right) \tag{74}$$

The idea for this configuration arrived from the generally studied appliance of derivative filtering used for generic PID-controllers, both in continuous and discrete domains [19]. Therefore, to filter the derivative behaviors of the ICO algorithm, a similar approach is used, wherein the derivative dynamic of ICO is filtered for smoothening and dampening. The BIBO stability criteria is still upheld, since the multiplication of the decaying low pass Poisson filters to the weight change mechanic in equation 51, also ensures bounded output over time. The pseudocode for this version of the ICO-PID can be seen in Algorithm 2.

Algorithm 2 Filtering of derivative dynamic in ICO learning

Require: Initial weights w_0, w_{1_kp}, w_{1_ki}, w_{1_kd}; learning rate μ; Poisson kernel $h_{poisson}[\,]$ (or alternate LTI filter) of length L; time step dt

Initialize previous values: $X_0[-1] \leftarrow 0$, $error[-1] \leftarrow 0$

Initialize buffer: $\mathscr{B} \leftarrow \emptyset$

for each time step n **do**

  Compute system error (for the robot arm, this is a cumulative joint angles error: $error[n] \leftarrow target - \sum_i joint\_angle_i$)

  Compute $X_0[n] \leftarrow error[n-1]$

  Compute integral $integral[n] \leftarrow \sum_{k=-\infty}^{n} error[n] \cdot dt$

  Compute raw derivative: $\Delta error[n] \leftarrow \frac{(error[n] - error[n-1])}{dt}$

  Compute $ICO\_derivative[n] \leftarrow \frac{(X_0[n] - X_0[n-1])}{dt}$

  Append $ICO\_derivative[n]$ to buffer $\mathscr{B}$ (keep latest L elements)

  Compute filtered derivative:

$$\hat{d}_{ICO}[n] \leftarrow \sum_{k=0}^{L-1} \mathscr{B}[k] \cdot h_{poisson}\left[\frac{(M-1)}{2} + L - 1 - k\right]$$

  (implemented as convolution over buffer)

Algorithm 2 Filtering of derivative dynamic in ICO learning (continued)

11: Update weights:

$$
\begin{aligned}
X_{1\_kp}[n] &\leftarrow error[n] \\
\Delta w_{1\_kp} &\leftarrow \mu \cdot X_{1\_kp}[n] \cdot \hat{d}_{ICO}[n] \\
w_{1\_kp} &\leftarrow w_{1\_kp} + \Delta w_{1\_kp}
\end{aligned}
$$

$$
\begin{aligned}
X_{1\_ki}[n] &\leftarrow error[n] \cdot integral[n] \\
\Delta w_{1\_ki} &\leftarrow \mu \cdot X_{1\_ki}[n] \cdot \hat{d}_{ICO}[n] \\
w_{1\_ki} &\leftarrow w_{1\_ki} + \Delta w_{1\_ki}
\end{aligned}
$$

$$
\begin{aligned}
X_{1\_kd}[n] &\leftarrow error[n] \cdot \Delta error[n] \\
\Delta w_{1\_kd} &\leftarrow \mu \cdot X_{1\_kd}[n] \cdot \hat{d}_{ICO}[n] \\
w_{1\_kd} &\leftarrow w_{1\_kd} + \Delta w_{1\_kd}
\end{aligned}
$$

12: Compute PID-gain parameters:

$$
\begin{aligned}
K_p &\leftarrow w_0 \cdot X_0[n] + w_{1\_kp} \cdot X_{1\_kp}[n] \\
K_i &\leftarrow w_0 \cdot X_0[n] + w_{1\_ki} \cdot X_{1\_ki}[n] \\
K_d &\leftarrow w_0 \cdot X_0[n] + w_{1\_kd} \cdot X_{1\_kd}[n]
\end{aligned}
$$

13: **end for**

2.6.4 Poisson filtering

The band-pass filter used for filtering in the ICO learning model contains the shape of the Poisson distribution. This is used because it resembles patterns observed in the biological learning process, which is also mentioned by Porr and Wörgötter [17]. This combines well with the Hebbian nature of the ICO algorithm, and makes the reactionary response of the robot more organic. The general formula for the Poisson distribution is given as

$$
f(k, \lambda) = P(X = k) = \frac{\lambda^k e^{-\lambda}}{k!} \tag{75}
$$

Where

- k is the number of occurrences.
- X is a discrete random variable.
- λ is a real number and $Var(X)$.

Since the model should not need to model probability, the filter only needs to mimic the form of the Poisson distribution. This is done via. combining two exponential functions, where each determines the decay of high-pass and low-pass frequencies. The Poisson-like 1D band-pass filter

is constructed by subtracting a low-pass exponential decay from a high-pass exponential decay. The filter impulse response $h[n]$ is given by:

$$h[n] = e^{-n/\lambda_{\text{low}}} - e^{-n/\lambda_{\text{high}}}, \quad n = 0, 1, \ldots, N-1 \tag{76}$$

To ensure the filter has unit energy or is otherwise normalized, its defined such that:

$$h[n] := \frac{h[n]}{\sum_{k=0}^{N-1} |h[k]|} \tag{77}$$

The corresponding (LTI) system can be described by the following FIR (finite impulse response) difference equation:

$$y[n] = \sum_{k=0}^{N-1} h[k] \cdot x[n-k] \tag{78}$$

Substituting the definition of $h[k]$, the equation becomes:

$$y[n] = \sum_{k=0}^{N-1} \left(\frac{e^{-k/\lambda_{\text{low}}} - e^{-k/\lambda_{\text{high}}}}{\sum_{j=0}^{N-1} \left| e^{-j/\lambda_{\text{low}}} - e^{-j/\lambda_{\text{high}}} \right|} \right) x[n-k] \tag{79}$$

Here, λ_{low} and λ_{high} are exponential decay parameters that control the high-pass and low-pass behavior of the filter, respectively. The general shape of the distribution is here obtained if $\lambda_{low} > \lambda_{high}$, thus mimicking a low-pass dynamic. N is the filter length, ideally an odd number for symmetry. The filter response effectively results in a signal that is not only damped but also infer change in future samples, To use this with the rest of the MATLAB code, the difference equation is implemented in a filter design function, that returns the filter array h based on the input parameters. As an example of the filters generated by this function, see Figure 15. This guarantees that the disturbances imposed on the system are damped, and their effect spread out on following samples. This effect can now be tested and compared to the unfiltered version.

2.7 Hardware implementation

In regards to hardware, for implementing the algorithm on a real robot arm, the microcontrollers Arduino uno R4 minima and Arduino Nano, made by Arduino™, were used. The IMU used is the BMX160 9 Axis Sensor Module SKU SEN0373 by Bosch Sensortec™, and the motor driver is the DRV8874 Single Brushed DC Motor Driver Carrier by Pololu™. The motors used are the JGB37-520 12V 178RPM. The frames for the robot arm are 3D printed. These parts were all acquired via the Faculty of Electrical and Computer Engineering (ECE) at Aarhus University.
Pictures of the robot at 0 degrees level can be seen on Figures 16 & 17.

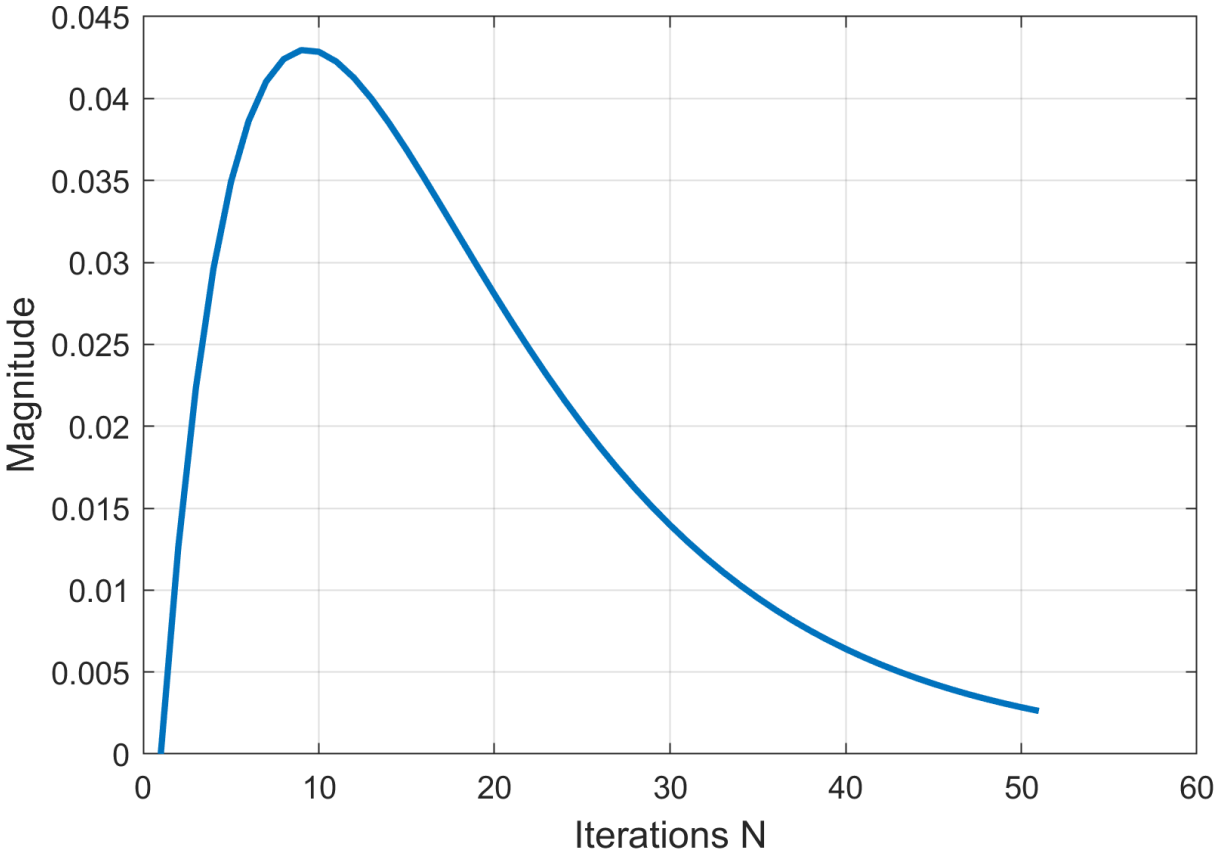


Fig. 15: Example of Poisson filter with parameters $N = 51, \lambda_{low} = 12, \lambda_{high} = 6$.

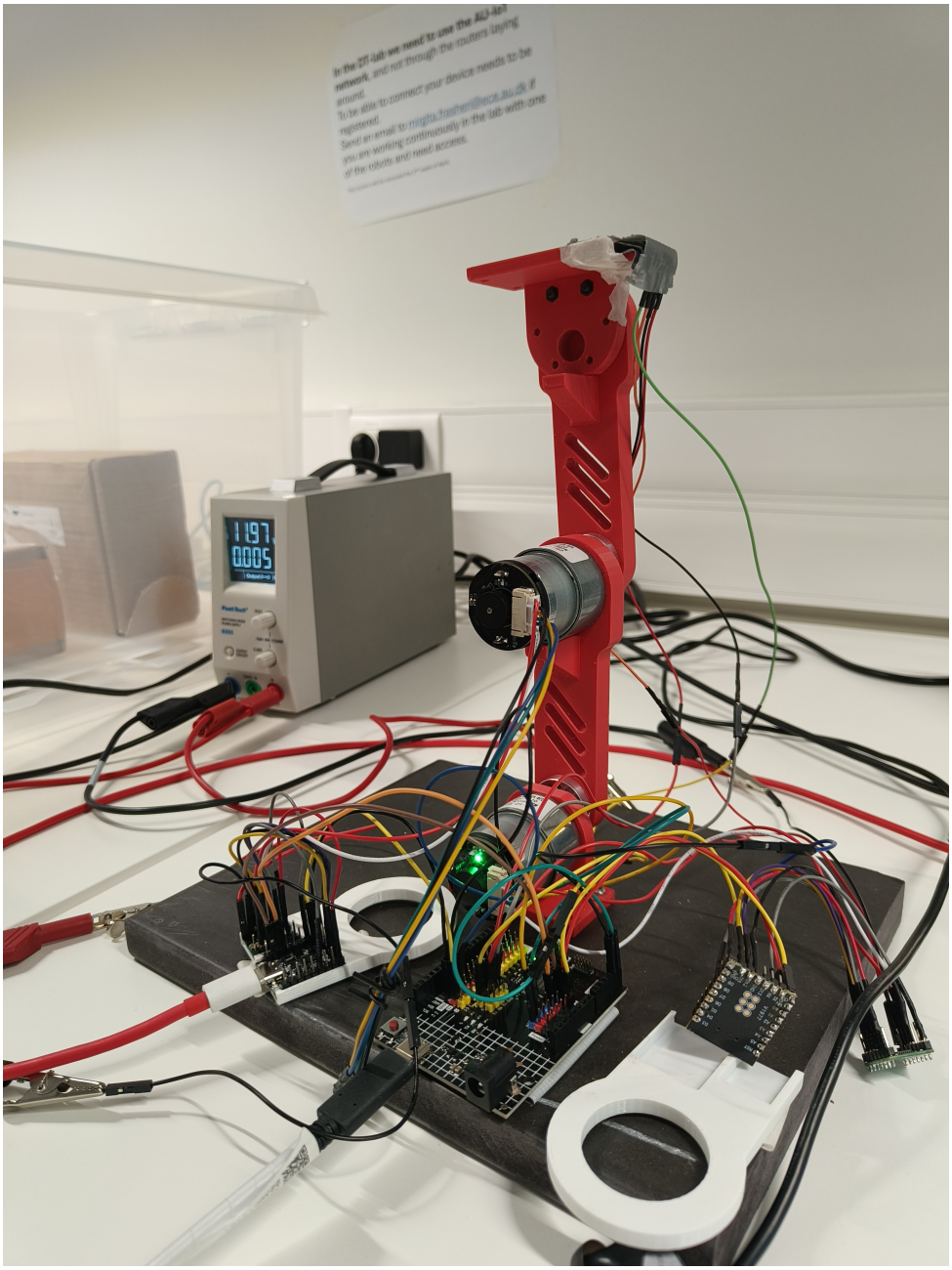


Fig. 16: Frontal view of the robot arm.

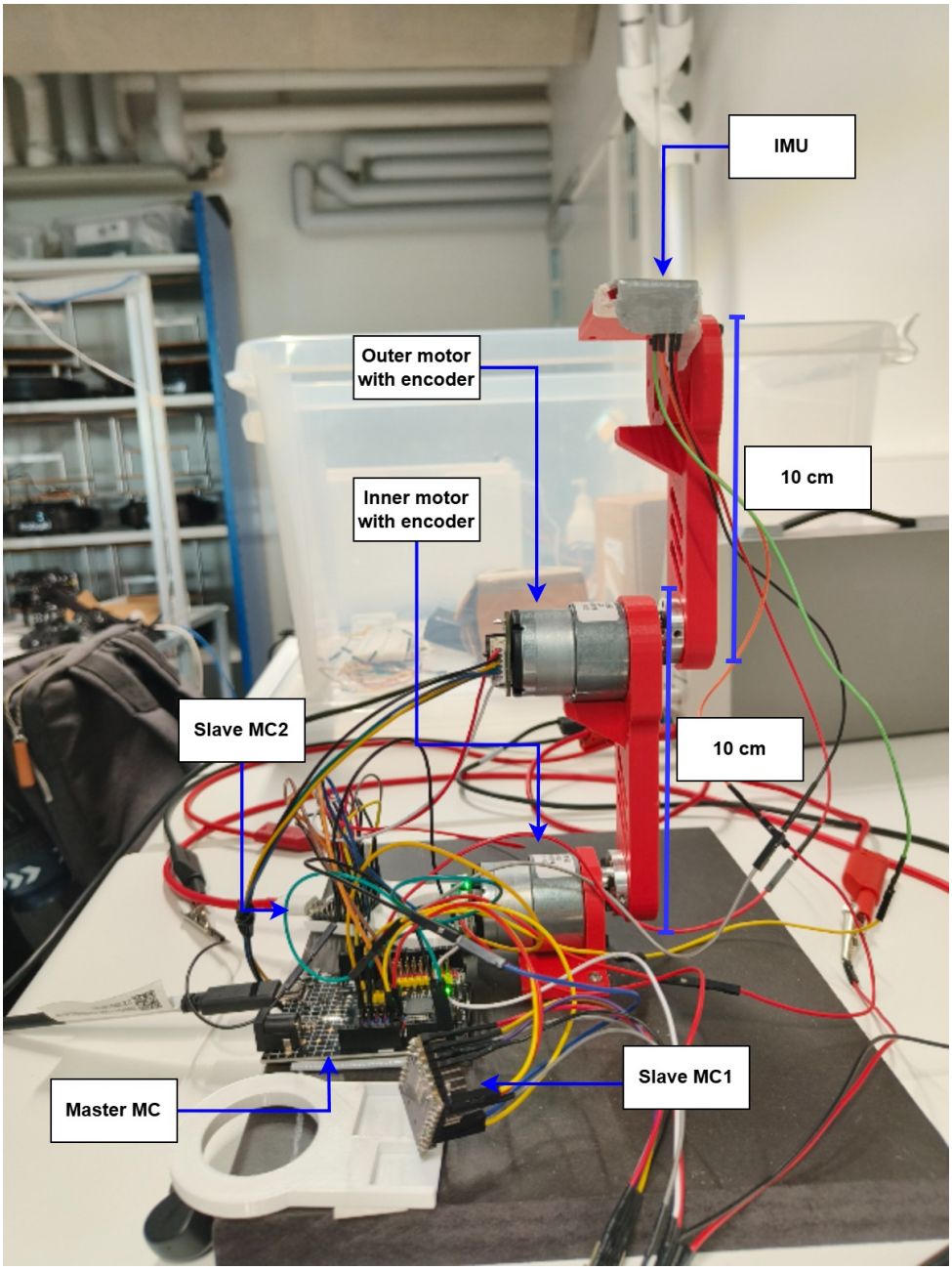


Fig. 17: Profile view of the robot arm

The code for the respective microcontrollers can be found on the associated GitHub Repository [20], in the *"Arduino files"* folder.

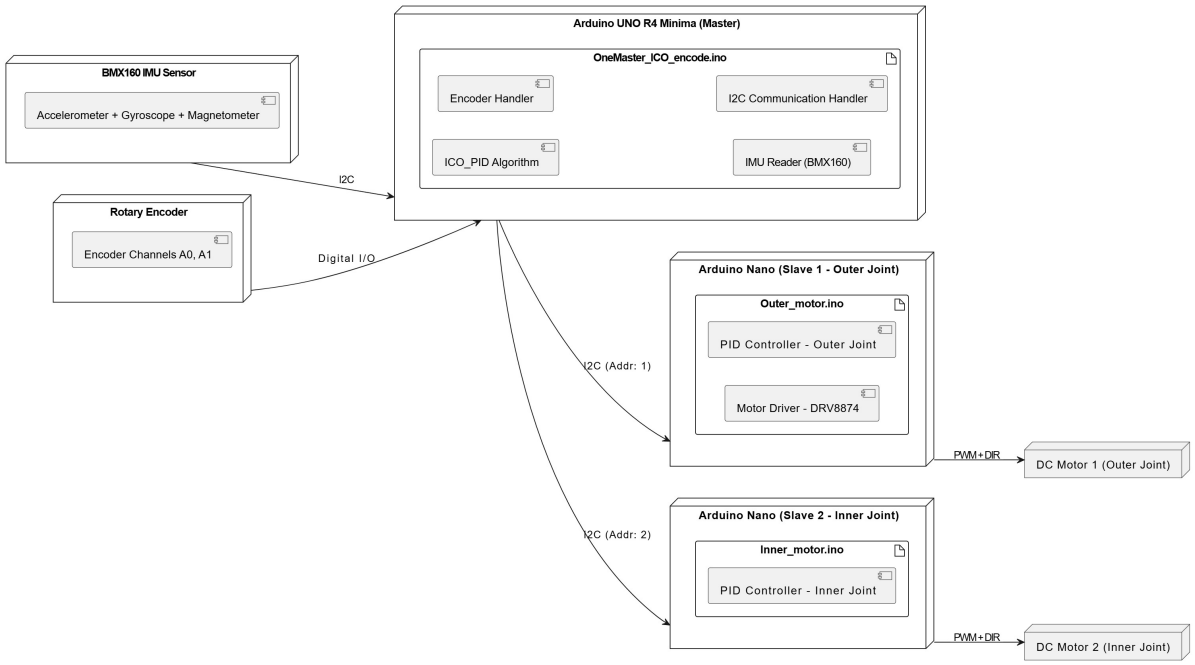


Fig. 18: Deployment diagram for the implementation of the robot arm.

2.7.1 Deployment

The microcontrollers were deployed in a Master-Slave network architecture [23] and utilized communication via the I2C protocol [24], where the Arduino Nano's are acting as slaves, controlling the individual PID-algorithms for the motors, and the Arduino uno R4 minima acting as the master. The master controller is the one subscribing to the sensor inputs and running the ICO learning algorithms. A deployment diagram for this configuration can be seen in Figure 18. Design of the simulation framework, specified in Appendix 3, was done with this configuration in mind, such that the system connections would mimic the behavior of the online system. The network is configured such that each slave-controller was assigned an incremented I2C network address, where the addresses would increment from the outer most motor, until the root of the inverse pendulum arm. I.e, the outer most motor, closest to the IMU, would be assigned Addr. 1. Then, for an arm of N joints, the motor closest to the root would be assingned Addr. N. The accumulative error specified in Eqn. (86) in Appendix 3 would be calculated on the hardware with this structure in mind, such that the total accumulated error is used for the deduction of PID parameters for the motor in Addr. 1.
The software for the master-controller was implemented with scalability in mind, such that a given amount of motors could be deployed and receive estimated PID-gain parameters. The code for the slave-controllers is identical for each joint and receives the specific parameters calculated for the specific allocated I2C address. With this configuration, the system is versatile in regards to the amount of physicals motors on the system. The bottleneck of the system would in this

configuration be identified as, for $N \rightarrow \infty$, the processing power of the micro controller itself, since the matrices used for the ICO-PID algorithm would be incrementally larger for every deployed motor, and the problem of configuring the transmitted struct to be sent on the I2C network to the slaves. Especially the latter object could be identified as an issue, especially for high sample rates dt, where each iteration loop would introduce I/O's. It would introduce N I/O's from the sensors (encoders, IMU), and N I/O's from sending deduced gain parameters to slaves. This would result in queuing and introduce system latency.
A class diagram depicting the parameters and methods for the controllers, along with their dependencies can be seen in Figure 19. The system was implemented using Filtering of derivative dynamic in ICO learning, specified in **Option 2** in Section 2.6, since the simulation results of this configuration seemed most promising (see Appendix 5). Testing the configuration for the system involved tuning it to identify the best learning rate μ, sampling time dt, and best Poisson filter $H_{poisson}$ in terms of filter length, high-pass decay and low-pass decay. Thus, when a suitable format was deduced, it would be expanded to more motors.

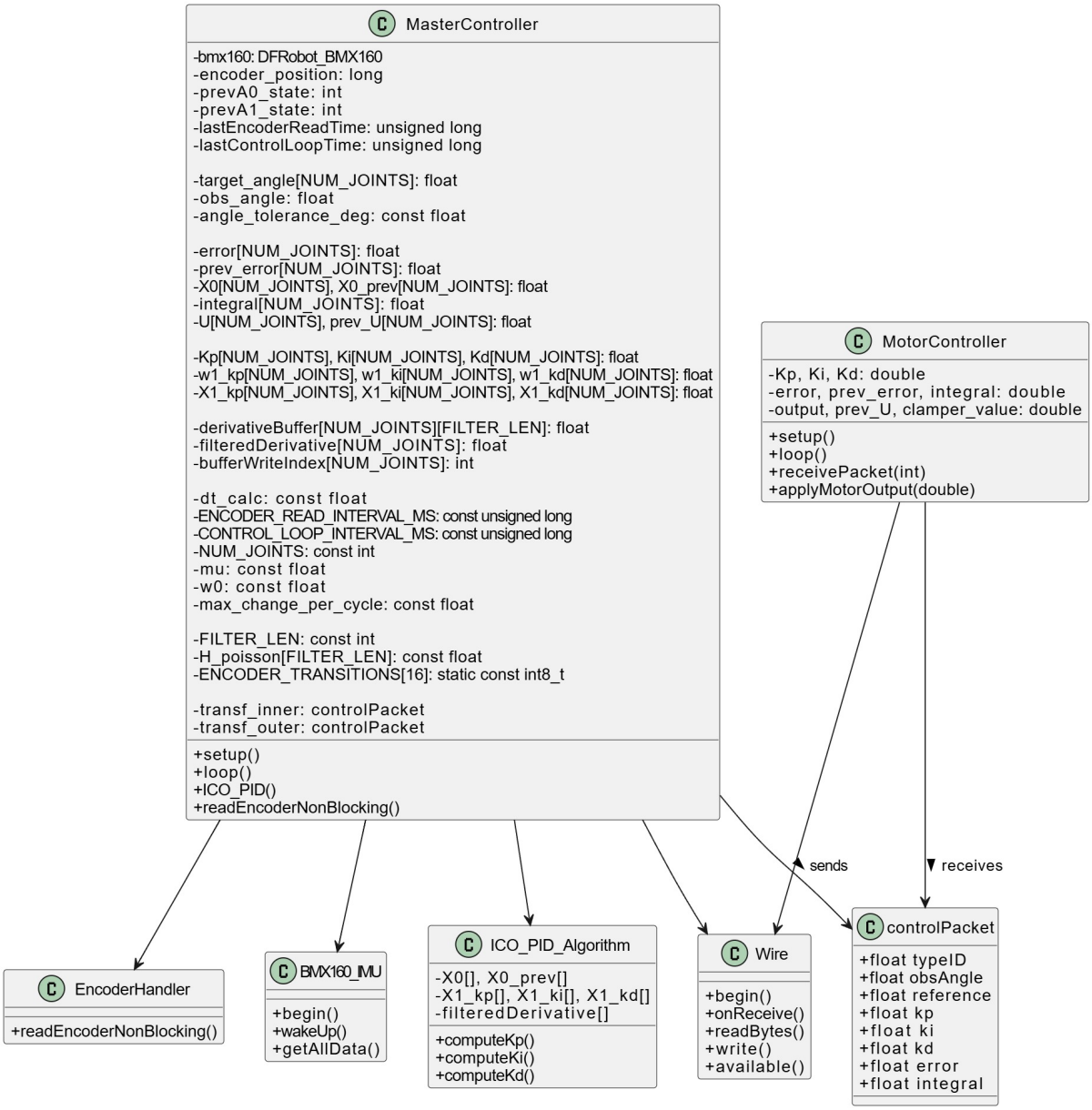


Fig. 19: Class diagram for the software implementation of the robot arm.

3 Results

This section outlines the results of the tests conducted for the deduced ICO-PID algorithms from Section 2 on simulation and embedded hardware environments. The documentation for the results are available in Appendix 5 and Appendix 6 for simulation and hardware results respectively. This section therefore focuses on how the tests were designed and what the initial yield of the tests was. For an in-depth discussion of the result, and the project as a whole, see Section 4.

3.1 Simulation test scenarios and results

The mjority of test scenarios conducted for this project is done in the simulated environment, since this provides a versatile benchmark for easily testing a variety of configurations. The test cases uses two joints and sections with the length of 10 cm to mimic the real-world system used for the project. The simulation framework utilized for conducting the test scenarios is outlined in Appendix 3.
The purpose is to assess whether the system exhibits stable behavior, or contains oscillation with different initial values. The first set of tests will be without disturbance, such that general convergence and gravity adaption can be observed. The factors which are subject to tuning is in general the learning rate μ and the sampling time dt. Another alterable variable is the reference signal (target), which optimally wouldn't change the convergence behavior of the system. Each scenario will run for 50 seconds, I.e. with a sample time of 0,1 seconds that is 500 iterations of the algorithm. When the sample time is altered, the amount of iterations for the algorithm will be either smaller or larger depending on the alteration.
In Appendix 6 there will be animations available of the simulation for each scenario. The scenarios will be labeled accordingly, and discussion of results is available in Section 4. All test scenarios are available in Appendix 5.

3.1.1 Scenarios without filtering

This section of tests contain test scenarios where filtering isn't applied on the signal channels of the model. This ranges over the test scenarios **S1** to **S3**.

3.1.2 Scenarios with filtering

When observing the results of the initial tests, it can be inspected that for a vast majority of the tests, the derivative in the models evolve with large instantaneous oscillations, which can be difficult for online embedded systems to accommodate. This can be observed in scenarios **S1.3**, **S2.3**, **S3.2** and **S3.3**. In most cases, this is due to an extreme initial error, low sampling frequency or high learning rate. To accommodate this, a filtered model should be used to smoothen these dynamics.
For these tests, two different filtering configurations for the algorithm will be tested as mentioned in Section 2.6.3. Both involve filtering with a Poisson band-pass filter. The same parameters as before are used for each scenario, while the filter used will be with the Poisson filter with parameters $N = 51, \lambda_{low} = 20, \lambda_{high} = 1{,}5$. This is a filter that is manually chosen based on iterative

trial and error. Whether this is the best possible filter is up for debate of course as the parameters should be tuned based on system and task specific dynamics. For more on this method, see Section 2.6.4.

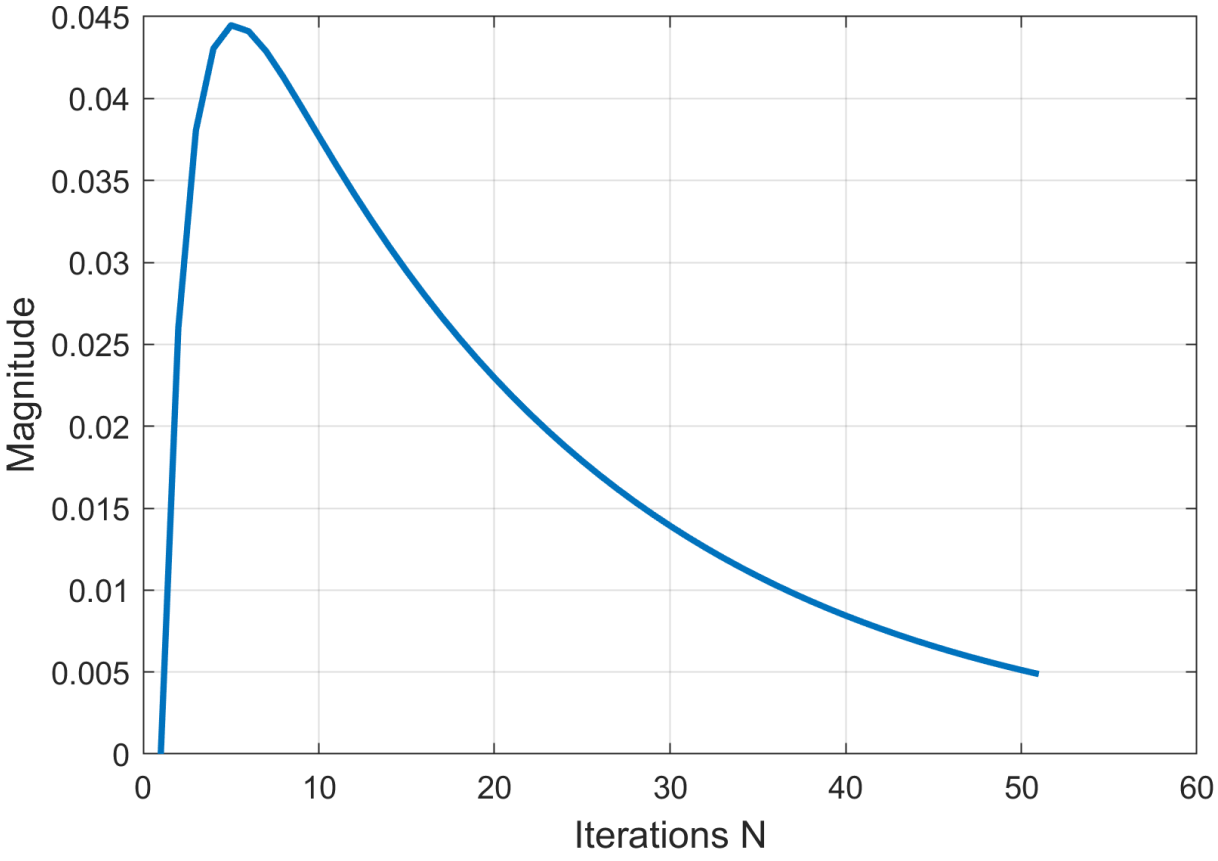


Fig. 20: Poisson filter used for test scenarios with parameters $N = 51, \lambda_{low} = 20, \lambda_{high} = 1,5$.

The filtered test scenarios without disturbances range over the test scenarios **S4** to **S9**.

3.1.3 Scenarios with disturbance

In general, ICO learning is capable of accommodating a given random disturbance D. While this is how it works in practice, it is a rather unfair test criteria. Thus, a momentary disturbance is chosen, and used for every test scenario. As mentioned in Section 7.3.5, the disturbance is designed such that it applies a random angular change in a random direction at a specified interval (iteration-interval). This is changed for testing purposes to be a single disturbance at one specified iteration-number. To observe the transient behavior of both the initial convergence and the convergence from the results of a disturbance, the simulation time is extended to 100 seconds, and the disturbance will be imposed on the system at 50 seconds. A disturbance of 0.3π is chosen as

the test disturbance, and will be imposed on both joints half way through a scenario. The filtered test scenarios including this single impulse disturbance range over the test scenarios **S10** to **S15**.

3.1.4 Periodic disturbances

This section of testing scenarios discovers how the filtered systems act in a more embedded system implementation oriented approach. This will include a more damped learning rate of $\mu = 0.00001$, and periodic disturbances every 10 seconds. The simulations run for 15 minutes, which is 9000 iterations of the algorithm for a sampling rate of $dt = 0.1\ s$. The target position will also be the initial position at 0π. Two versions of disturbance will be used, one that subjects the system to a momentary disturbance as described in Listing 3. The other version discovers a more realistic implementation, where the disturbance is applied gradually over the duration of a few iteration, such that it isn't a momentary change of angle within 0.1 seconds. This is done because a 0.3π change of angle within 0.1 seconds on the robot arm is deemed unrealistic given the physical boundaries of the system. The disturbance ramp used for this is seen in Figure 21. This should give the algorithm the opportunity to show its adaptiveness. The filtered test scenarios including periodic disturbances range over the test scenarios **S16** to **S17**.

3.2 Hardware implementation tests

For the hardware implementation, iterative tuning was done to assert a set of testing parameters for the system. The documentation for the tests can be seen in Appendix 6, and are ordered in three categories (**1-3**) where each scenario would run on both mono (**MS**) and dual (**DS**) motor set up configurations. The reason behind separating the tests into three categories are outlined in Section 4.3. The test for the system were conducted such that the robot arm would be held at an initial position, which was also the target position, and then let go of such that the algorithm could adapt to gravity, and afterwards adapt to disturbances. This showed promising results for a single motor configuration, where it would be capable of returning to the setpoint and adapt to small disturbances. The test setup for this can be seen in Figure 22. A similar set up was used for a deployment with dual motors, which can be seen in Figure 23. The dual motor configuration utilized the same system parameters and the same code for both slave controllers as used for the single motor configuration. For reasons also explained in Section 4.3, testing was done with a range of motion (ROM) of $\pm 30^\circ$ of the setpoint at $R = 0^\circ$ level. In general, the test scenarios would use the same system parameters in all three testing categories. The filter used can be seen in Figure 20, a sampling rate of $dt = 0.03\ s$, and a learning rate of $\mu = 0.0000001$. The specific differences between scenarios are as laid out:

- **(M)(D)S.1** uses a higher learning rate of $\mu = 0.0001$,
- **(M)(D)S.2** uses the shorter filter seen in Figure 24.
- **(M)(D)S.3** uses a sampling rate of $dt = 0.01\ s$.

A few additional tests are documented which doesn't fall into the categories, but were still worth inclusion are also available in the appendix.

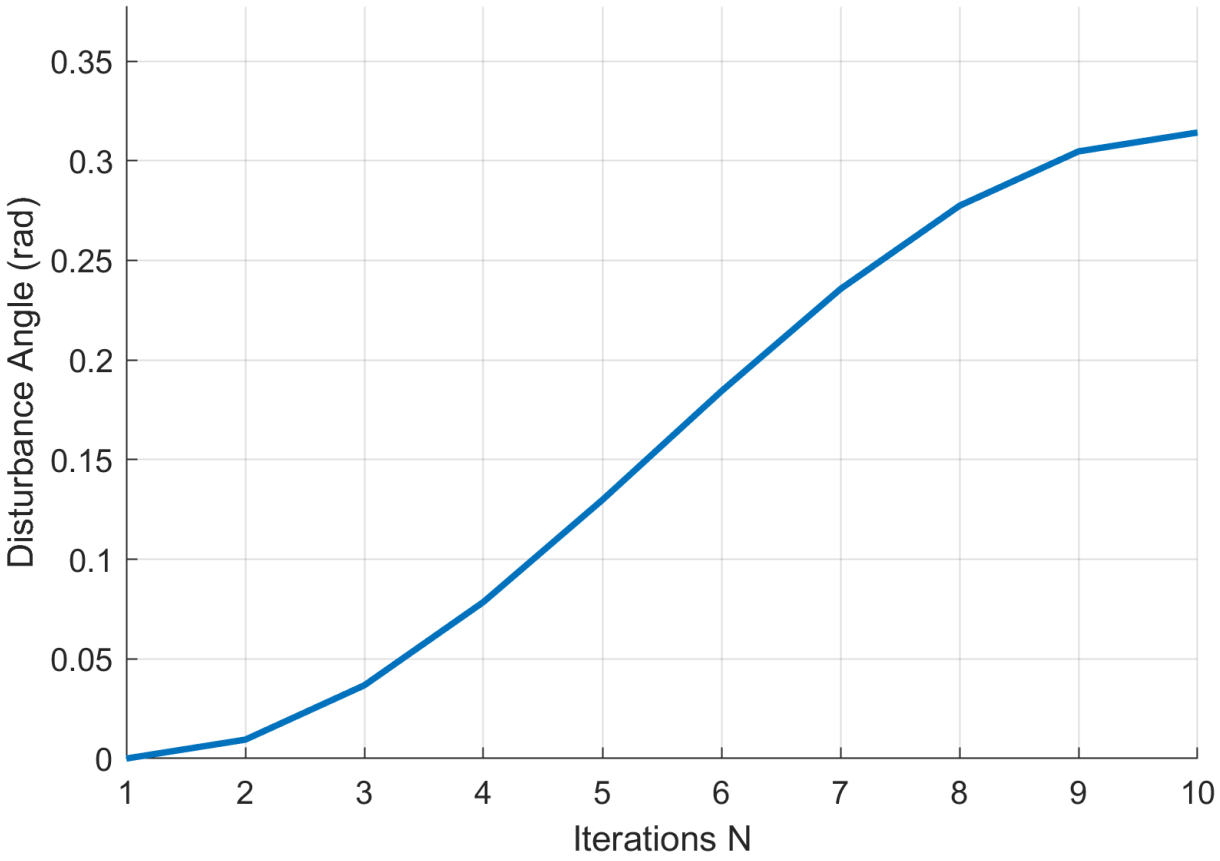


Fig. 21: Implemented disturbance ramp.

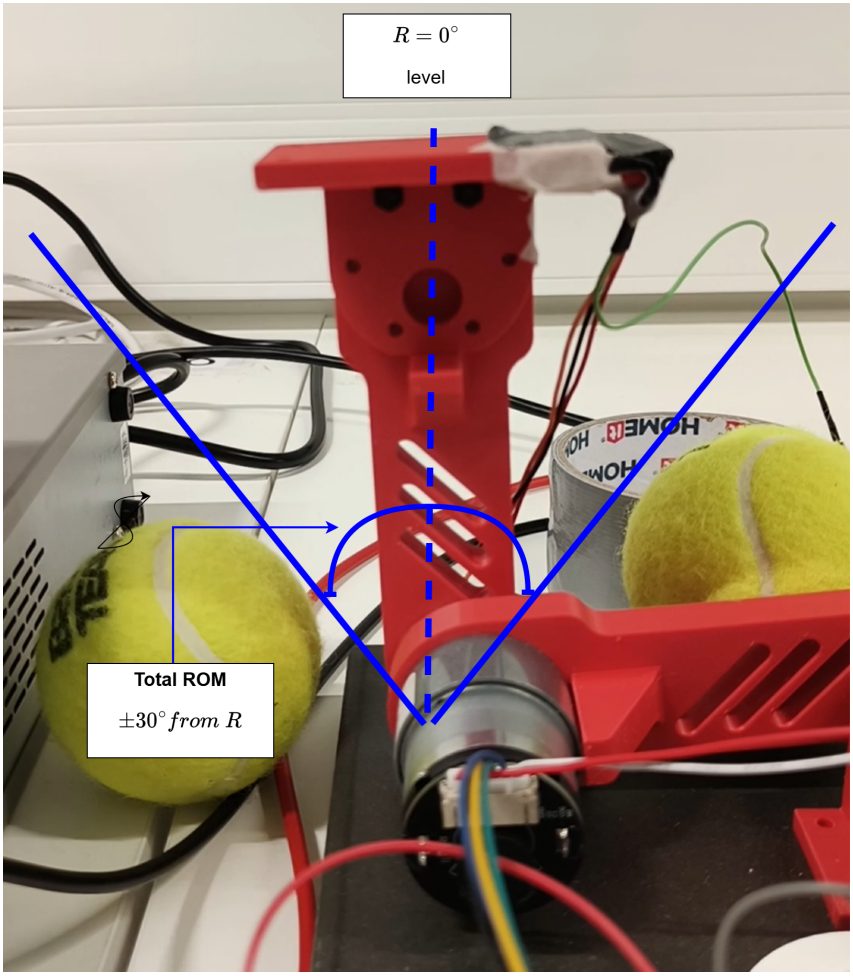


Fig. 22: Single motor test setup for the robot arm

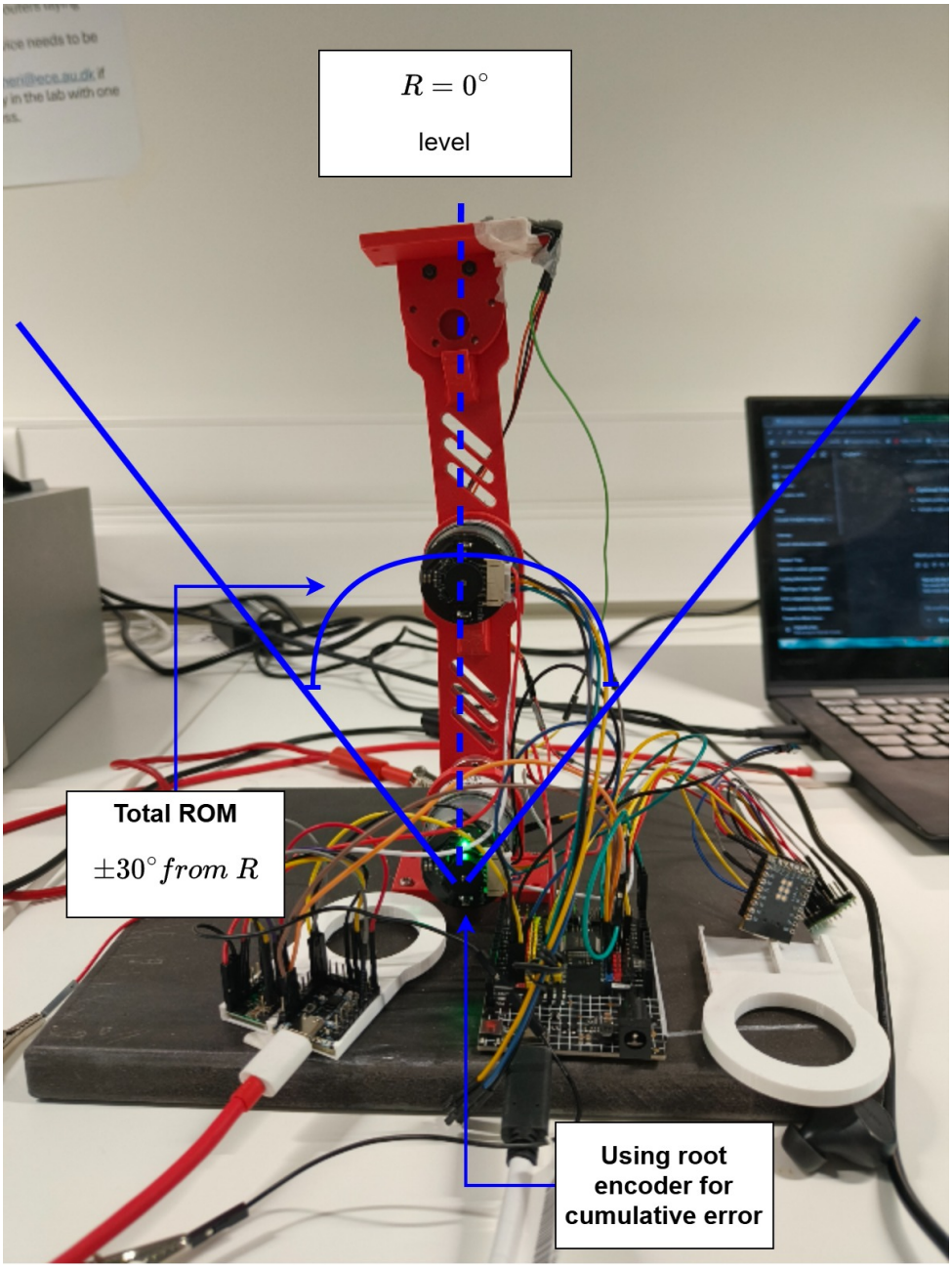


Fig. 23: Double motor test setup for the robot arm

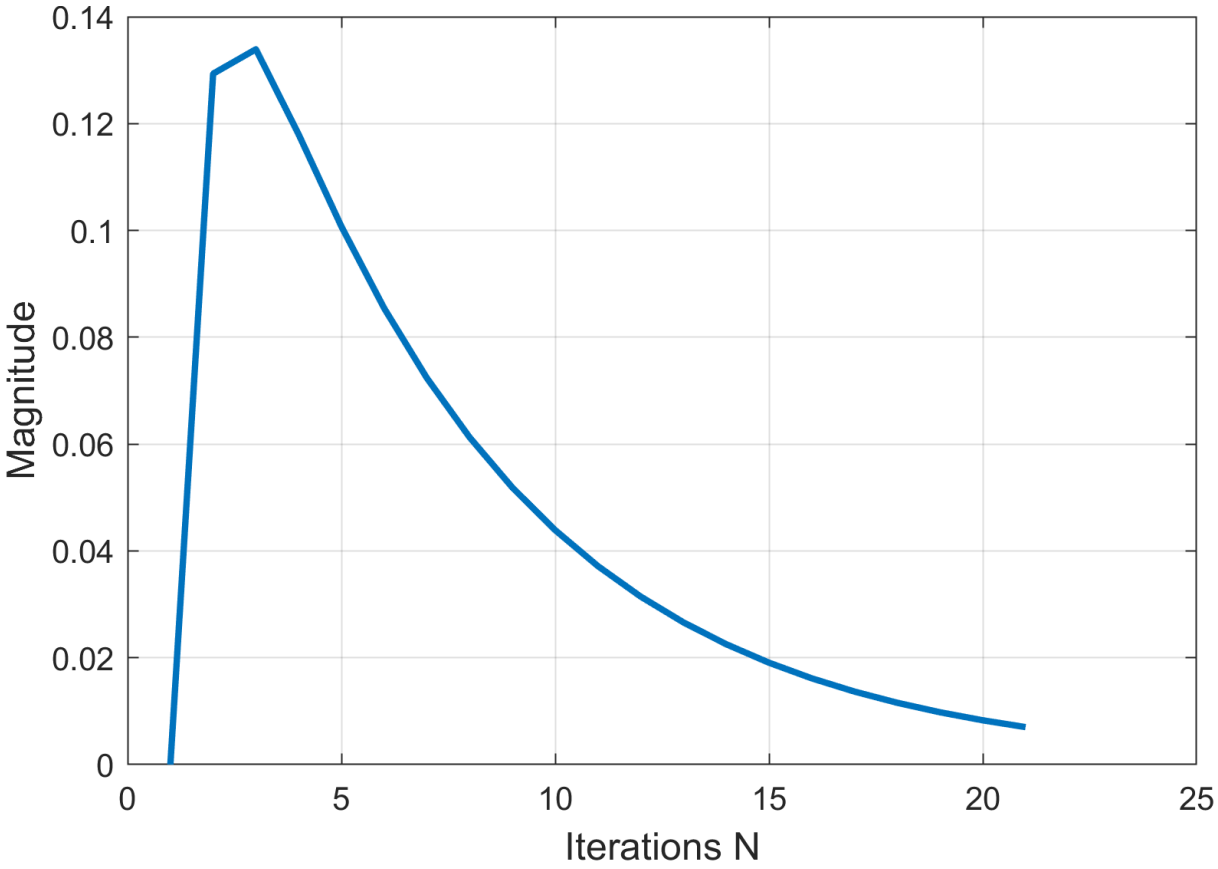


Fig. 24: Poisson filter implemented on the master controller for tests **(M)(D).2** with parameters $N = 21, \lambda_{low} = 6, \lambda_{high} = 0.6$.

4 Discussion

With all test scenarios outlined above in Section 3, it is possible to observe how the different models adapt in the respective scenarios. Here, a discussion is provided on the effect of filtering, and the effect of disturbances upon the system.

4.1 Effect of filtering

The simulation tests can generally be divided into two scenario groups. A filtered group and an unfiltered group. The unfiltered group was conducted initially to assert the general behavior of the system and deduce what kind of filtering was required for the system to control the weight evolution dynamics. Once this was done, the filtered scenarios were conducted to ensure that the prior observed unfiltered evolution dynamics were indeed contained. Therefore, this section contains an outline and a discussion of the individual results, and then a comparison.

4.1.1 Unfiltered scenarios

As all other scenario, the unfiltered ones are done in a gravitational environment. There are however no unfiltered scenarios that contains disturbances. This is because the system, as a requirement, must be able to adapt to the gravitational dynamics, and if this is not done to a satisfactory extent, then testing with disturbances would be redundant. The positive take away from the theoretical simulations is that the ICO weights of the models in all the scenarios, except that of scenario **S3.3**, converges. There are on the contrary a notable dynamic which is represented in nearly all the scenarios which is a hurdle for the system to adapt, which being overcoming the torque effect. The biggest hurdle for the system is the spike in integral windup. Once the integral has gained enough accumulated effect, the K_i gains explode the system, lessening the torque effect, and therefore overshooting the setpoint. A filtering dynamic of the weights in this case could dampen this change and lessen the resulting weight oscillations. Another dynamic to accommodate is discovered in the test scenarios section **S3**, where $R \neq \texttt{init\_pos}$. This imposes a large initial error value on the system, which winds up the error derivative because a large control output is initially computed.

4.1.2 Filtered scenarios

The filtered scenarios contain very damped and smoothed curves both for the PID gain parameters, and also for the ICO weights. In the initial iterations of the scenarios this becomes apparent. A large initial curve change is initiated, and proceeding this the curve smoothens out with a tail-like development such as the Poisson distribution depicts it. This especially can be seen in scenarios **S4.1**, **S5.1**, **S7.2**, **S7.3** and **S9.1**. The ICO weight and PID curves can also be observed to behave much like the Delta pulse response from that of Figure 1(B) by Porr and Wörgötter [5]. this can be seen, for example, by closer inspection of scenario **S17.2**, which can be seen in Figure 25.

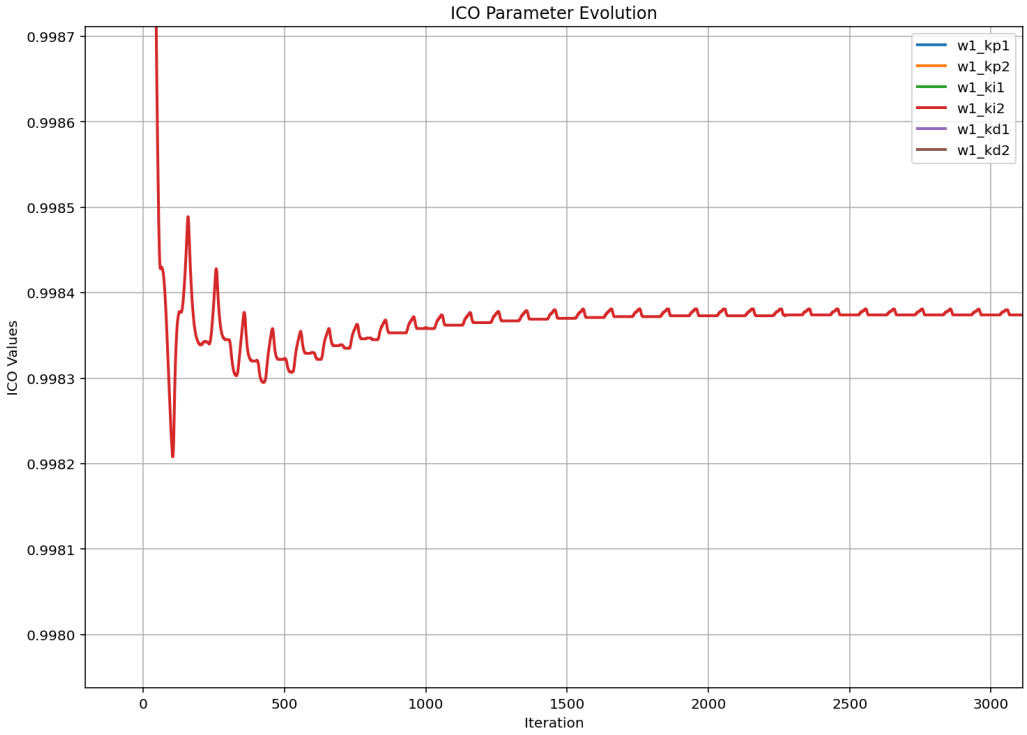


Fig. 25: Closer inspection of $w1_{ki1}$ weight curve for scenario **S17.2**.

As Figure 8(B) [5] also depicts, the weight adaption curve becomes more stable and converges, and has thereby compensated for the disturbances.

4.1.3 Filtering in ICO update mechanic

The 'ICO Derivative Filtering' model, as explained briefly in Section 2.6.3, was intended to dampen the derivative mechanic of a controller, much like a derivative term of a PID controller would. This would become useful for dampening the X_{1ki} signal in a way that wouldn't result in delays, and by extension, windup. This would happen if the channel was filtered directly by a Poisson filter, where the effect would be stretched over proceeding samples. By filtering the derivative of the error X_0 for the ICO weight $w1_{ki}$ update instead, the dampening of the Poisson filter would be passed via a scalar instead of a delay, mimicking the filtering via $\Delta w1_{ki}$. This means that when $e(t) \rightarrow 0$, Eqn. (74) and $w1_{ki}X_{1ki} \rightarrow 0$. Another bonus of this method is that of only a single convolution pr. model, and no filtering convolutions as part of the output sum. This makes the system less computationally heavy, and also easier to implement.

4.1.4 Comparison

When inspecting the results of the simulations and comparing the system responses, it shows that the filtering of the system has a stabilizing and smoothening effect. This effect, as a result of Poisson filtering, becomes apparent, when inspecting test scenarios **S3.1**, **S8.1** and **S9.1**. The unfiltered version in **S3.1** has an instantaneous value change at the initiation of the scenario because of the initial error. The motor has a hard time actuating this value into the PID algorithm because it needs to learn the non-linearity of gravity before it can accommodate the linear initial error.
This means that when the algorithm has finally converged towards the target, the w_{1k_d1} weight of the ICO learning model for the K_d gain has been winded up to $w_{1k_d1} \approx -76$ and $w_{1k_d2} \approx -59$, and is therefore more sensitive to disturbances. Comparably, the filtered scenarios are not winded up, and has a more smooth and observable evolution pattern. The derivative weights converge towards $w_{1k_d1} \approx w_{1k_d2} \approx 0.7$ and $w_{1k_d1} \approx w_{1k_d2} \approx 1$ for scenarios **S8.1** and **S9.1** respectively. In both of these scenarios, and also the other filtered scenarios for that matter, the effect of the Poisson distributed filters are prevalent. Most prevalent is this trend in the scenarios with the ICO-derivative filtered model, which is an interesting dynamic in the sense that the output itself is computationally unaffected by filtering, where it is only the update mechanic that is effected. This insinuates that with only a single filter implemented in the ICO learning model, compared to that of both ICO-channels being subject to filtering in the output summation, a smoothened signal dynamic can be obtained. Comparing **S3.3** and **S9.3**, it can be observed how effective this filtering is. Here, the unfiltered scenario has an extreme initial spike down towards $w_{1k_d1} \approx w_{1k_d2} \approx -3200$, where the filtered scenario is damped such that it converges at $w_{1k_d1} \approx w_{1k_d2} \approx -17$.
In scenario **S3.3**, the extreme instantaneous ICO-weight change has a noticeable effect on the PID parameters themselves. The K_d gains are extremely winded up at initiation, which results in a marginally oscillating system when the $e(t) \to 0$, which can be seen in Figure 26. Another point, besides the dampening of the signal, is the effect of stretching impulse changes to later iterations to acquire the smoothening behavior. When inspecting scenario **S9.3**, it can be observed that the initial error has its effect distributed over the subsequent iterations, aligning with the filter in Figure 20, which is exactly what is desired. Therefore, depending on system constraints and dynamics, the length of the filter N can be tuned to apply the exact smoothening/delay wanted from the system it is implemented in. It is worth noticing however, that there is a notch in learning curve at around 51 iterations in every filtered scenario, which corresponds to the filter length. This is likely because at this point the filter buffer is filled up. This could imply that the model has the most information and therefore begins to act more accurately on it.

4.2 Accommodating disturbances

The purpose of conducting the simulations in general was to deduce the ICO learning model that would best accommodate the non-linear impact of gravity, while also adapting to imposed disturbances in the system. The test scenarios show that this is feasible. Both filtered ICO learning models are capable of effectively adapting to disturbances and ensuring that the setpoint is reached and that convergence occurs such that $X_0 \to 0$. The disturbance adaption in the filtered models resemble each other in terms of PID gains change. The parameters increase greatly to ac-

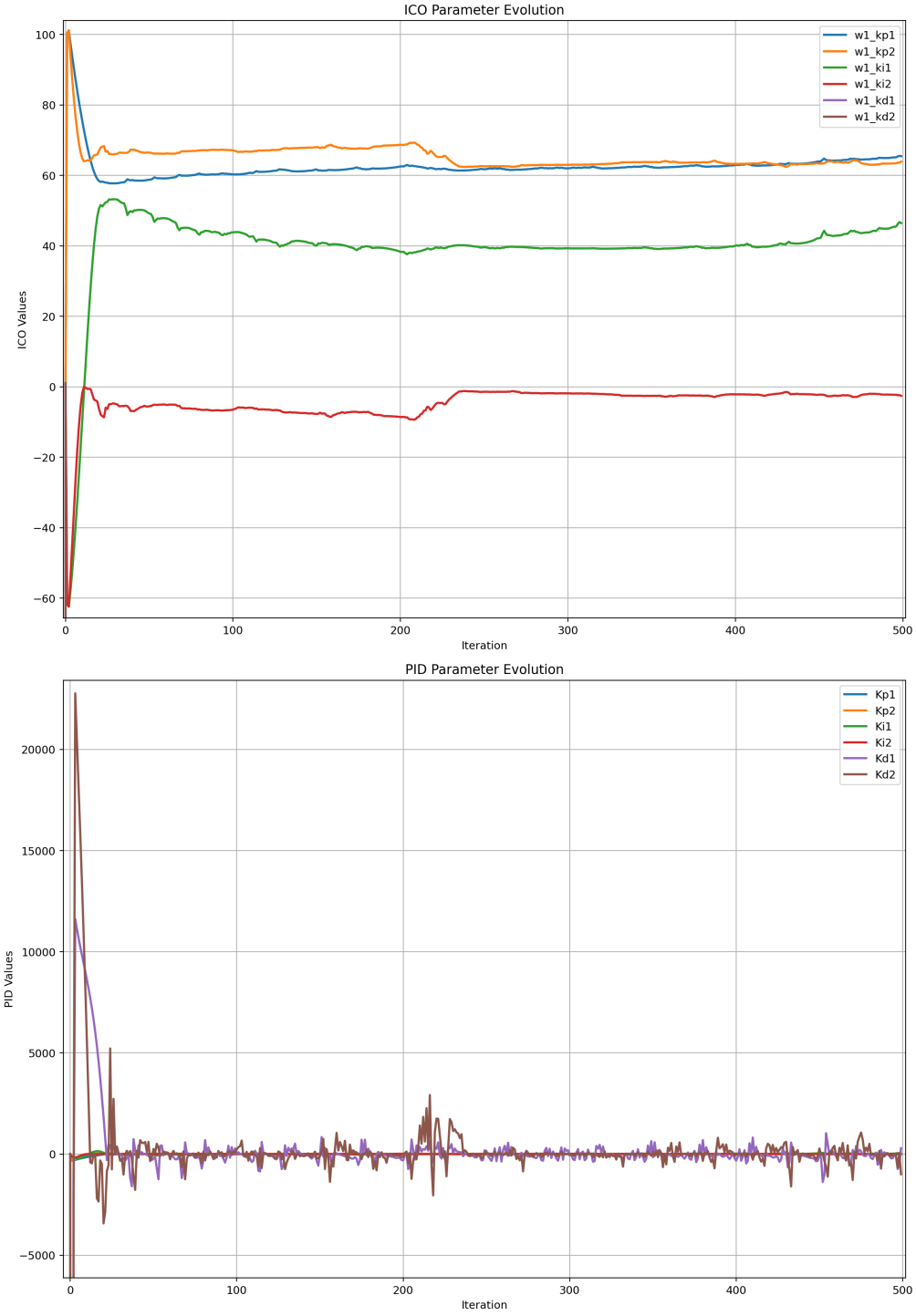


Fig. 26: Closer inspection of Scenario **S3.3**.

commodate the sensed disturbance. The motor angles are then forced back to the position prior to the disturbance and the PID values return to pre disturbance state. The main difference in weight adaption can be observed at smaller sampling intervals. For these cases, Δw_1 achieves proportionally higher changes in Eqns. (72) and (74). This is confirmed in the test scenarios **S12.1** and **S13.1**, where the weights do not necessarily return to the pre-disturbance steady state. Comparably, for the scenarios with the opposite cases they do, such as scenarios **S12.3** and **S13.3**.

4.2.1 Alternate disturbance simulation

When inspecting the response of the systems in terms of disturbances over a longer period separated by intervals, it's discovered that this also seems to converge in a gravitational environment. The first test scenarios with disturbances (**S10** - **S15**) were all under gravitational influence and with only a single instantaneous disturbance. This was done with the intent to assert that the system didn't become unstable at the first sensed disturbance and that it would continue to converge. When comparing these to the experiments conducted by Bernd Porr and Florentin Wörgötter for both their ICO [5] and ISO [17] algorithm articles, they were conducted on systems that weren't affected by as extreme non-linear dynamics, such as gravity in this study, and they were configured as the main control methodology compared to that of adapting PID gains. They were also initiated with very low learning rates of $\mu = 10^{-6}$ [5] and $\mu = 0.00002$ [17] respectively, which generally for embedded systems would be considered a reasonable configuration. This will make sure that the algorithm proceeds slowly and doesn't wind up the system before it has time to react. When configuring the robot arm simulation with a relatively low learning rate, and subjecting it to continued disturbances, the results will be as seen in test scenarios **S16.1** and **S17.1**. The results for the two filtered models are varied, where the best stable result is yielded by the 'ICO Derivative Filtering' configuration. This makes sense in that, for the 'K_d Model Filtering' configuration, it is only the K_d ICO learning model that has its predictive channel X_{1d} filtered, where it is only the error channel X_0 that is filtered in the others. This would explain the oscillations in the response, especially those of the integral weights. To solve this, one might consider applying Poisson filtering to the K_i ICO models as well, however smoothening integral changes into future samples can have unwanted effects, and potentially result in windup of the K_i gains, which is also discussed in Section 4.1.3.
The test for the 'ICO Derivative Filtering' configuration has promising results where the weights converge, and as can be inspected in the visualization for the scenario, the robot arm quickly gets back on track at each disturbance. Even though the ICO weights converge, the PID-values seem to be diverging. Where this originally would introduce a discussion about the integrity of the algorithm, it should rather lead to inspection of how the disturbance is implemented in the simulation, as seen in Listing 3, and then be compared to how a real-life disturbances work. The disturbance in the simulation applies an impulse change in angle over a single sample, where the change of $\Delta\theta = 0.3\pi$ between iterations is realistically considered extreme. Practically, the derivative change of the initial disturbance effect will be detected long before the full effect has been applied on the system, assuming that the sampling rate dt allows this. This inspired the test scenarios **S16.2** and **S17.2**. Changing how the disturbances work, and inspecting the results, especially how the angles evolve, which can be seen on Figure 27 and Figure 28 for scenarios **S16.2**

and **S17.2** respectively, has given the algorithm ability to show its adaptiveness. The results are nearly identical, but an obvious improvement can be seen in the effect of the disturbance for every iteration.

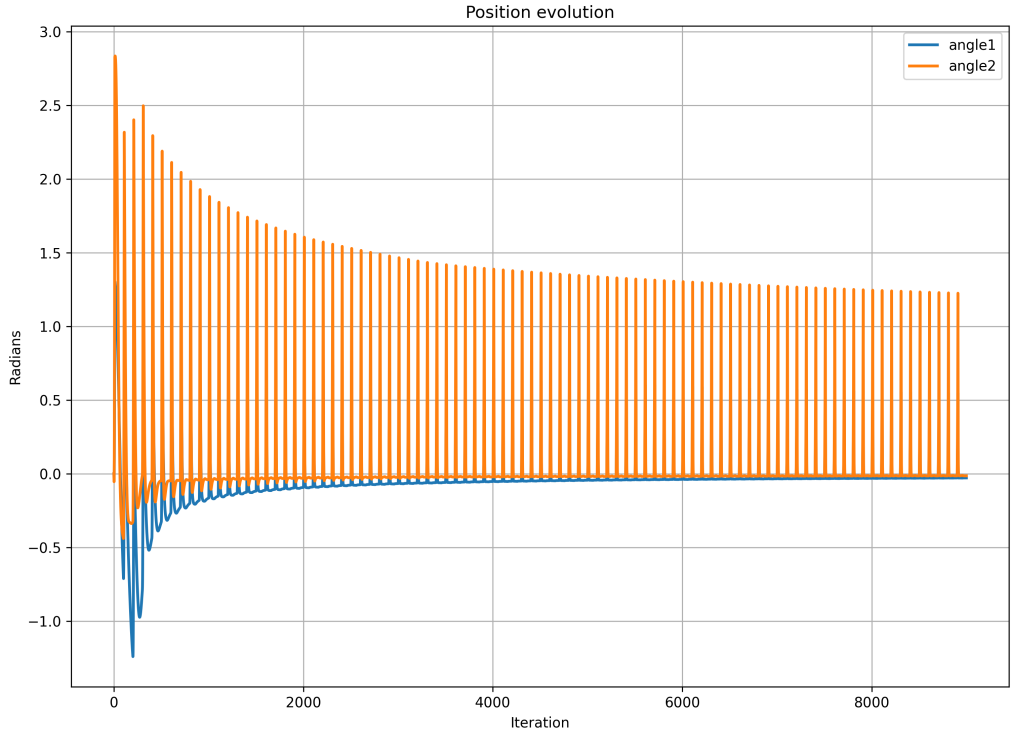


Fig. 27: Angles for test scenario **S16.2**.

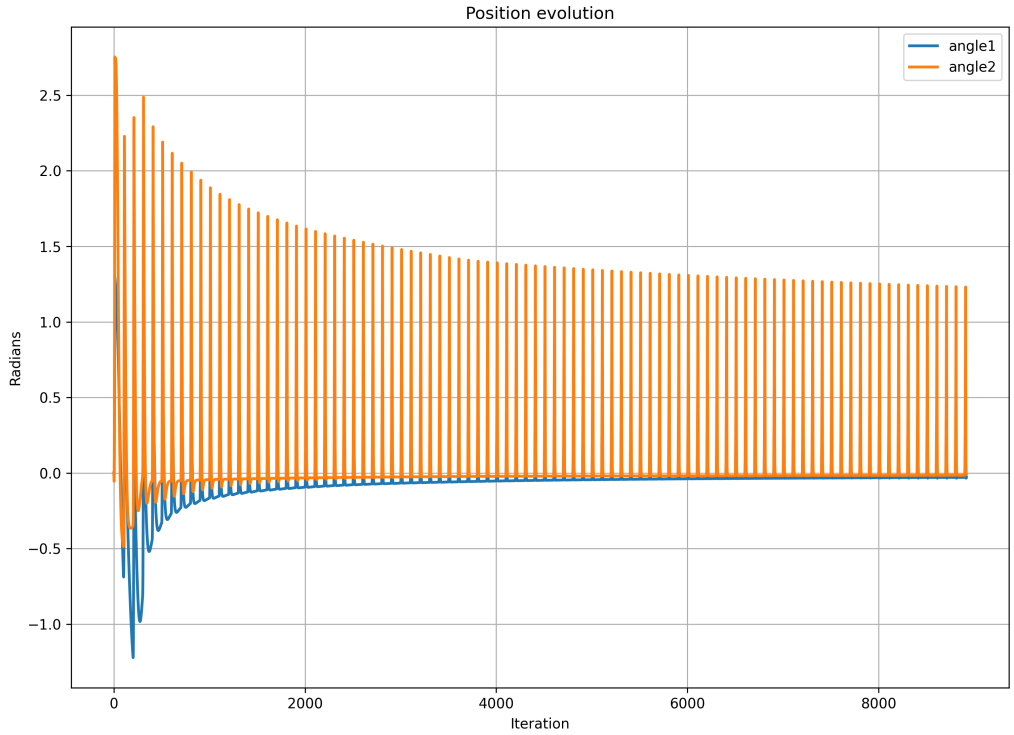


Fig. 28: Angles for test scenario **S17.2**.

4.3 Hardware restrictions

It was prior to the initiation of the project known that the motors would be physically challenged at angles exceeding ±30° from 0° level, which is why testing was done to see if the system could be capable of adapting sufficient ICO weights within that range. Initial testing without filters, showed that the system would become unstable because of the extreme initial errors caused by gravity, which was also a trend seen in the test scenarios for the simulation. When the Poisson filter was added, this mostly solved this issue when testing on the single motor configuration when subject to small disturbances. When expanded to the dual motor format, it became apparent that the ±30° ROM wasn't sufficient to adapt to the greater nonlinearities in the system as a result of the longer robot arm. Thus, the arm would exit the ROM threshold before the algorithm would have ample opportunity to learn the nonlinear dynamics efficiently.
The methods used to try and accommodate the problem are as laid out:

- Using a higher learning rate μ, such that the system would converge on a set of ICO weights faster, thus keeping it within the threshold. However, as explained in Section 2.5.2, a higher

learning rate could introduce instability in the system. This was the case in this method, where the higher tuned learning rate would on a growing differential error force the motor the opposite way and make it pass the setpoint and the opposite threshold boundary. The effect of this measure can be seen for both for the mono and dual motor setups in tests **MS.1** and **DS.1**, respectively

- Reducing the filter length N of the Poisson filter, which would both reduce computations and make the system react faster before the threshold boundary was breached. It would also mean that when the system had passed the setpoint, it would stop actuating in the wrong orientation earlier because the filter has less time delay. However, the resulting, less smoothened, response would then resemble more and more the unfiltered system response, where the less smoothened dynamic became a problem for the system to effectively actuate, and made it fail faster. The effect of this measure can be seen for both for the mono and dual motor setups in tests **MS.2** and **DS.2**, respectively

- Introducing a higher sampling rate dt such that fast gravitational change would be picked up sooner and thereby have time to accommodate to the growing error before the threshold boundary is breached. However, this introduced lag in the system, which resulted in the master and the slave becoming unsynced. The controllers would compute differing errors among themselves, resulting in the system not having a consensus about error, derivative and integral gain values. This resulted in the slaves using PID gain values, which didn't reflect the current system reality observed by the slaves. A mediation strategy for this was forwarding the system view of the master to the slaves, thus force syncing the system. Congestion would then be a problem because the I2C I/Os channels would have to accommodate more bytes per packet sent. The effect of this measure can be seen for both for the mono and dual motor setups in tests **MS.3** and **DS.3**, respectively

The documentation for the tests can be seen in Appendix 6. The hardware restrictions would in general result in the system algorithm not having the ROM to fully experience the nonlinear dynamics and adapt to them.

4.4 Future work

The potential of the ICO-PID learning algorithm has, as shown in the various simulated test scenarios, a potential for tuning a vast variety of systems using PID controllers for achieving better environment adaptiveness and robustness. To leverage this, a restructured hardware system that is capable of exposing the learning algorithm to the full extend of the environments nonlinear dynamics should be used. This would make sure that the robot arm theoretically would be capable of adapting to any magnitude of disturbance and not just small ones on a single motor as described in Section 4.3. With the current embedded system, if given a disturbance, of significant magnitude, such that the arm breaches the ROM threshold, it will never recover as a result of the motor strength, and never get the opportunity to learn and adapt. In terms of software and

the ICO-PID algorithm, it would be a topic for future work to test the algorithm on other verities of nonlinear dynamics and not just gravity as experienced in the self-balancing robot arm environment. Further exploration into the algorithms implementation on diverse environments operating control systems is warranted. Additionally, a general development of a cross scenario framework should be worked on, in which learned PID-gain parameters from previous scenarios are applied on boot-up of the systems. This would ensure that the system won't have to learn and adapt in the environment at initiation of every new session, but will still be able to effectively accommodate new environment anomalies.

5 Conclusion

Studying the means of deploying the ICO learning algorithm as a viable option for intelligent control, via auto tuning PID-gain parameters for a self-balancing platform, has shown to be a strategy worth considering. The ICO algorithms ability to adapt to disturbances proves a benefit as an intelligent mean of adapting the correct PID-gain parameters in nonlinear environments. After creating a framework for guaranteeing stability in a digital architecture, where the disturbance is sensed through the same sensors as the control signal response, two filtering alternatives were developed utilizing a Poisson-like FIR filter. The first option used traditional filtering of the reflex and predictive channels of ICO learning, while the second option used filtering the learning mechanic of ICO instead of the signal channels. Both filtering models were tested in a simulated environment and showed good results in the context of convergence and disturbance adaption. Because of its low computational cost compared to option 1, option 2 (*'Filtering of Derivative Dynamic in ICO learning'*) was deployed on an online embedded hardware system of the inverse pendulum robot arm for testing. Initial promising results were observed with a single motor configuration, but because of hardware restrictions outlined in Section 4.3, total feasibility within the system boundaries for a dual motor setup could not be inferred. The generally positive results from the simulation environment opens up for investigating alternate hardware environments where PID control is prevalent as framework for further testing and implementation. This includes both filtering models deduced in this study, and alternative filtering configurations.

6 References

[1] A. D. Anthony, P. L. Almvig, and S. A. Andreasen, "Autotuning pid using relay feedback method: Acs case study," technical report, Aarhus University, ECE, ETACS, Group 1, Denmark, November 2024. Unpublished student report.

[2] K. J. Åström and B. Wittenmark, "Adaptive control second edition," 1995.

[3] R. P. Borase, D. Maghade, S. Sondkar, and S. Pawar, "A review of pid control, tuning methods and applications," *International Journal of Dynamics and Control*, vol. 9, pp. 818–827, 2021.

[4] D. O. Hebb, *The organization of behavior: A neuropsychological theory*. Psychology press, 2005.

[5] B. Porr and F. Wörgötter, "Strongly improved stability and faster convergence of temporal sequence learning by using input correlations only," *Neural computation*, vol. 18, no. 6, pp. 1380–1412, 2006.

[6] K. J. Åström, "Ziegler-nichols auto-tuners," 1982.

[7] T. Hägglund and K. J. Åström, "Industrial adaptive controllers based on frequency response techniques," *Automatica*, vol. 27, no. 4, pp. 599–609, 1991.

[8] MathWorks, "Gain-scheduled pid autotuning torque control for a nonlinear pmsm," 2024. [Online]. Available: `https://se.mathworks.com/help/slcontrol/ug/pmsm-tuning-using-gain-scheduled-pid-autotuner.html`.

[9] C. Hang, K. Astrom, and Q. Wang, "Relay feedback auto-tuning of process controllers—a tutorial review," *Journal of process control*, vol. 12, no. 1, pp. 143–162, 2002.

[10] J. G. Proakis and D. G. Manolakis, *Digital Signal Processing: Principles, Algorithms, and Applications*. Upper Saddle River, New Jersey: Prentice-Hall, Inc., 3rd ed., 1996.

[11] K. J. Åström and J. Jacob, "A control application of the direct method of lyapunov," 1963.

[12] M. P. Schoen, *Introduction to intelligent systems, control, and machine learning using MATLAB*. Cambridge University Press, 2023.

[13] K. G. Vamvoudakis, Y. Wan, F. L. Lewis, and D. Cansever, *Handbook of reinforcement learning and control*. Springer, 2021.

[14] H. Zhang, W. Assawinchaichote, and Y. Shi, "New pid parameter autotuning for nonlinear systems based on a modified monkey–multiagent drl algorithm," *IEEE Access*, vol. 9, pp. 78799–78811, 2021.

[15] R. Zhao and W. Tang, "Monkey algorithm for global numerical optimization," *Journal of Uncertain Systems*, vol. 2, no. 3, pp. 165–176, 2008.

[16] O. Dogru, K. Velswamy, F. Ibrahim, Y. Wu, A. S. Sundaramoorthy, B. Huang, S. Xu, M. Nixon, and N. Bell, "Reinforcement learning approach to autonomous pid tuning," *Computers & Chemical Engineering*, vol. 161, p. 107760, 2022.

[17] B. Porr and F. Wörgötter, "Isotropic sequence order learning," *Neural Computation*, vol. 15, no. 4, pp. 831–864, 2003.

[18] MathWorks, "General outline for the cascade control." *MathWorks outline of controlling processes using cascaded PI controllers and single PI control of multi-process plant black-box abstraction*, n.d. https://se.mathworks.com/help/control/ug/designing-cascade-control-system-with-pi-controllers.html#cascadepiddemo-1.

[19] F. A. Haugen, "Discretizing a pid controller." *Online PDF*, 2008. [Online]. Available: https://techteach.no/fag/sesm3401/h08/pid_diskret/tidsdiskret_pid_reg.pdf.

[20] A. Anthony, "Bsc project: Automatic tuning of multiple pid motor controllers for a self-balancing platform via ml." Github Repository Link, 2025. Accessed: 2025-05-21.

[21] D. G. Manolakis and V. K. Ingle, *Applied digital signal processing: theory and practice*. Cambridge university press, 2011.

[22] B. Porr, "Ico learning algorithm repository." *GitHub repository for the ICO learning algorithm*, n.d. https://github.com/berndporr/ICO-learning.

[23] Robottronic, "Arduino i2c and multiple slaves." https://www.instructables.com/Arduino-I2C-and-Multiple-Slaves/, n.d. Accessed: 2025-05-22.

[24] D. Kalinsky and R. Kalinsky, "Introduction to i2c," *Embedded Systems Programming*, vol. 14, no. 8, pp. 1101–1105, 2001.

[25] F. Wörgötter and B. Porr, "Temporal sequence learning, prediction, and control: a review of different models and their relation to biological mechanisms," *Neural computation*, vol. 17, no. 2, pp. 245–319, 2005.

[26] K. L. Luther, R. Yang, and H. S. Seung, "Unsupervised learning by a" softened" correlation game: duality and convergence," in *2019 53rd Asilomar Conference on Signals, Systems, and Computers*, pp. 876–883, IEEE, 2019.

[27] J. Zhang, A. Koppel, A. S. Bedi, C. Szepesvari, and M. Wang, "Variational policy gradient method for reinforcement learning with general utilities," *Advances in Neural Information Processing Systems*, vol. 33, pp. 4572–4583, 2020.

[28] S. Ahmed, A. U. Shah, A. R. Liaqat, and M. O. Khan, "Forward kinematics of a 6-dof robotic arm," tech. rep., Centre for Robotics Research, School of Natural and Mathematical Sciences, King's College London, 2021. Accessed from institutional materials.

[29] Wikipedia contributors, "Angular acceleration." *Wikipedia article on the Angular acceleration*, n.d. https://en.wikipedia.org/wiki/Angular_acceleration.

[30] M. O. Okelola, D. O. Aborisade, and P. A. Adewuyi, “Performance and configuration analysis of tracking time anti-windup pid controllers,” *Jurnal Ilmiah Teknik Elektro Komputer Dan Informatika*, vol. 6, no. 2, pp. 20–29, 2020.

7 Appendix

7.1 Appendix 1: ICO learning interpreted as learning via gradient ascent

To provide an alternate view of the ICO learning process, the learning mechanism in Eqn. (4) can be interpreted as a form of gradient ascent on a correlation-based utility function, where learning reinforces input features that consistently precede increases in reflex signal. Specifically, the learning rule seeks to maximize the temporal correlation between a predictive signal $u_j(t)$ and the reflex signal $\frac{du_0(t)}{dt}$. In practice of course, the ICO learning mechanism does not explicitly define or minimize a cost function as an RL approch does (which was mentioned in Section 1.2.2). However, this is a methodology used in other Hebbian-based learning algorithms [25] [26], often applied in a RL framework [13] [25] [27], and is therefore an interesting inside into the ICO learning dynamics. If defining the correlation-based utility function on the form

$$C(w_j) = \mathbf{E}\left[u_j(t) \cdot \frac{du_0(t)}{dt}\right] \tag{80}$$

where $\mathbf{E}$ denotes the expectation operator [13], which effectively in this context means: The average value/mean of the product $u_j(t) \cdot \frac{du_0(t)}{dt}$ over time and is defined as

$$\mathbf{E}[X] = \int X(\zeta) p(\zeta) d\zeta \tag{81}$$

where ζ is a random variable representing the system's state or the disturbances (t in this case), and $p(\zeta)$ is the probability of getting a disturbance input, I.e., the probability density function of ζ. Thus the utility function for weight change in Eqn. (80) over a time period T can be described as

$$\mathbf{E}\left[u_j(t) \cdot \frac{du_0(t)}{dt}\right] = \frac{1}{T}\int_{t_0}^{t_0+T} u_j(t) \cdot \frac{du_0(t)}{dt} dt \tag{82}$$

Thus, for a single stochastic time step, the learning mechanism in Eqn. (4) can be approximated as

$$\Delta w_j \approx \mu \cdot \frac{\partial C(w_j)}{\partial w_j} \tag{83}$$

This is valid because of the implicit dependence of the reflex derivative $\frac{du_0(t)}{dt}$ on the previous weight $w_j^{(t-1)}$. Consequently, the update rule in Eqn. (5) can be reformulated as

$$w_j^{(t)} = w_j^{(t-1)} + \mu \cdot u_j(t) \cdot \frac{d}{dt}\left[u_0\left(t, w_j^{(t-1)}\right)\right] \tag{84}$$

to show this dependency. This dependency arises because of the output summation of the previous state given as

$$u(t) = \sum_j w_j^{(t-1)} u_j(t) \tag{85}$$

Since the system output $y(t)$ is a result of a convolution response with $u(t)$, and the reflex signal u_0 being an error signal based on $y(t)$, the weight w_j influence the reflex signal indirectly through the system's closed-loop dynamics.

7.2 Appendix 2: Constants and parameters

This documentation for digital implementation contains a variety of different algorithms, and by extension a variety of constant and variable names. For ease, these are outlined in this section in descriptive tables.

Variable / Symbol	Description
`N`	Number of joints/segments in the robot arm.
`L`, `arm_lengths`	Vector containing the length of each arm segment.
`t`, `target`	Target angle or cumulative angle reference for the end-effector.
`init_pos`, `joint_angles`	Initial joint configuration (angles in radians).
`dt`	Time step used for simulation and integration.
`link_masses`	Vector of masses for each link (assumed to be 1 by default).
`gravity`	Gravitational constant $g = 9.81\,\mathrm{m/s^2}$.
`D`	External disturbance vector applied to joints.
`I`	Moment of inertia for each link (uniform rod assumption).
`gravity_torque`	Torque induced on joints due to gravity.
`gravity_acc`, `gravity_displacement`	Angular acceleration/displacement induced by gravity.

Tab. 1: Control loop parameters and robot dynamics.

Symbol / Variable	Description
θ_i	Angle at the i^{th} joint, measured in radians (relative rotation from the previous link).
L_i	Length of the i^{th} link or arm segment.
N	Number of joints (or equivalently, the number of links).
$\mathbf{p}_i$	Position vector of the end of the i^{th} link in 2D space, $\mathbf{p}_i = \begin{bmatrix} x_i \\ y_i \end{bmatrix}$.
Θ_i	Cumulative angle up to the i^{th} joint: $\Theta_i = \sum_{k=1}^{i} \theta_k$.
$\Delta\mathbf{p}_i$	Displacement vector due to link i: $\Delta\mathbf{p}_i = L_i \begin{bmatrix} \cos(\Theta_i) \\ \sin(\Theta_i) \end{bmatrix}$.
`angles`	MATLAB vector containing all joint angles $\theta_1, \theta_2, \ldots, \theta_N$.
`L`	MATLAB vector containing all link lengths $L_1, L_2, \ldots, L_N$.
`coords`	Output 2xN array containing the Cartesian positions of each joint and the end-effector.

Tab. 2: Parameters used in the forward kinematics model.

Variable / Symbol	Description
w0	Constant weight for reference error.
w1_kp, w1_ki, w1_kd	Adaptive weights for proportional, integral, and derivative components in ICO.
X1_kp, X1_ki, X1_kd	Respective predictive signals for proportional, integral, and derivative components in ICO learning
Kp, Ki, Kd	PID control gains updated adaptively.
error	Difference between current and target cumulative angles.
integral, raw_derivative, ICO_derivative	Accumulated and rate-of-change of error.
clamper	Anti-windup mechanism: boolean array to disable integral update during saturation.
U	Final control output.
mu	Learning rate for adaptive weight updates.
is_saturated, is_winding_up, is_error_growing	Conditions used in anti-windup logic.

Tab. 3: ICO and PID adaptive control parameters.

Variable / Symbol	Description
plotWheelLegRobot()	Function that handles animated plotting of the robot arm.
h_lines, h_joints, h_end	Plot handles for updating segments, joints, and end-effector marker.
num_steps	Number of interpolation steps for smooth animation between joint states.
drawnow, pause()	MATLAB functions to update plots in real-time.
xlim, ylim	Axis scaling based on max arm length.

Tab. 4: Plotting and visualization elements.

Variable / Symbol	Description
`fil_len`	Length of the filter (number of coefficients). Should be a positive integer. Defines the support of the FIR filter.
`low_lambda`, λ_{low}	Decay parameter for the high-pass component. Lower values result in a sharper high-pass cutoff.
`high_lambda`, λ_{high}	Decay parameter for the low-pass component. Higher values result in a smoother low-pass cutoff.
n	Index vector from 0 to `fil_len`-1, used to generate filter coefficients.
`lp`	Low-pass exponential decay: $e^{-n/\texttt{high\_lambda}}$. Controls the smooth component of the filter.
`hp`	High-pass exponential decay: $e^{-n/\texttt{low\_lambda}}$. Controls the sharp component of the filter.
h	Final filter response: $h = \frac{hp - lp}{\sum_{k=0}^{\texttt{fil\_len}-1} \lvert hp_k - lp_k \rvert}$, ensuring unit normalization.

Tab. 5: Parameters for the `poisson_bandpass_1d` Function.

7.3 Appendix 3: Simulation framework

Before implementing any algorithm configuration on a real robot, it should be simulated to assert stability and ease in the process of debugging. To simulate the system, MATLAB is utilized. The developed simulation framework must be able to accommodate and simulate different system dynamics, these are outlined as follows.

1. Recreate the closed-loop structure, where the setpoint R (or 'target' if preferred) is the angle used for reference.
2. Be a dynamic model which can simulate a robot arm with more than one joint/controller, where PWM-values are the actuation parameters.
3. A forward kinematics model is used for plotting and animation to ensure correct angle/joint placement.
4. Simulate disturbances inflicted on individual joints.
5. Simulate gravitational dynamical effects on the arm.
6. Include logging of important system parameters, such as PID gain values and ICO learning weights.

All versions of the code can be found on the associated GitHub repository [20].

7.3.1 Closed-loop structure

To create a closed-loop structure for the simulation, the function `simulateRobotArm(N,L,t,init_pos)` is created to contain the main system-loop and parameter setup. Here, the input parameters are `N, L, t, init_pos`, which are the number of arm joints, a vector containing the length of each individual arm joint, the target/reference angle, and a vector for the initial positions for the joints, respectively. Before the main loop is initiated, the parameters for the ICO and PID algorithms are initiated with preferred initial values. At this stage, a sampling rate `dt` is also set.

7.3.2 Error calculations and controller structure

Since the simulation should be dynamic, and by extension have an `N` amount of joints and associated controllers. An association between the individual joints should be established to assert if a target has been reached or not. To do this, the error for each motor is calculated as the cumulative sum of all previous joint angles minus target angle, which is given as

$$E_i = \left(\sum_{k=0}^{i} \theta_k \right) - t \tag{86}$$

Where $i \in \mathbf{N} \mid 1 \leq i \leq N$, θ_k is the angle of the k'th joint, and t is the target. The error, as well as all other controller parameters are set as vectors of length `N`, and all algorithm calculations are done element-wise. They are therefore also initialized as vectors.

7.3.3 Dynamic actuation model

When calculating the angular change between iterations, it should resemble the actuation structure of the structure used on the real robot arm, where a high byte is sent to represent the amount of torque to actuate, which is on a scale of 0 to 255, and a low byte with the direction of the applied torque. This is done with PWM signal actuation, which is applied to the motor-controller. This is calculated and applied in MATLAB as seen in code listing 1.

Listing 1: PWM signal implementation

```
1     pwm_signal = max(0, min(1, abs(U) / 255));
2     motor_speed = pwm_signal * 1*pi;
3     direction = sign(U);
```

Where U is the calculated control signal. With this implementation, a saturation is also implemented for the control signal. the `motor_speed` parameter is the PWM signal multiplied with the total speed/ power that the motor can utilize. This can be change to whatever fits the simulation best.
Thus based on the control signal, the new angles are calculated as

$$\theta_{new} = \theta_{old} + \texttt{direction} \cdot \texttt{motorspeed} \cdot \texttt{dt} \tag{87}$$

7.3.4 Forward kinematics visualization

For visualization, a forward kinematics approach is used, where the coordinates are calculated for a specific set of N angles for an arm with N joints. This is done in the `forwardKinematics` function which returns an array of coordinates for joint positions. This implementation can be seen in listing 2, and the formal mathematical notation in Eqn. (88) [28].

$$\mathbf{p}_i = \mathbf{p}_{i-1} + L_i \begin{bmatrix} \cos(\Theta_i) \\ \sin(\Theta_i) \end{bmatrix}, \quad \text{for } i = 1, 2, \ldots, N \tag{88}$$

Listing 2: Forward kinematics implementation

```
1 function coords = forwardKinematics(angles, L, N)
2     coords = zeros(2, N + 1);
3     for i = 1:N
4         coords(:, i + 1) = coords(:, i) + [L(i) * cos(sum(angles(1:
              i))); L(i) * sin(sum(angles(1:i)))];
```

```
    end
end
```

This function is called by the `plotRobot` function, which can be inspected on the associated GitHub repository [20]. This gives a visual real-time interpretation of the ICO learning & PID algorithms. A visualization for this can be seen on Figure 29.

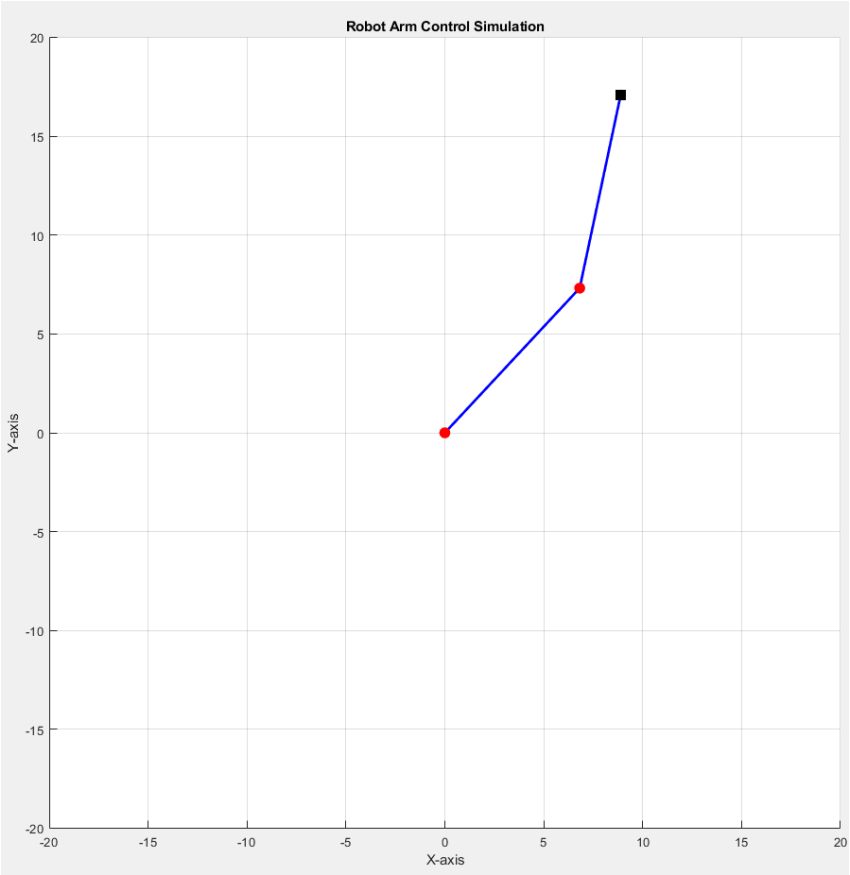


Fig. 29: Plot of robot arm simulation

7.3.5 Simulating disturbance

To simulate a disturbance, a random sudden change of angle is applied in the new angle calculation. which can be changed to a specific kind of disturbance if wished. For implementing it in this iteration of the simulation, it will be a random force with random orientation each joint, to make it as less predictable as possible. This is done at set iteration intervals, the implementation of which can be seen in listing 3.

Listing 3: Random disturbance implementation

```
    % Random Diturbance on every joint
```

```
    if last_iter + 100 < iter
        D = (2 * rand(1, N) - 1);
        last_iter = iter;
    else
        D = (zeros(1, N));
    end
```

The ramp disturbance used is implemented as seen in Listing 4, and the resulting disturbance ramp applied in Figure 21.

Listing 4: Ramp disturbance implementation for test scenarios

```
    disturbance_total_angle = 0.1 * pi;
    disturbance_duration = 1.0;                    % in seconds
    disturbance_interval = 10.0;                   % in seconds
    disturbance_steps = round(disturbance_duration / dt);  % number
        of iterations over which disturbance is applied
    disturbance_period = round(disturbance_interval / dt); %
       interval between disturbances

    % Smooth ramp using half-cosine
    disturbance_ramp = disturbance_total_angle * 0.5 * (1 - cos(
       linspace(0, pi, disturbance_steps)));
                        .
                        .
                        .
    % Random Diturbance on every joint
    D = zeros(1, N);

    % Check if we are within a disturbance window
    mod_iter = mod(iter, disturbance_period);
    if mod_iter > 0 && mod_iter <= disturbance_steps
        D = disturbance_ramp(mod_iter) * ones(1, N);
    elseif mod_iter == 0
        % Optional: for exact multiples, force zero or reset
        D = zeros(1, N);
    end
```

This disturbance is then applied in Eqn. (87).

7.3.6 Simulating gravity

To simulate gravity, the angular change pr. time step caused by gravitational torque is deduced for each iteration. To do this, the inertia impact on each joint is calculated and based on the torque effect of the remaining distal joints from the respective joint [29]. Thus, the commutative torque is calculated first

$$\texttt{gravity\_torque}_i = \sum_{j=i}^{N} m_j \cdot g \cdot r_{i,j} \cdot \cos(\Theta_i) \tag{89}$$

where the distance to center of mass is given as

$$r_{i,j} = \sum_{k=i}^{j-1} L_k + \frac{L_j}{2} \tag{90}$$

Each arm-links is assumed to be a cylinder. Thus the inertia is given as

$$\texttt{I}_i = \frac{1}{3} m_i \cdot L_i^2 \tag{91}$$

The angular acceleration for the given joint is thus

$$\texttt{gravity\_acc}_i = \frac{\texttt{gravity\_torque}_i}{\texttt{I}_i} \tag{92}$$

And by integrating, it gives the angular change as

$$\texttt{gravity\_displacement}_i = -\frac{1}{2} \cdot \texttt{gravity\_acc}_i \cdot \texttt{dt}^2 \tag{93}$$

Where the -1 is for orienting the force in the negative direction for simulation purposes. This is also added to Eqn. (87) for angular change.

7.3.7 Anti-windup

To ensure that the integral terms of the controller doesn't grow unbounded because of non-linear dynamics, a clamping method is used to combat windup [30].
Given the control output U, current error E, the anti-windup mechanism is defined as follows:

$$\text{is\_saturated} = |U(t)| \geq 255 \tag{94}$$

$$\text{is\_winding\_up} = \text{sign}(E(t)) == \text{sign}(U(t)) \tag{95}$$

$$\text{is\_error\_growing} = |E(t)| > |E(t-1)| \tag{96}$$

Where the clamper is defined as the negated union of the boolean parameters.

$$\text{clamper} = \neg(\text{is\_saturated} \wedge \text{is\_winding\_up} \wedge \text{is\_error\_growing}) \tag{97}$$

Which gives the integral as

$$\text{integral}(t) = \begin{cases} \text{integral}(t-1) + \text{error}(t) \cdot \Delta t, & \text{if clamper} = \text{true} \\ \text{integral}(t-1), & \text{otherwise} \end{cases} \tag{98}$$

7.3.8 Logging and plotting

The logging of values is done by writing the relevant values during the control loop to file for each simulation instance conducted. For plotting the relevant algorithm values, a python script is used, which can be inspected on the associated GitHub repository [20]. This script plots the ICO learning weights and the PID-gain values, which are fetched from file, for every iteration.

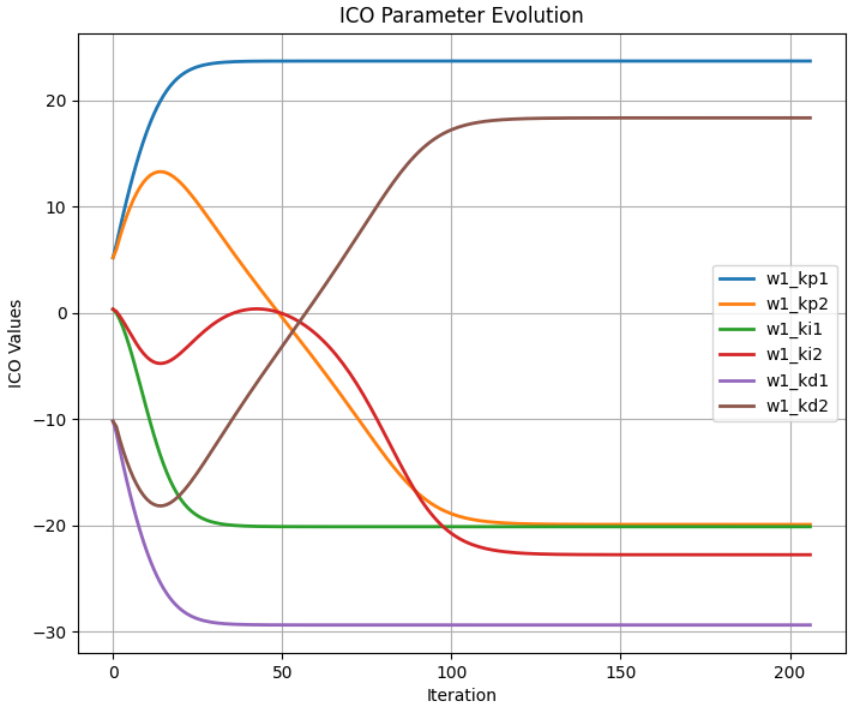


Fig. 30: Plotting example for ICO learning weights using Python

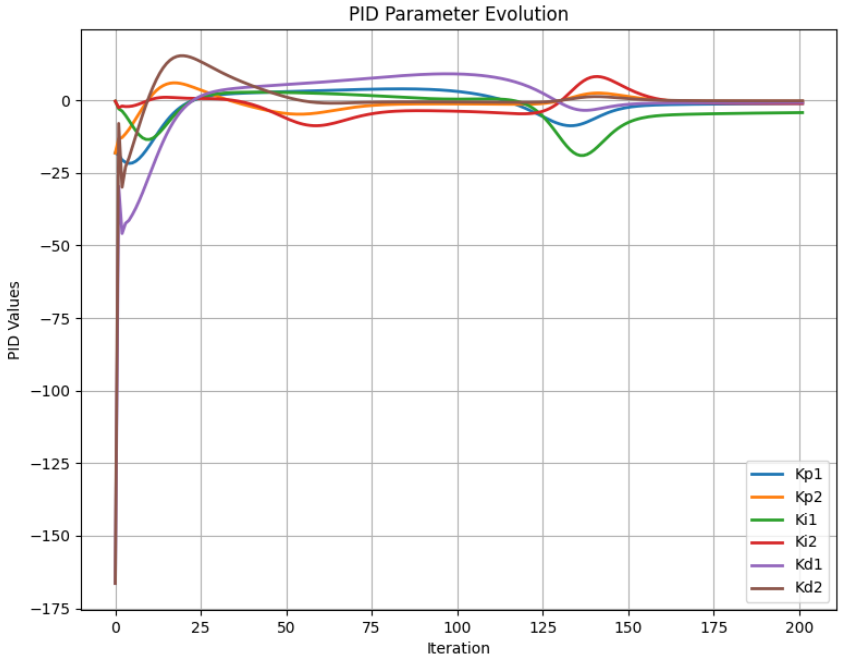


Fig. 31: Plotting example for PID-gain values using Python

7.4 Appendix 4: Relay feedback auto tuning case study

Relay feedback auto tuning

A previously conducted case study of relay feedback auto tuning of a PID controller on a LEGO robot car in collaboration with Peter L. Almvig and Simon A. Andreasen [1]. The final case study is available in the attached ZIP file.

7.5 Appendix 5: Simulation test scenarios

7.5.1 Differing learning rates

This section of scenarios explores how the unfiltered system behaves at different tuned learning rates (μ), while the initial position is also the target position and using a sampling rate of $0.1s$.

S1.1 $\mu = 0.1$**.**

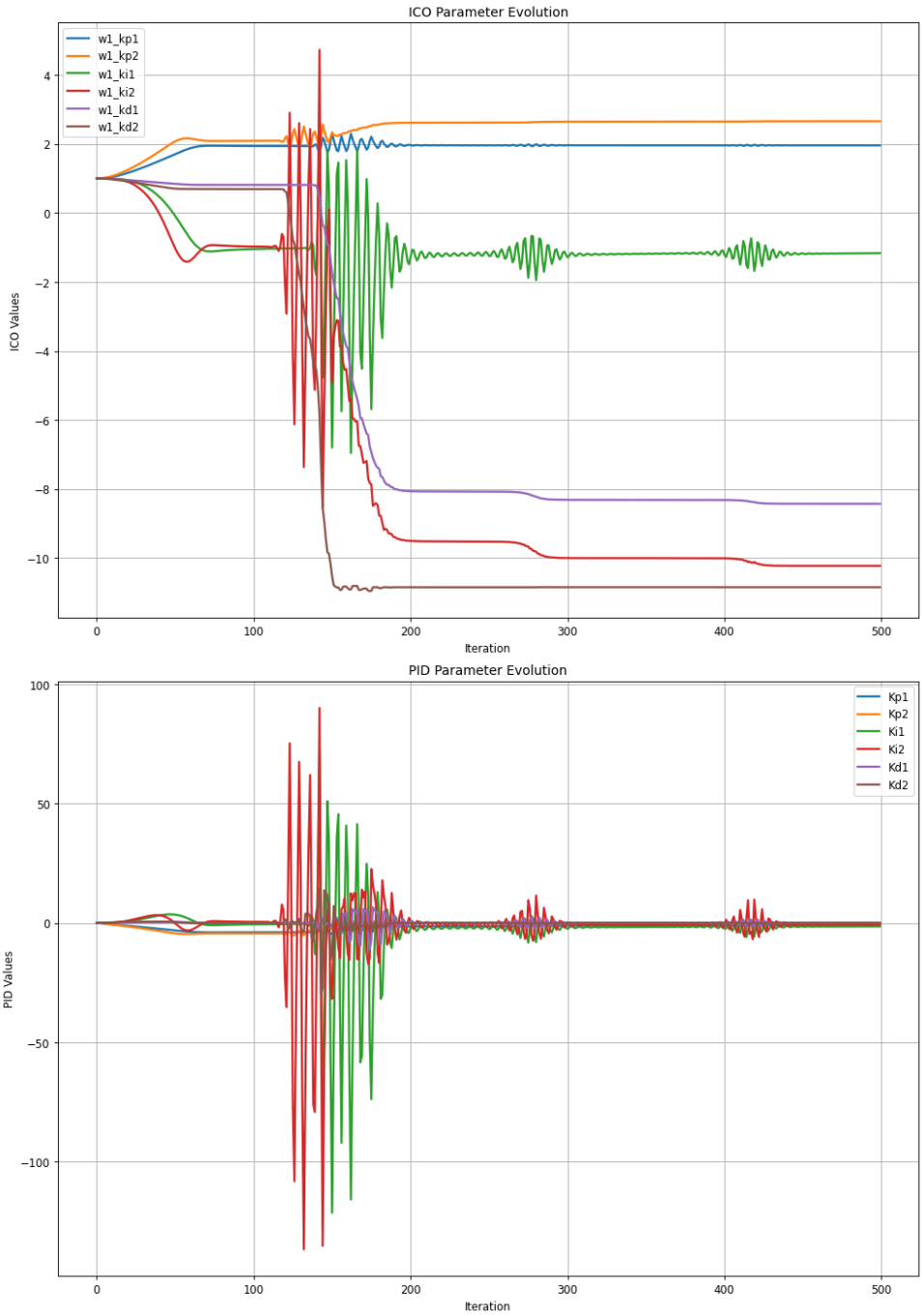


Fig. 32: Evolution of ICO and PID parameters for test scenario S1.1, using $\mu = 0.1$

S1.2 $\mu = 1$**.**

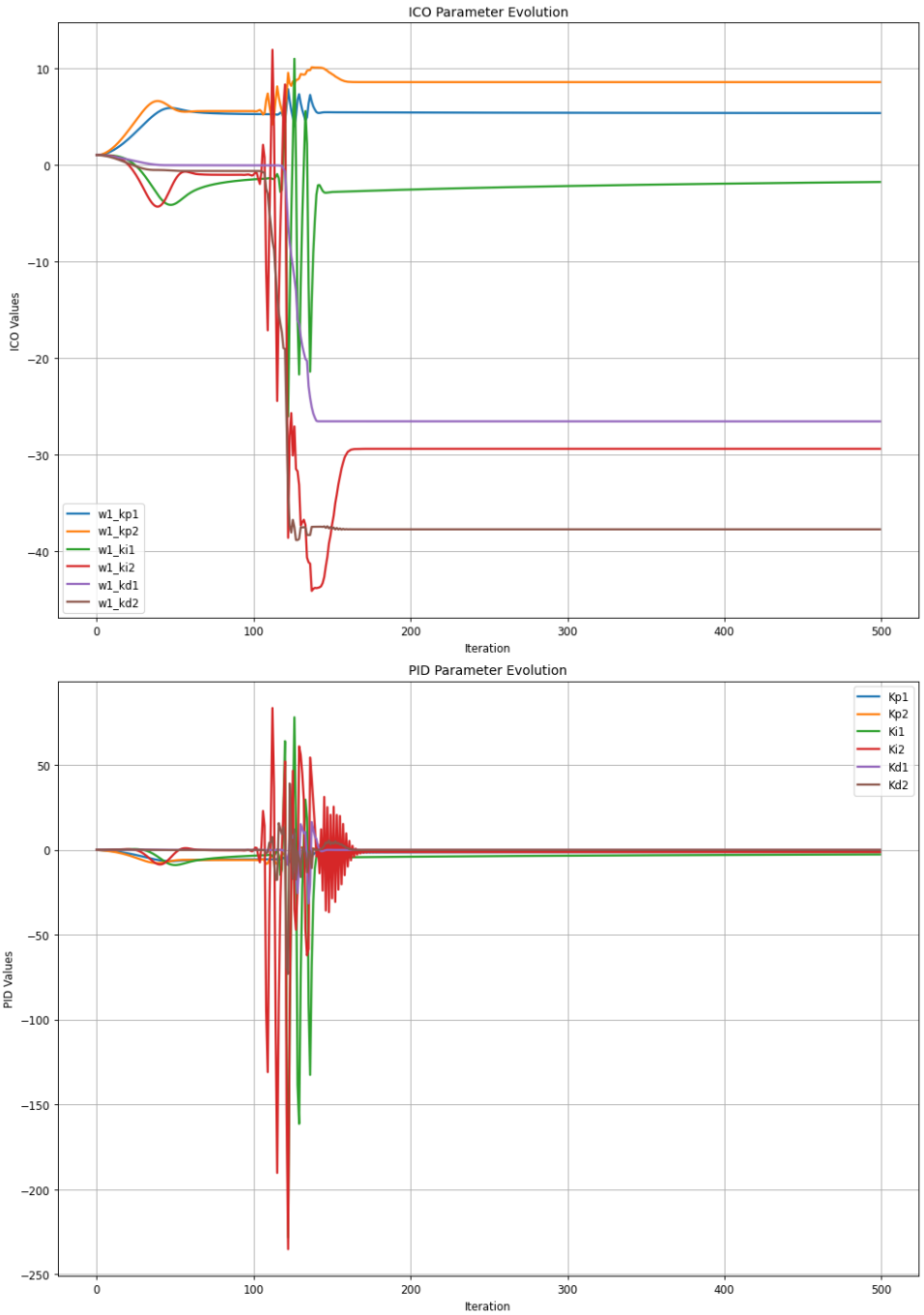


Fig. 33: Evolution of ICO and PID parameters for test scenario S1.2, using $\mu = 1$

S1.3 $\mu = 3$**.**

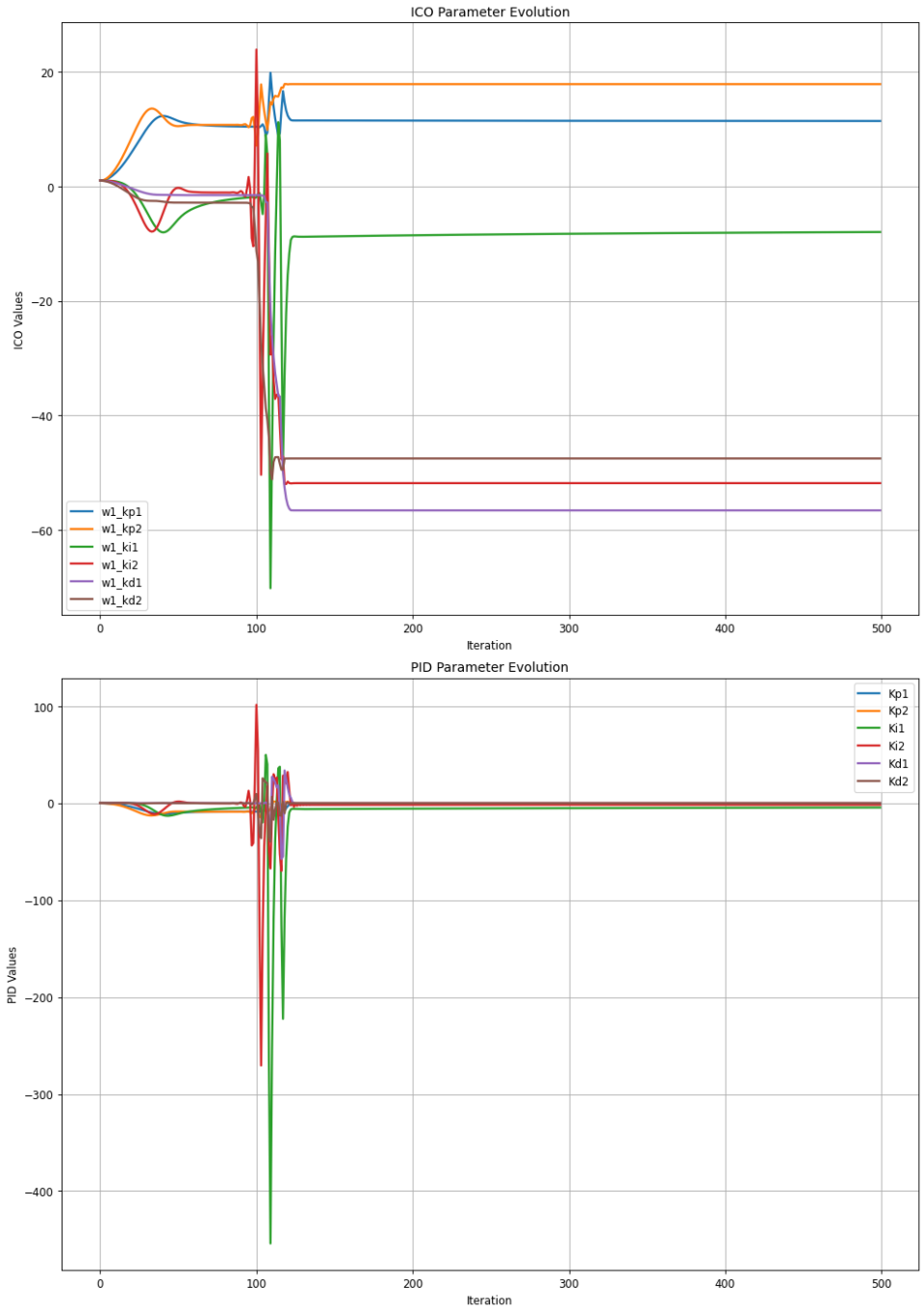


Fig. 34: Evolution of ICO and PID parameters for test scenario S1.3, using $\mu = 3$

With this initial position and algorithm sampling rate, tuning of the learning rate has a visual impact on the system. All three values tested converge in the end to stable values, but a noticeable windup occurs as the learning rate increases.

7.5.2 Differing sample rates

This section of scenarios explores how the unfiltered system behaves at different tuned sampling rates (μ), while the initial position is also the target position and using a learning rate of 1.

S2.1 $dt = 0.05\ s$

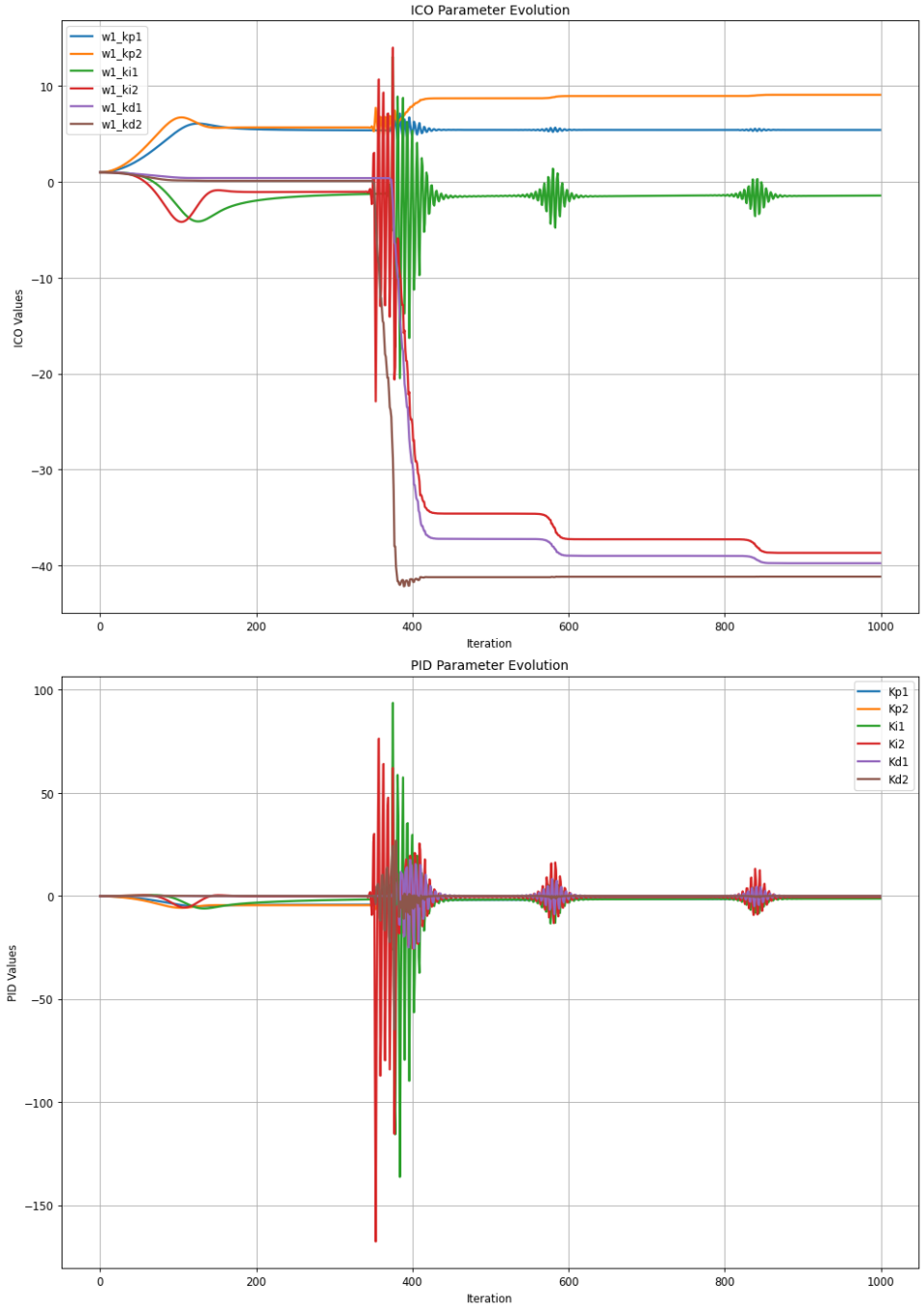


Fig. 35: Evolution of ICO and PID parameters for test scenario S2.1, using $dt = 0.05\ s$

S2.2 $dt = 0.08\ s$

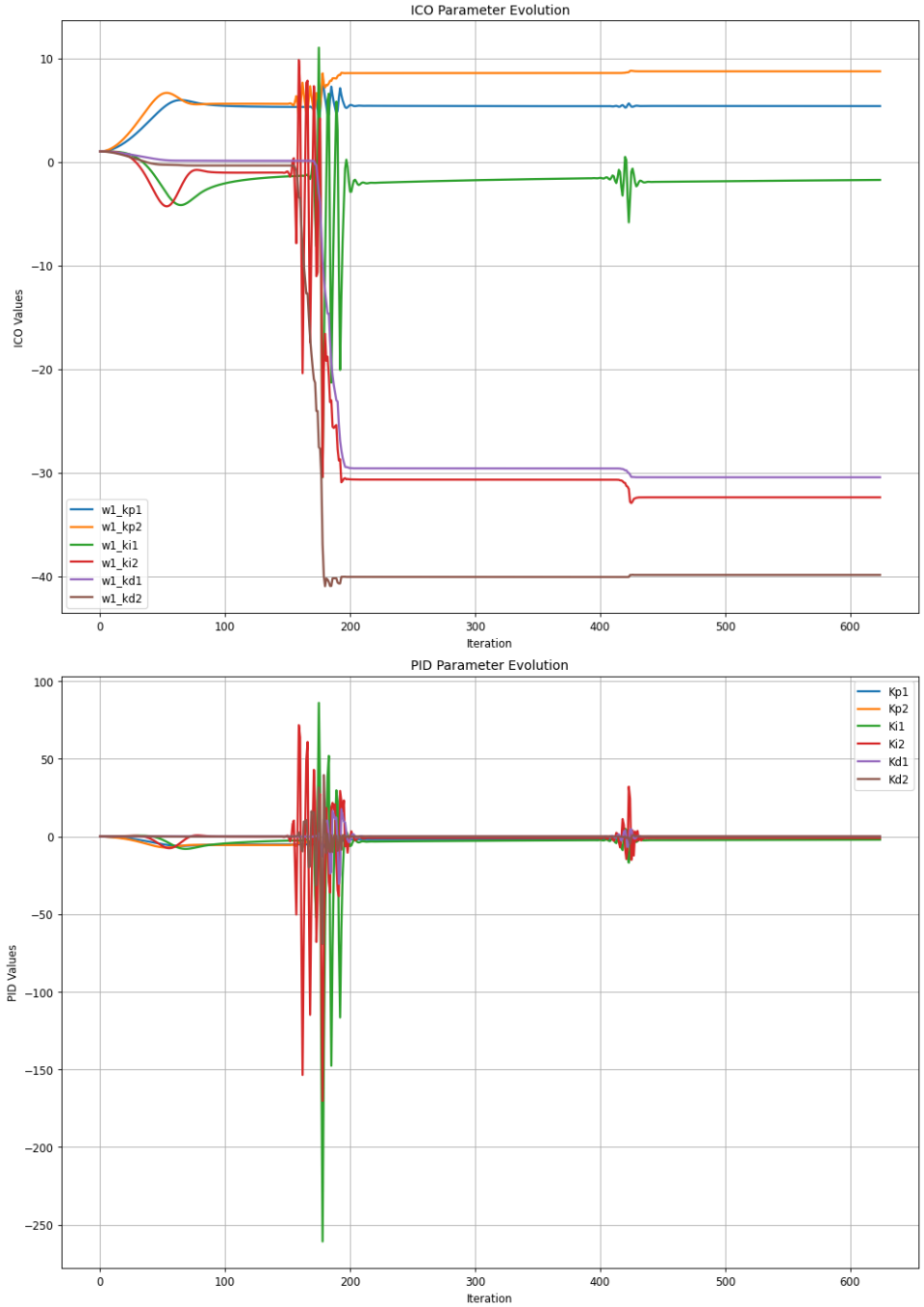


Fig. 36: Evolution of ICO and PID parameters for test scenario S2.2, using $dt = 0.08\ s$

S2.3 $dt = 0.15\ s$

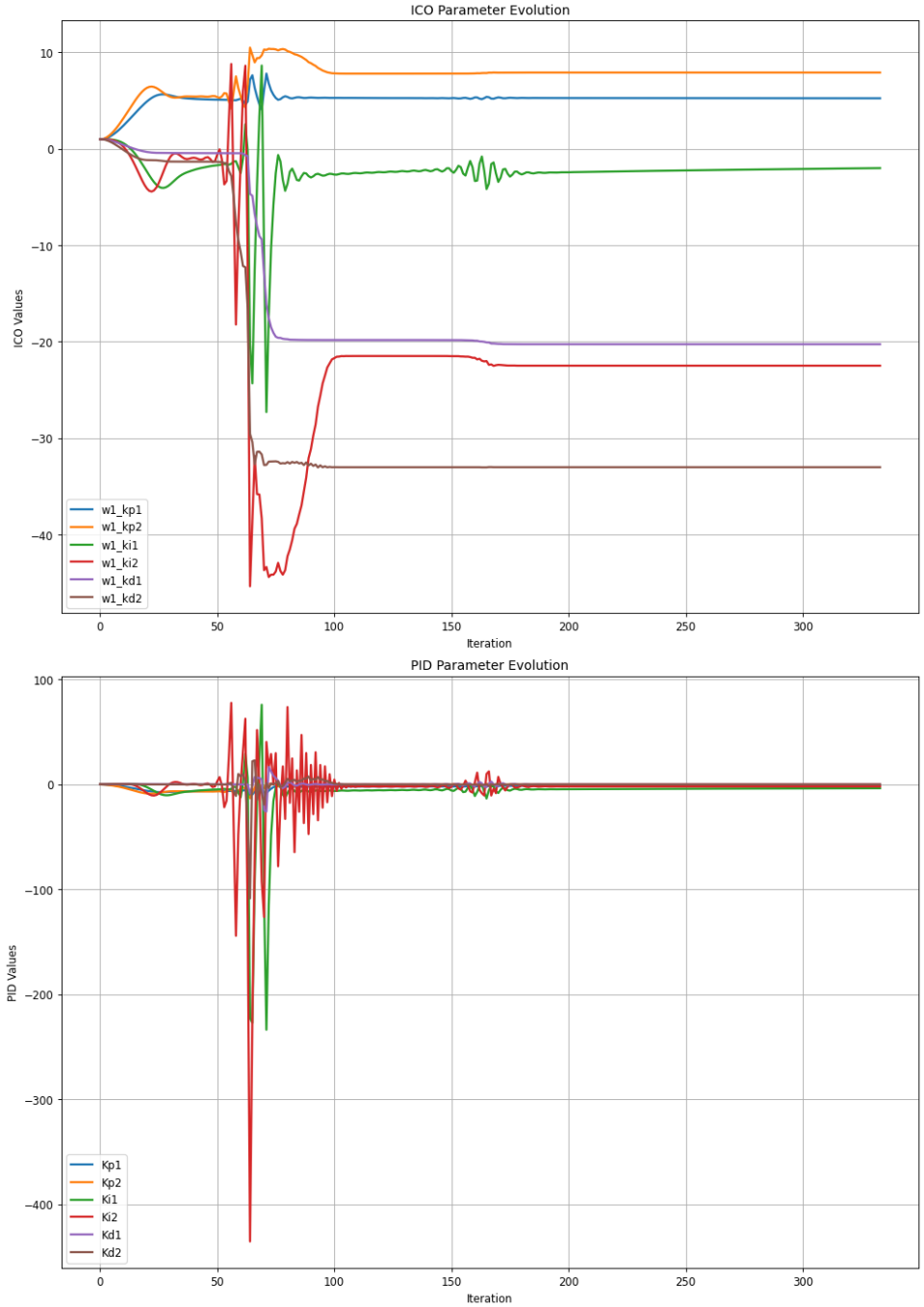


Fig. 37: Evolution of ICO and PID parameters for test scenario S2.3, using $dt = 0.15\ s$

7.5.3 Differing initial error

This section of scenarios explores how the unfiltered system behaves at different setpoints (R) / initial errors ($E(0)$), while the learning rate of 1 and a sampling rate of 0,1 seconds.

S3.1 $R = 0.2\pi$

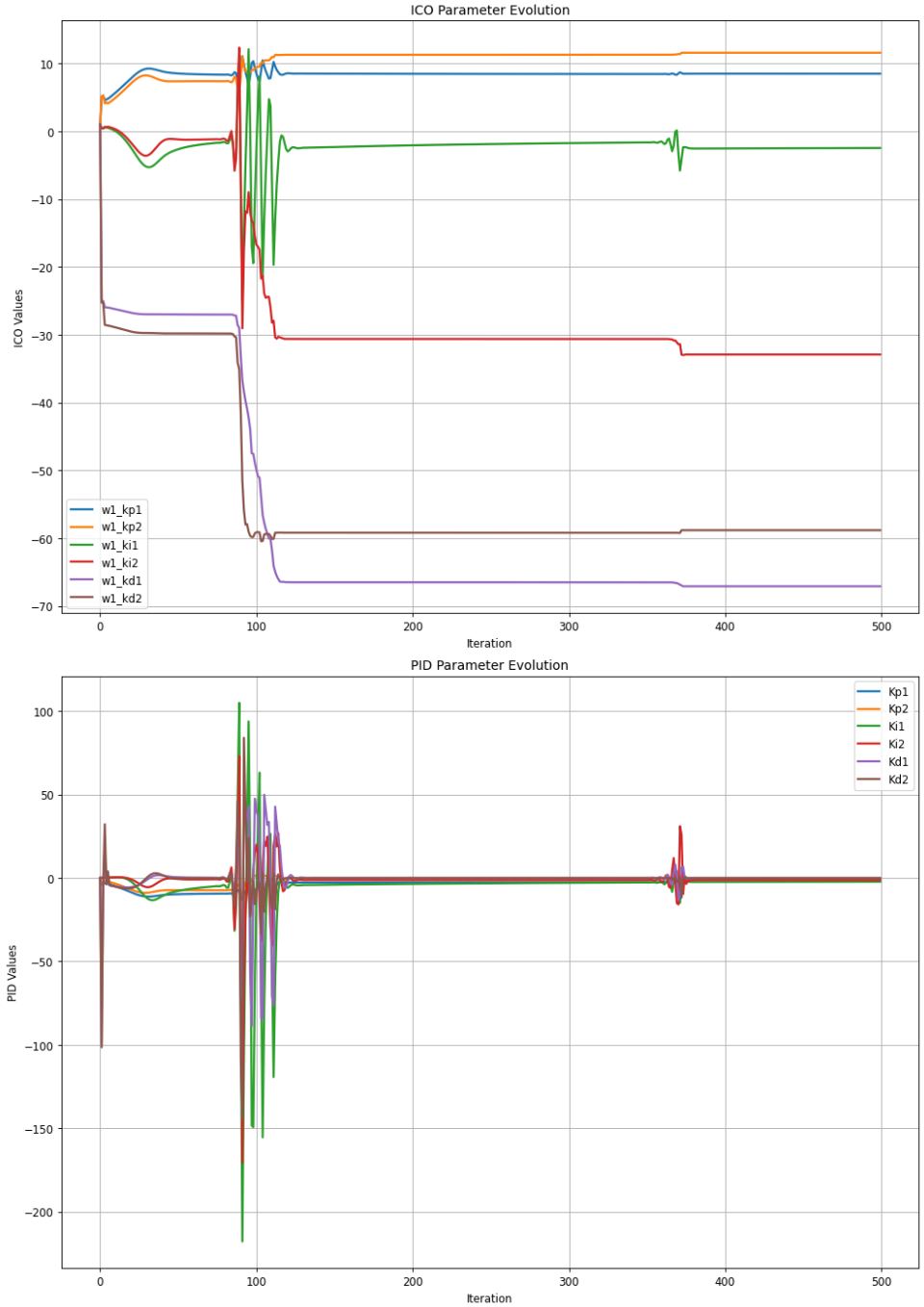


Fig. 38: Evolution of ICO and PID parameters for test scenario S3.1, using $R = 0.2\pi$

S3.2 $R = 0.5\pi$

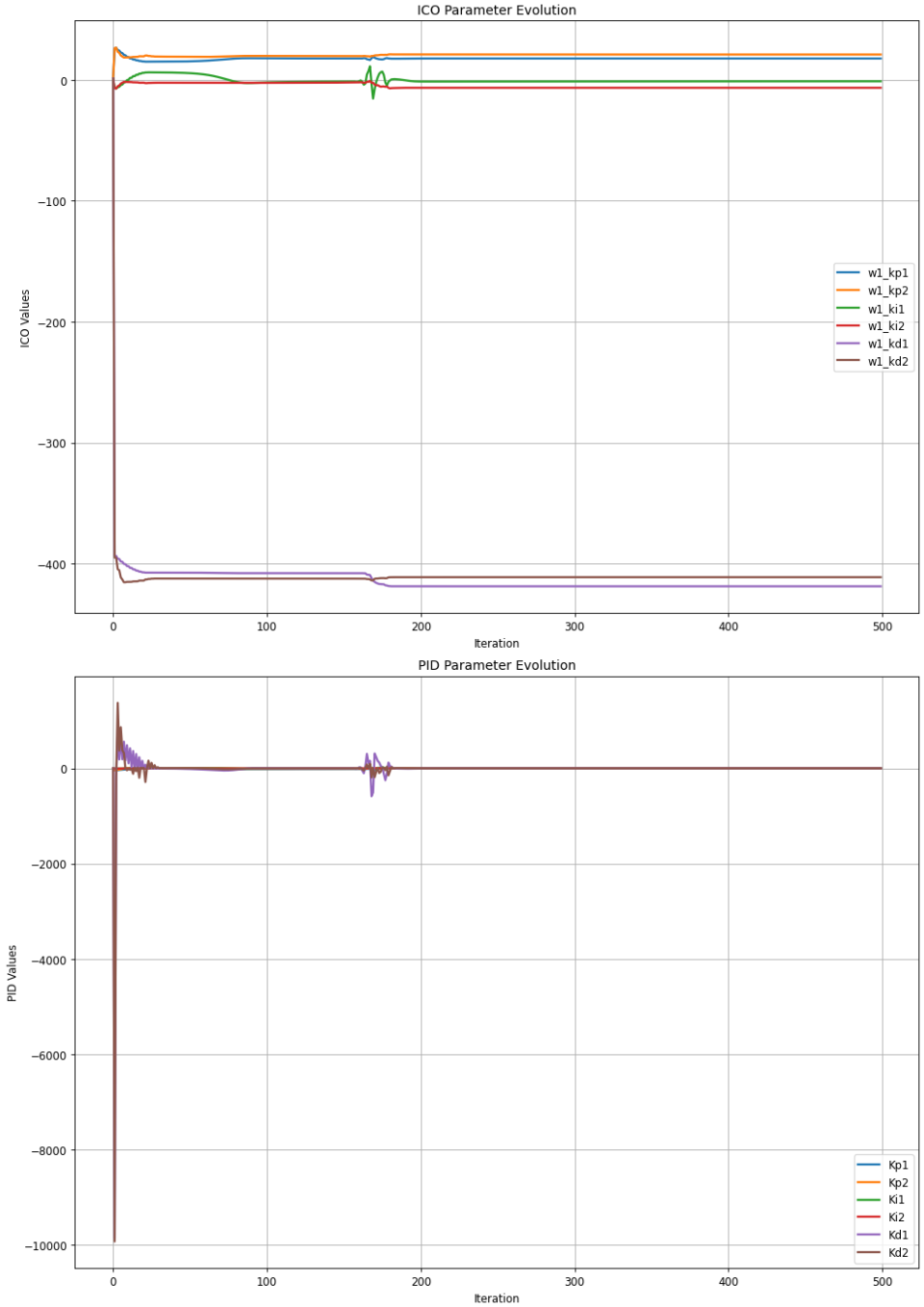


Fig. 39: Evolution of ICO and PID parameters for test scenario S3.2, using $R = 0.5\pi$

S3.3 $R = \pi$

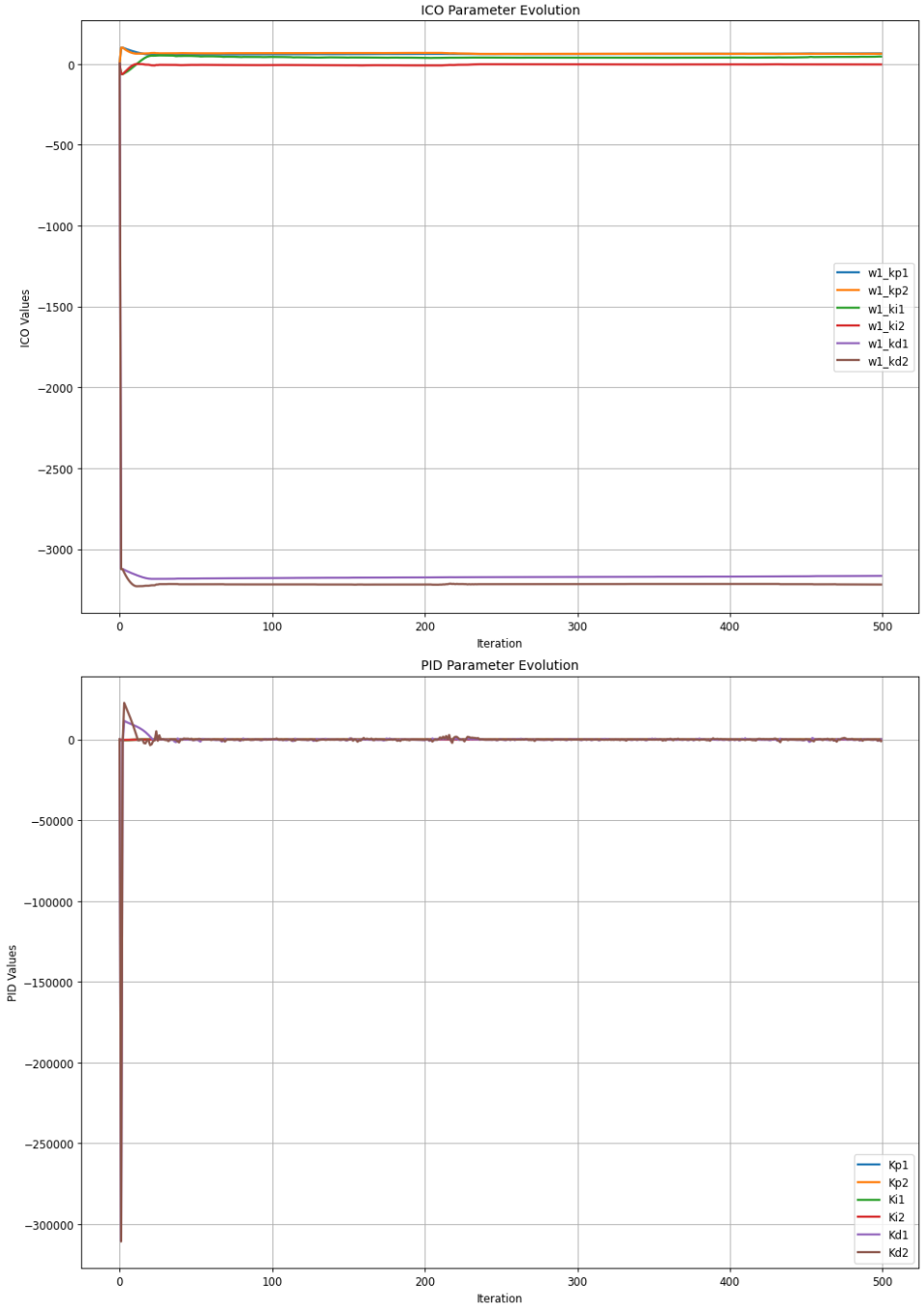


Fig. 40: Evolution of ICO and PID parameters for test scenario S3.3, using $R = \pi$

7.5.4 Differing learning rates

This section of scenarios explores how the filtered systems behave at different tuned learning rates (μ), while the initial position is also the target position and using a sampling rate of 0,1s.

K_d model filtering

S4.1 $\mu = 0.1$**.**

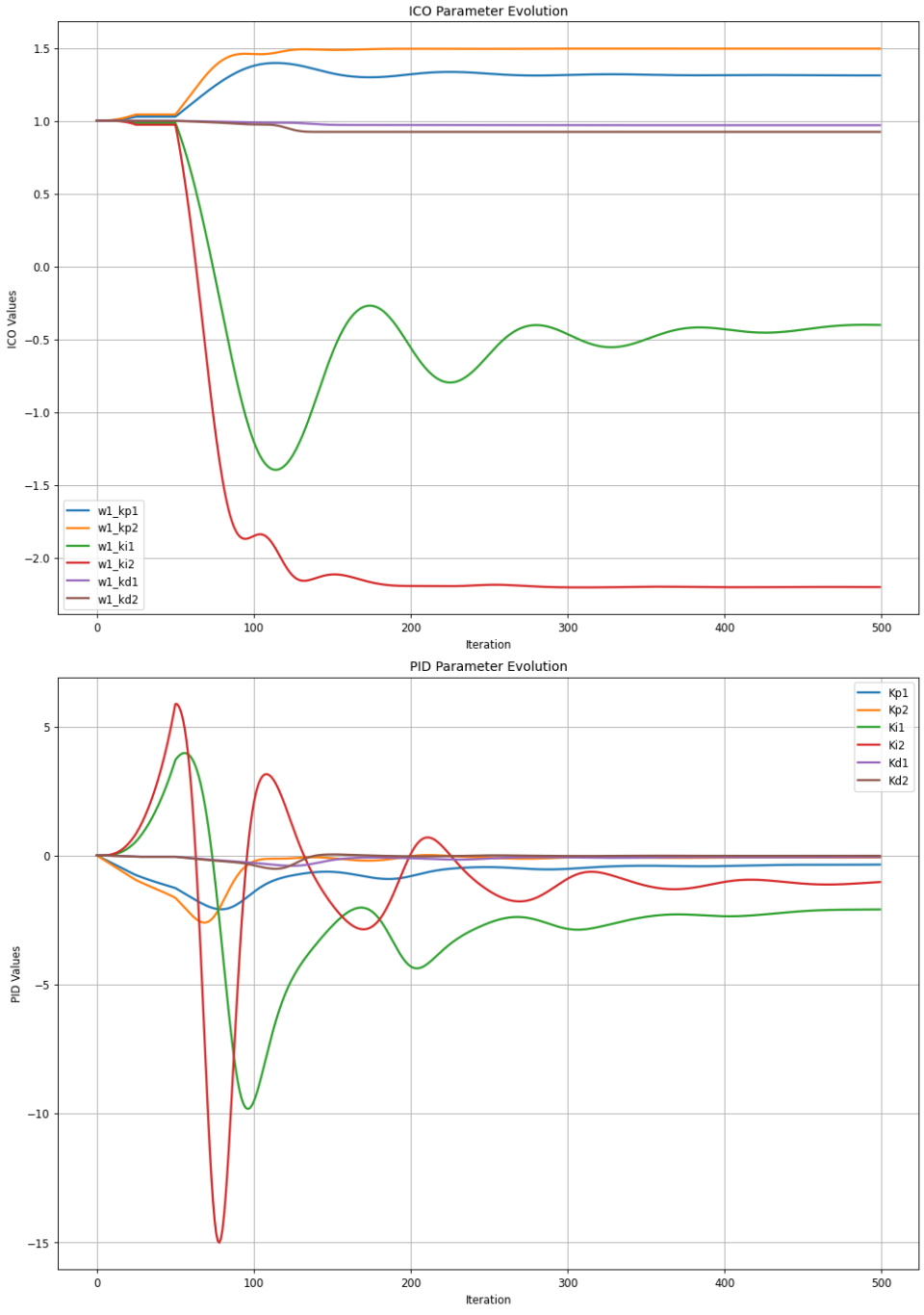


Fig. 41: Evolution of ICO and PID parameters for test scenario S4.1, using $\mu = 0.1$.

S4.2 $\mu = 1$**.**

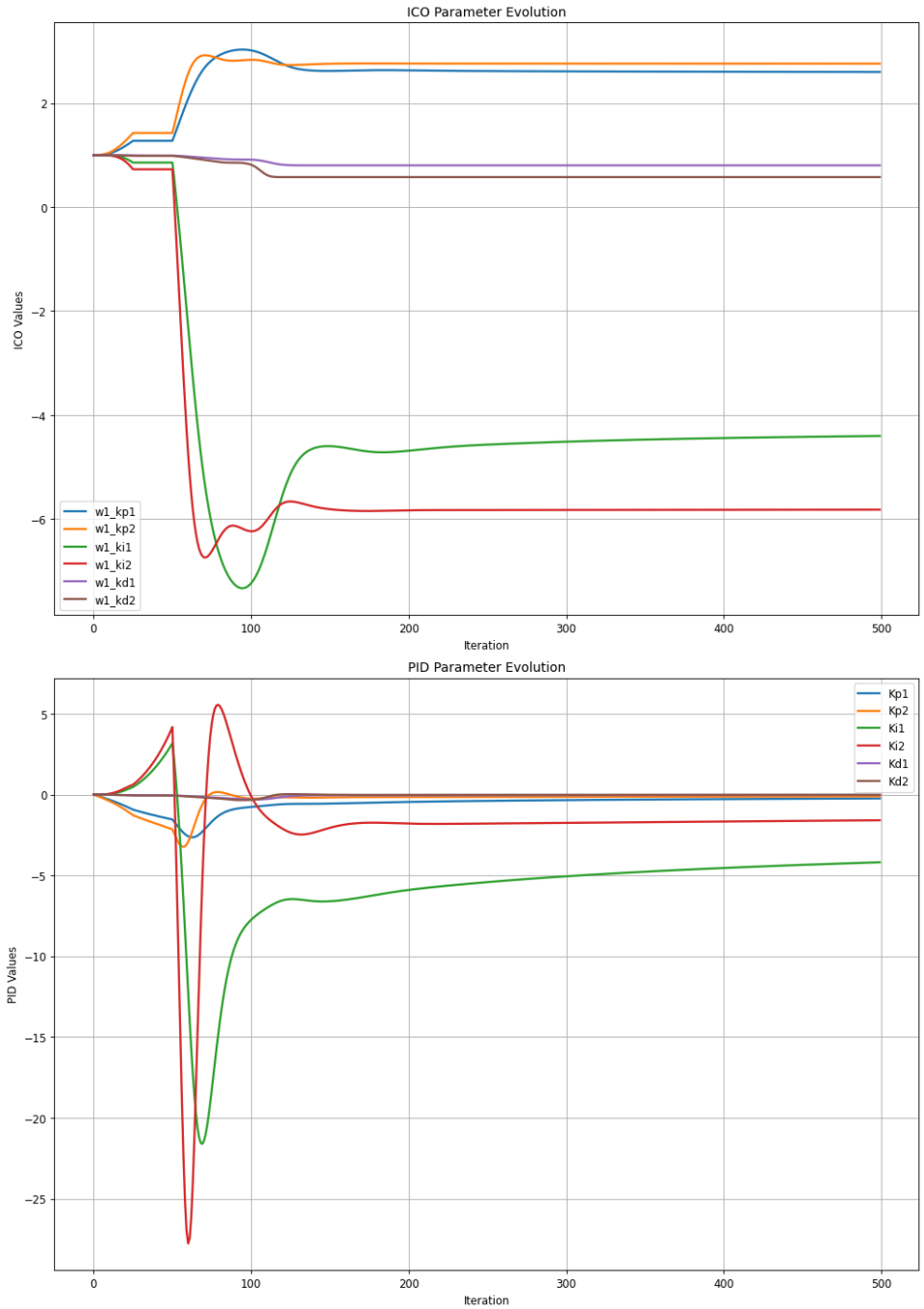


Fig. 42: Evolution of ICO and PID parameters for test scenario S4.2, using $\mu = 1$.

S4.3 $\mu = 3$.

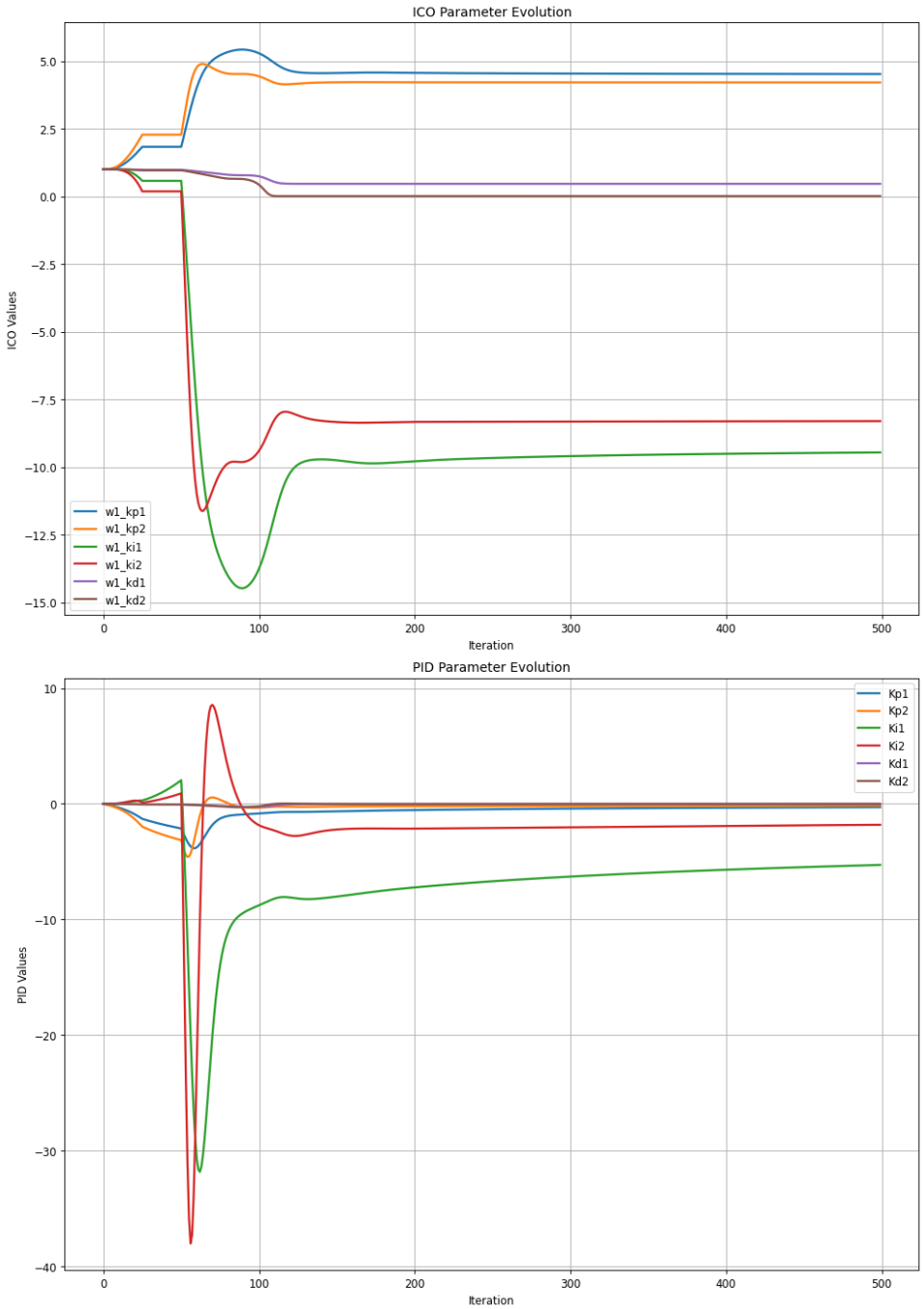


Fig. 43: Evolution of ICO and PID parameters for test scenario S4.3, using $\mu = 3$.

ICO derivative filtering

S5.1 $\mu = 0.1$**.**

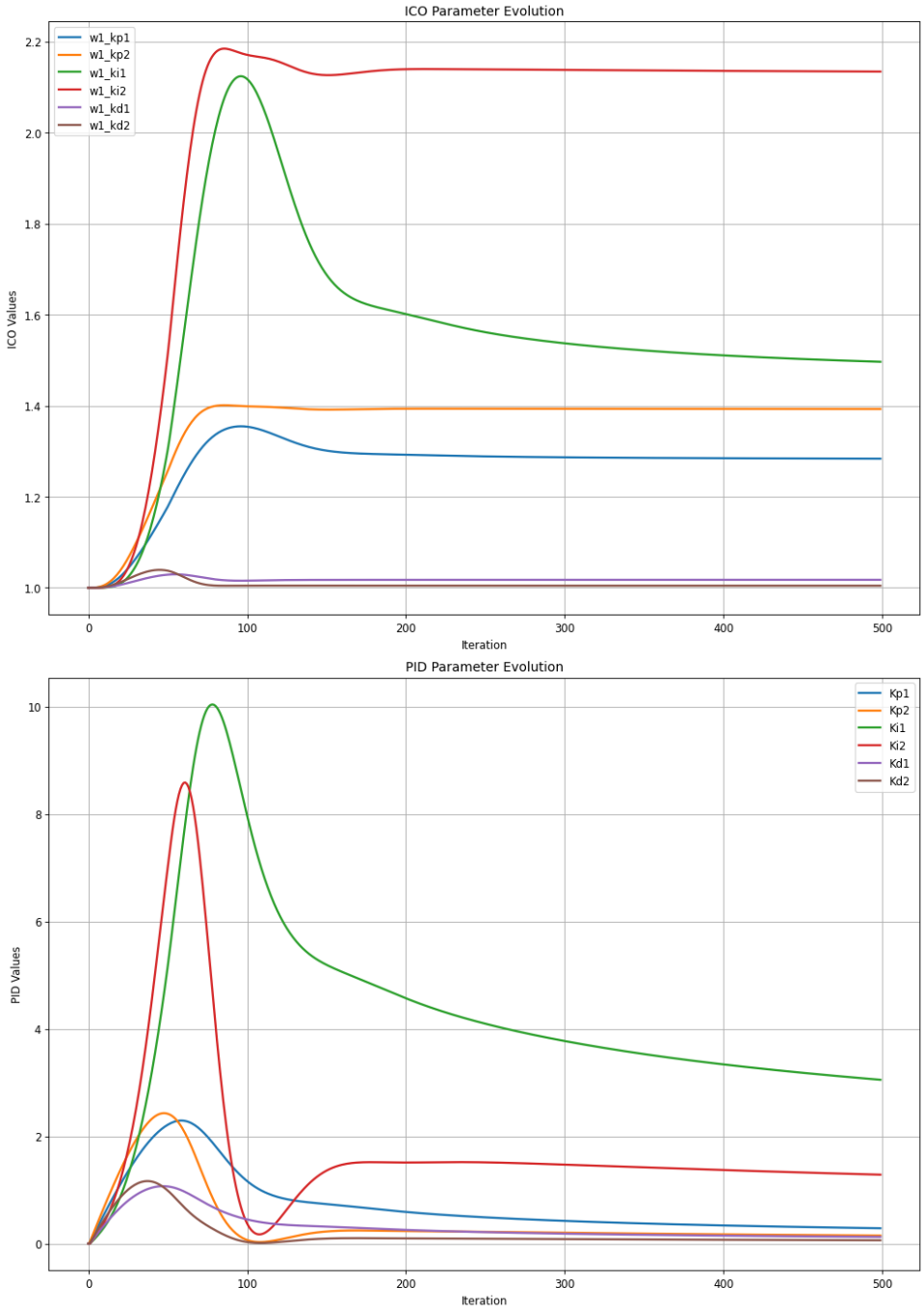


Fig. 44: Evolution of ICO and PID parameters for test scenario S5.1, using $\mu = 0.1$.

S5.2 $\mu = 1$**.**

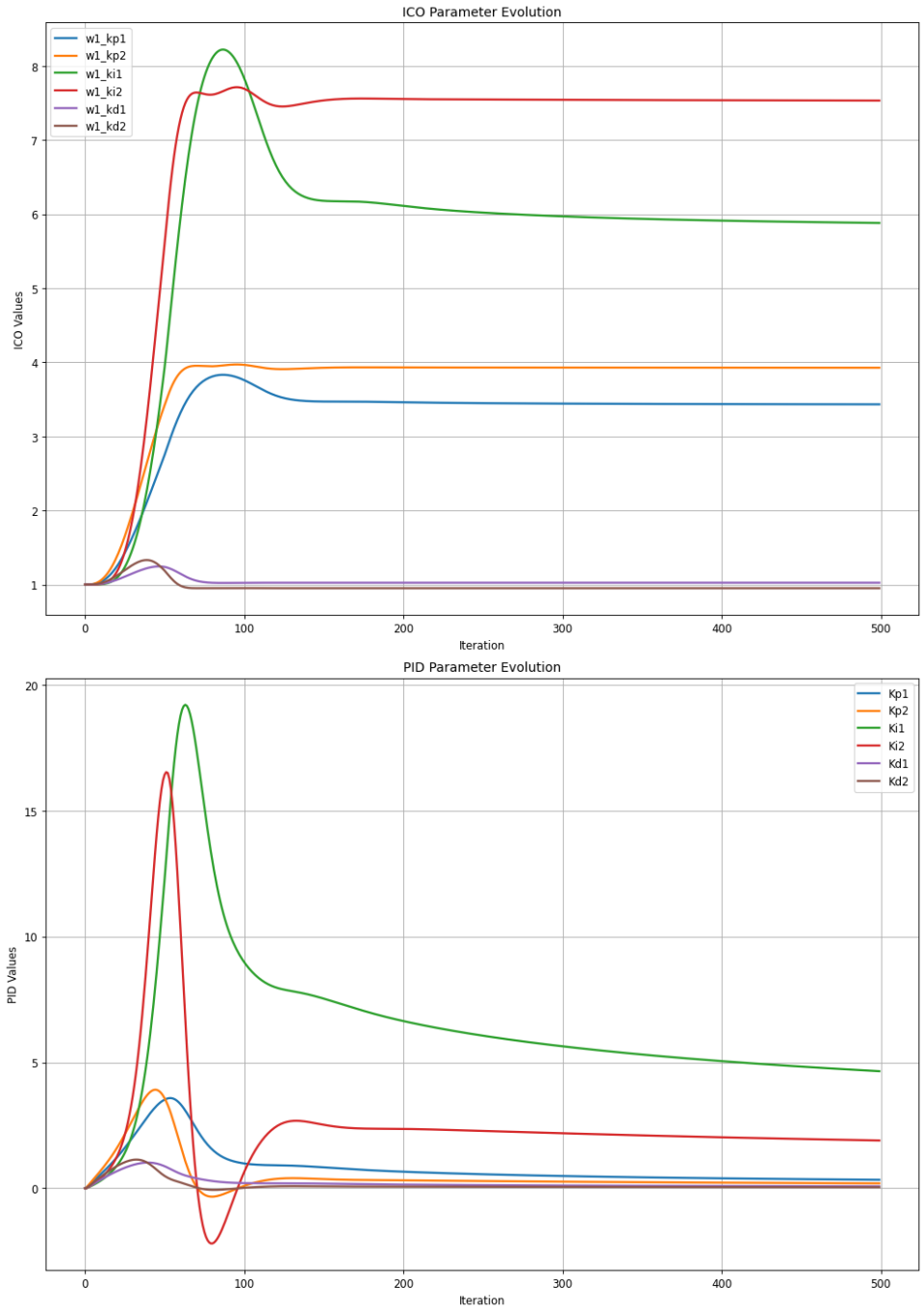


Fig. 45: Evolution of ICO and PID parameters for test scenario S5.2, using $\mu = 1$.

S5.3 $\mu = 3$.

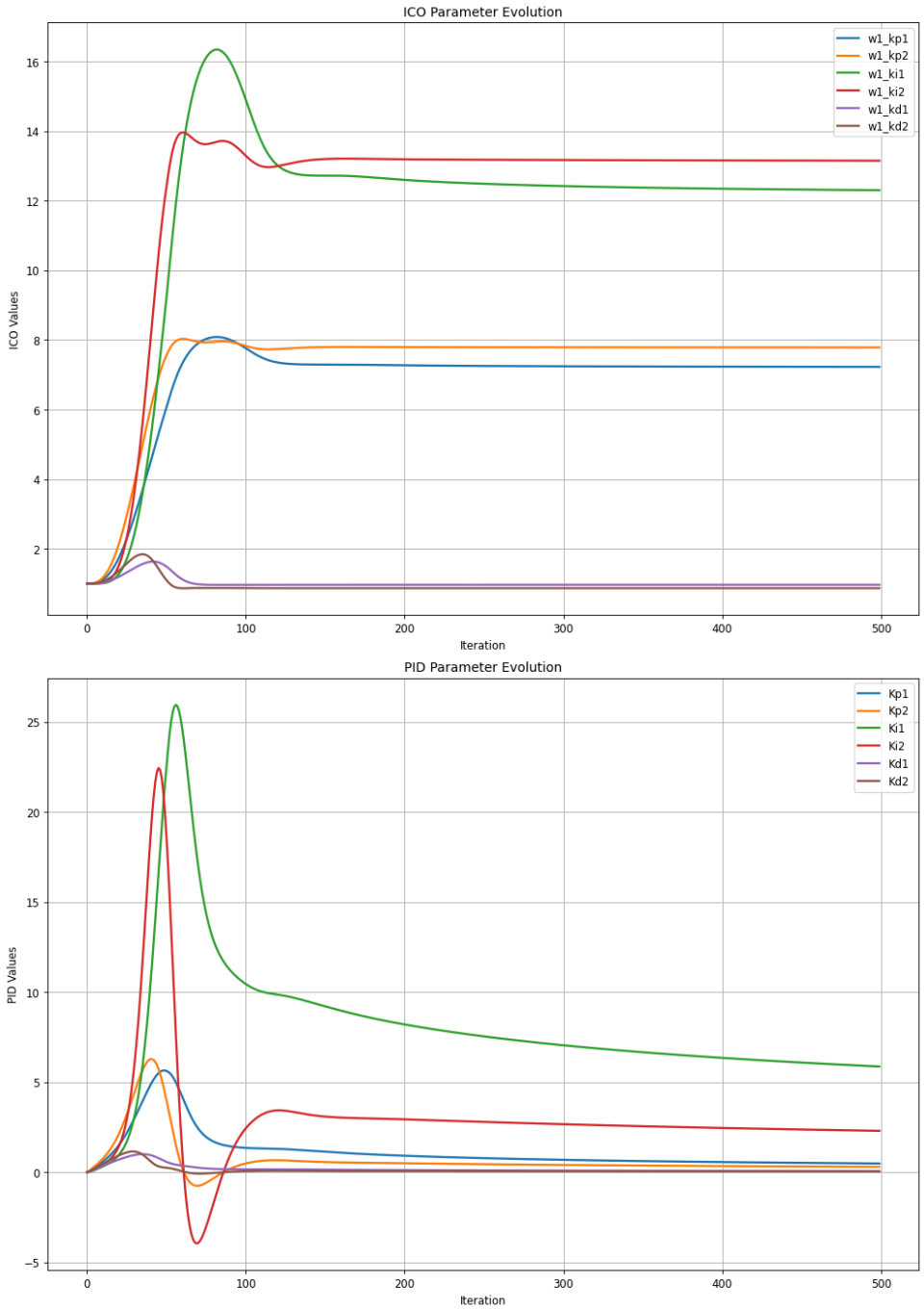


Fig. 46: Evolution of ICO and PID parameters for test scenario S5.3, using $\mu = 3$.

7.5.5 Differing sample rates

This section of scenarios explores how the filtered systems behave at different tuned sampling rates (dt), while the initial position is also the target position and using a learning rate of 1.

K_d model filtering

S6.1 $dt = 0.05\ s$

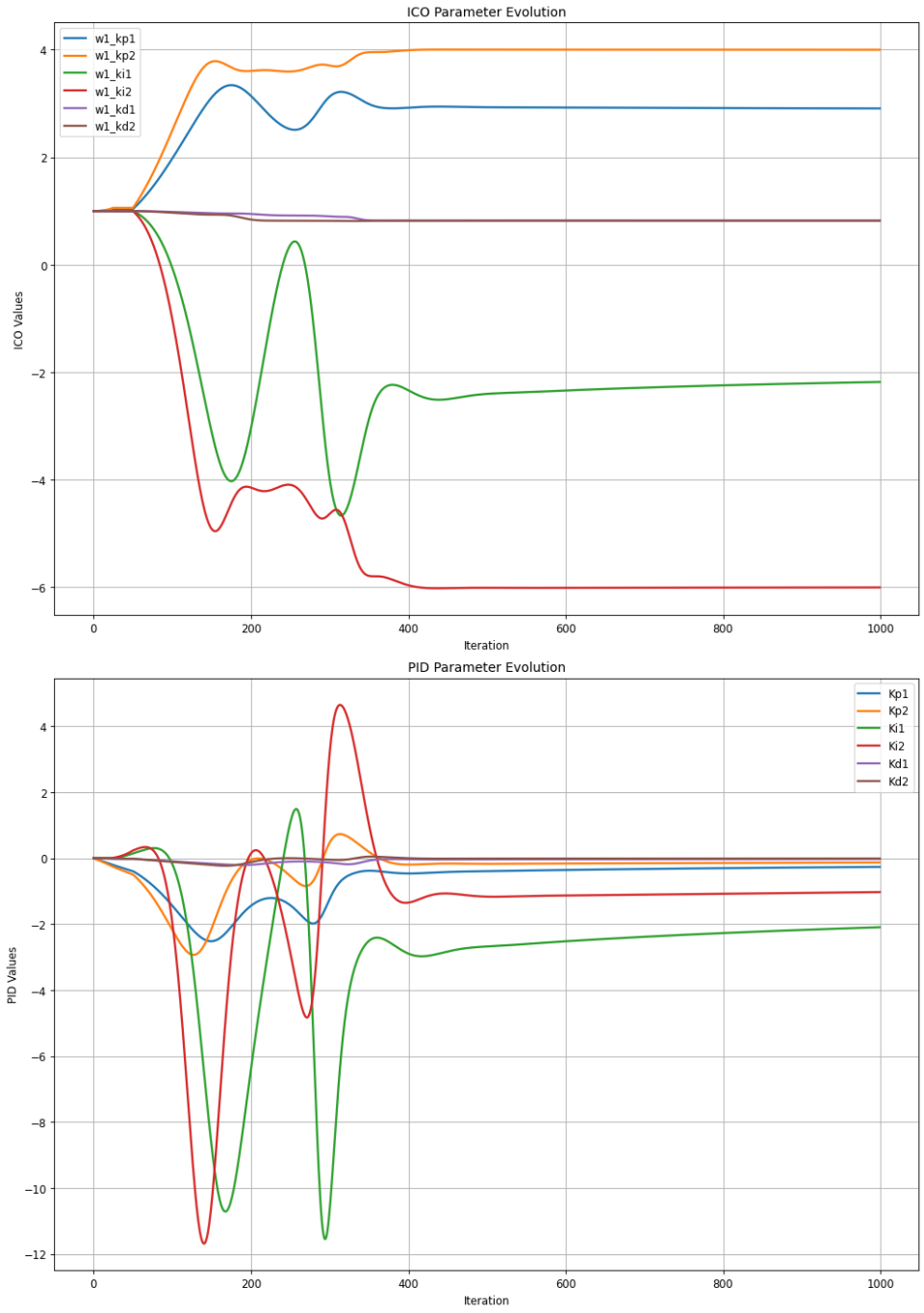


Fig. 47: Evolution of ICO and PID parameters for test scenario S6.1, using $dt = 0.05\ s$.

S6.2 $dt = 0.08\ s$

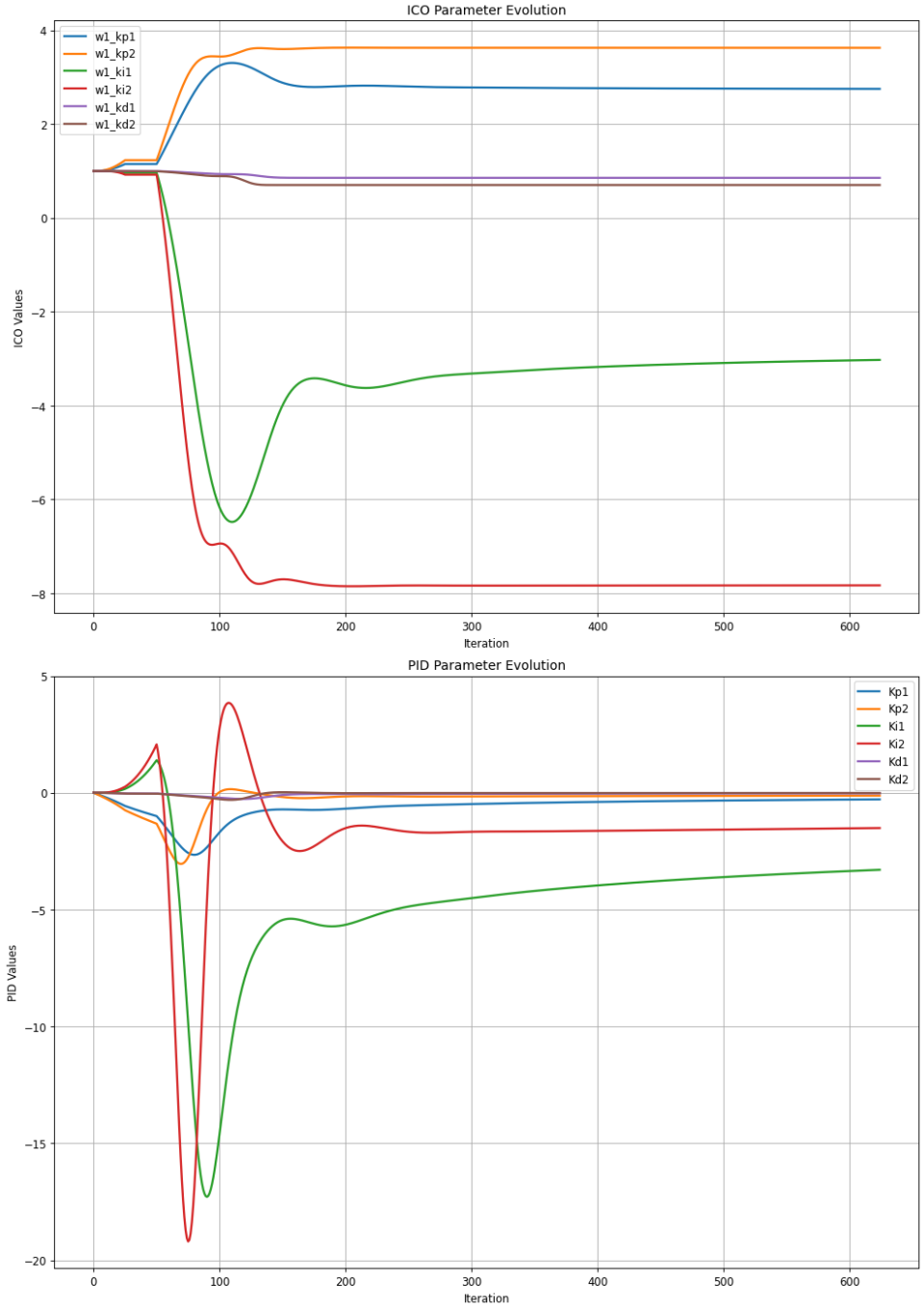


Fig. 48: Evolution of ICO and PID parameters for test scenario S6.2, using $dt = 0.08\ s$.

S6.3 $dt = 0.15\ s$

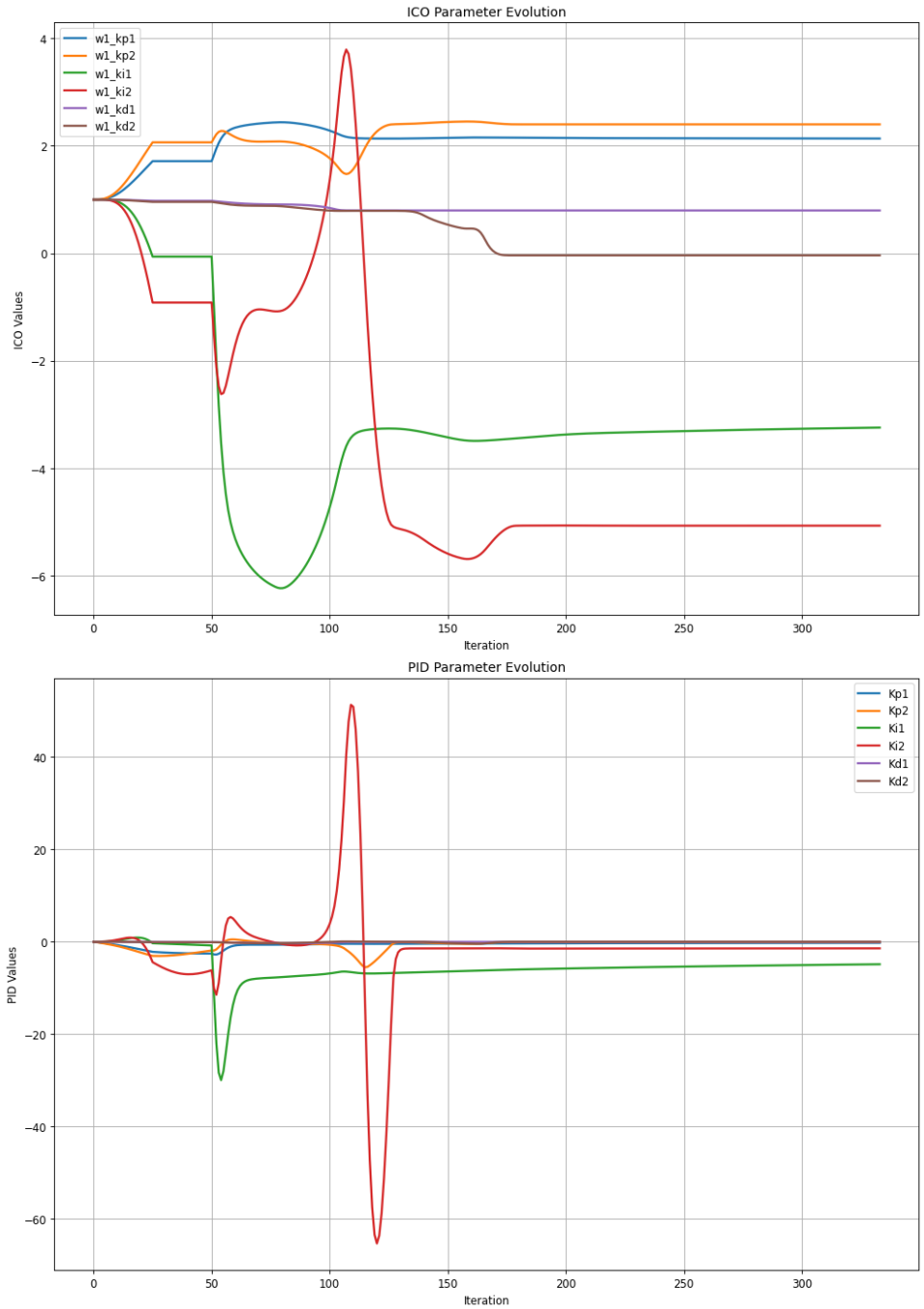


Fig. 49: Evolution of ICO and PID parameters for test scenario S6.3, using $dt = 0.15\ s$.

ICO derivative filtering

S7.1 $dt = 0.05\ s$

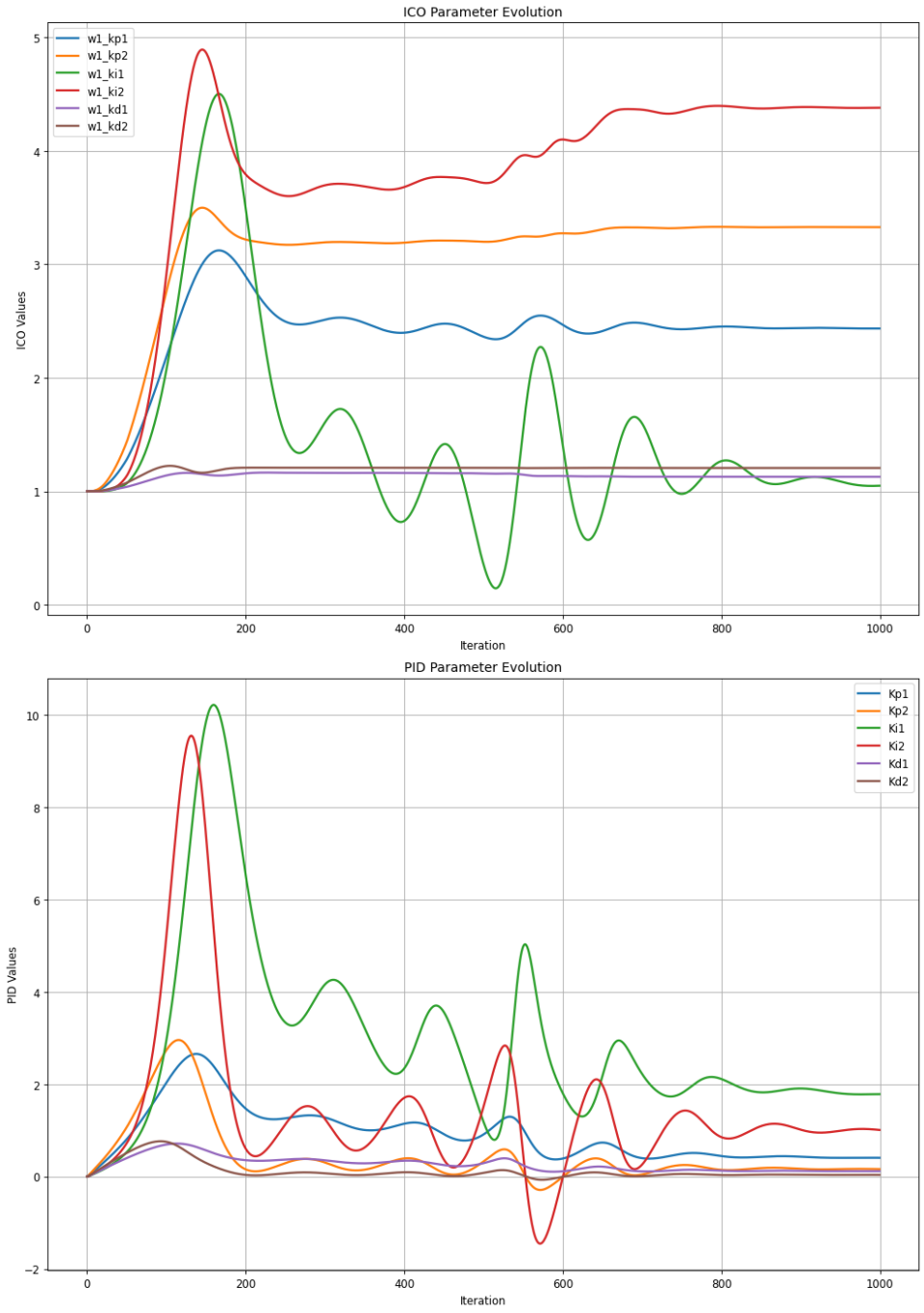


Fig. 50: Evolution of ICO and PID parameters for test scenario S7.1, using $dt = 0.05\ s$.

S7.2 $dt = 0.08\ s$

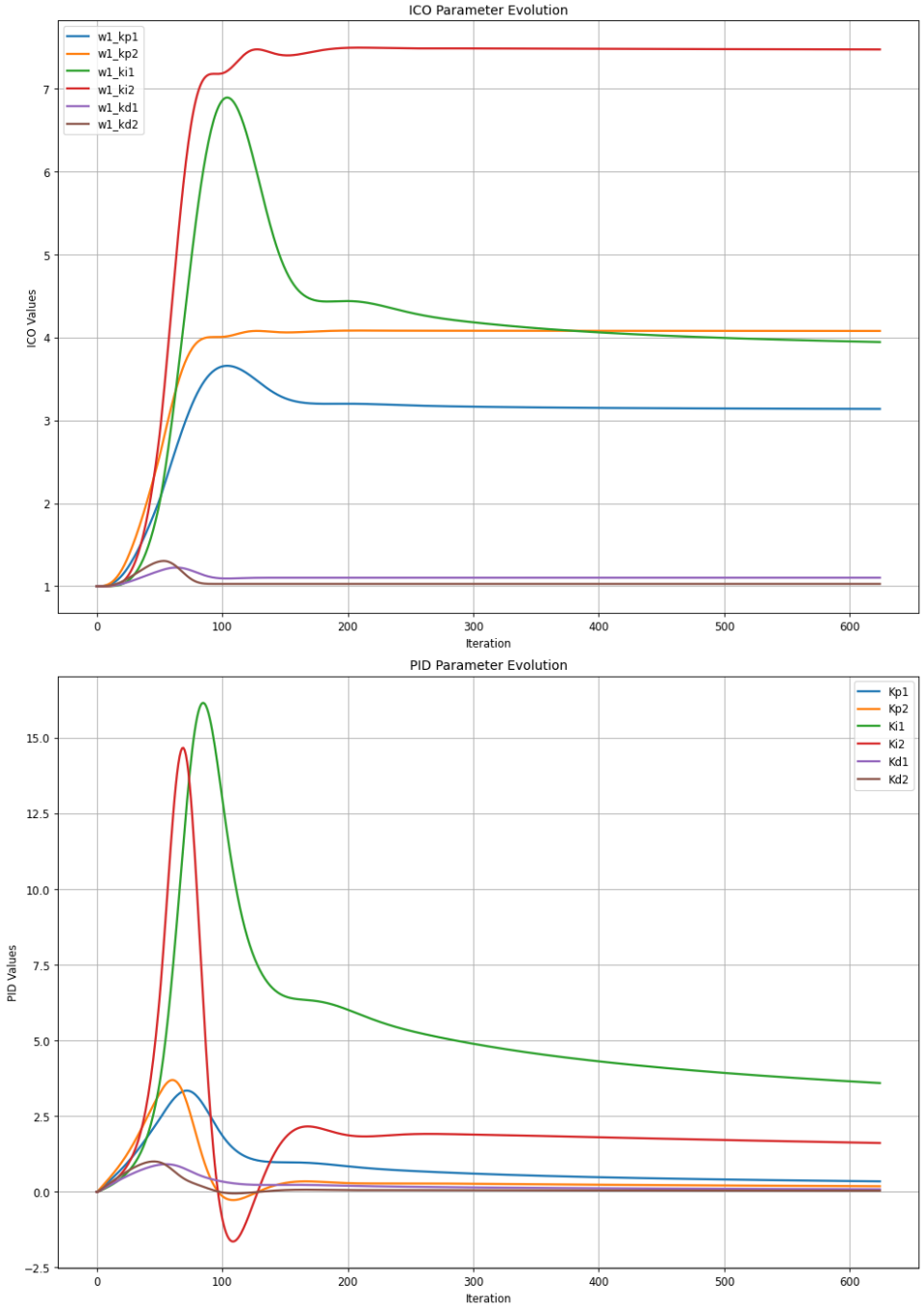


Fig. 51: Evolution of ICO and PID parameters for test scenario S7.2, using $dt = 0.08\ s$.

S7.3 $dt = 0.15\ s$

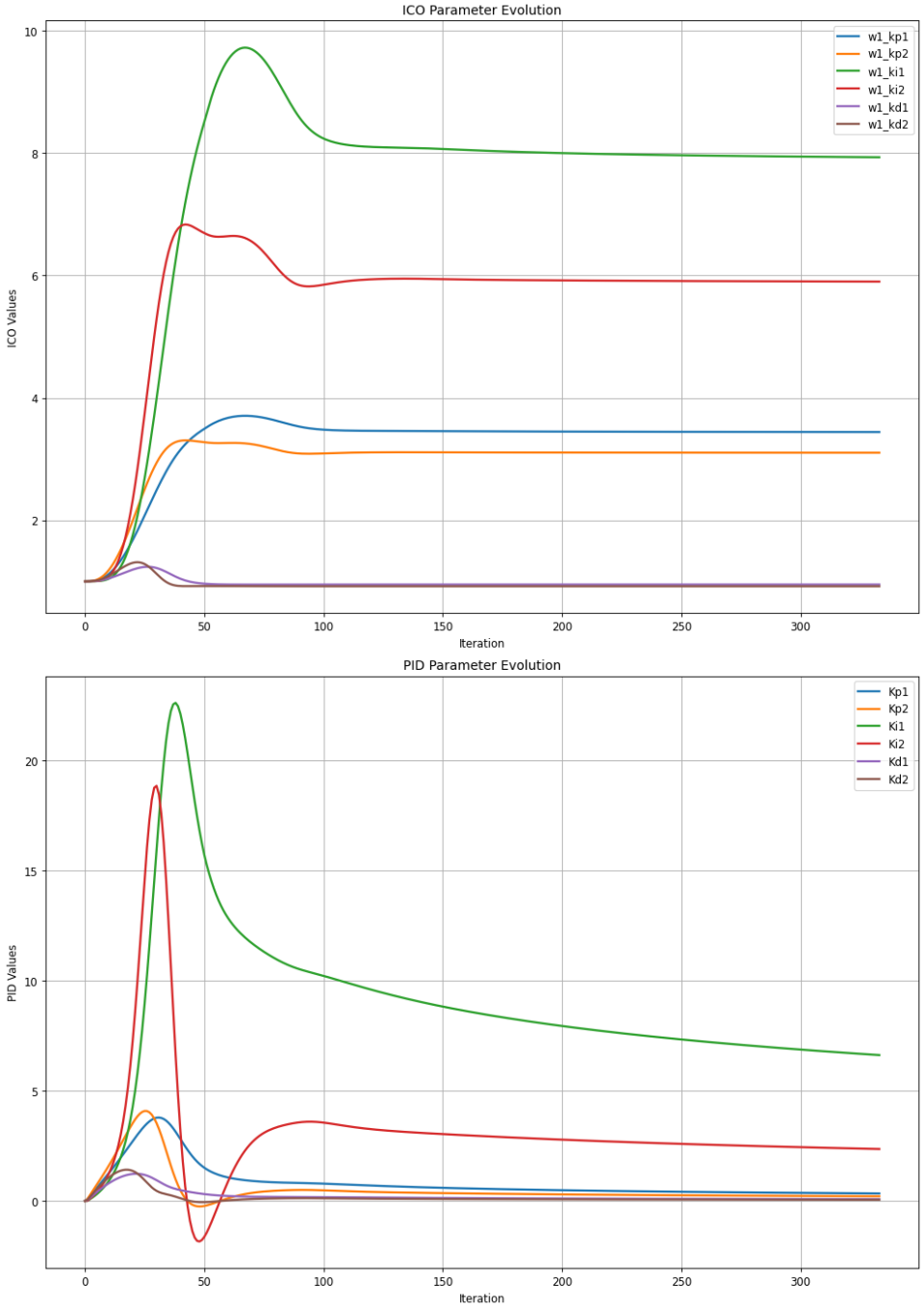


Fig. 52: Evolution of ICO and PID parameters for test scenario S7.3, using $dt = 0.15\ s$.

7.5.6 Differing initial error

This section of scenarios explores how the filtered systems behave at different setpoints (R) / initial errors ($E(0)$), with the learning rate of 1 and a sampling rate of 0,1 seconds.

K_d model filtering

S8.1 $R = 0.2\pi$

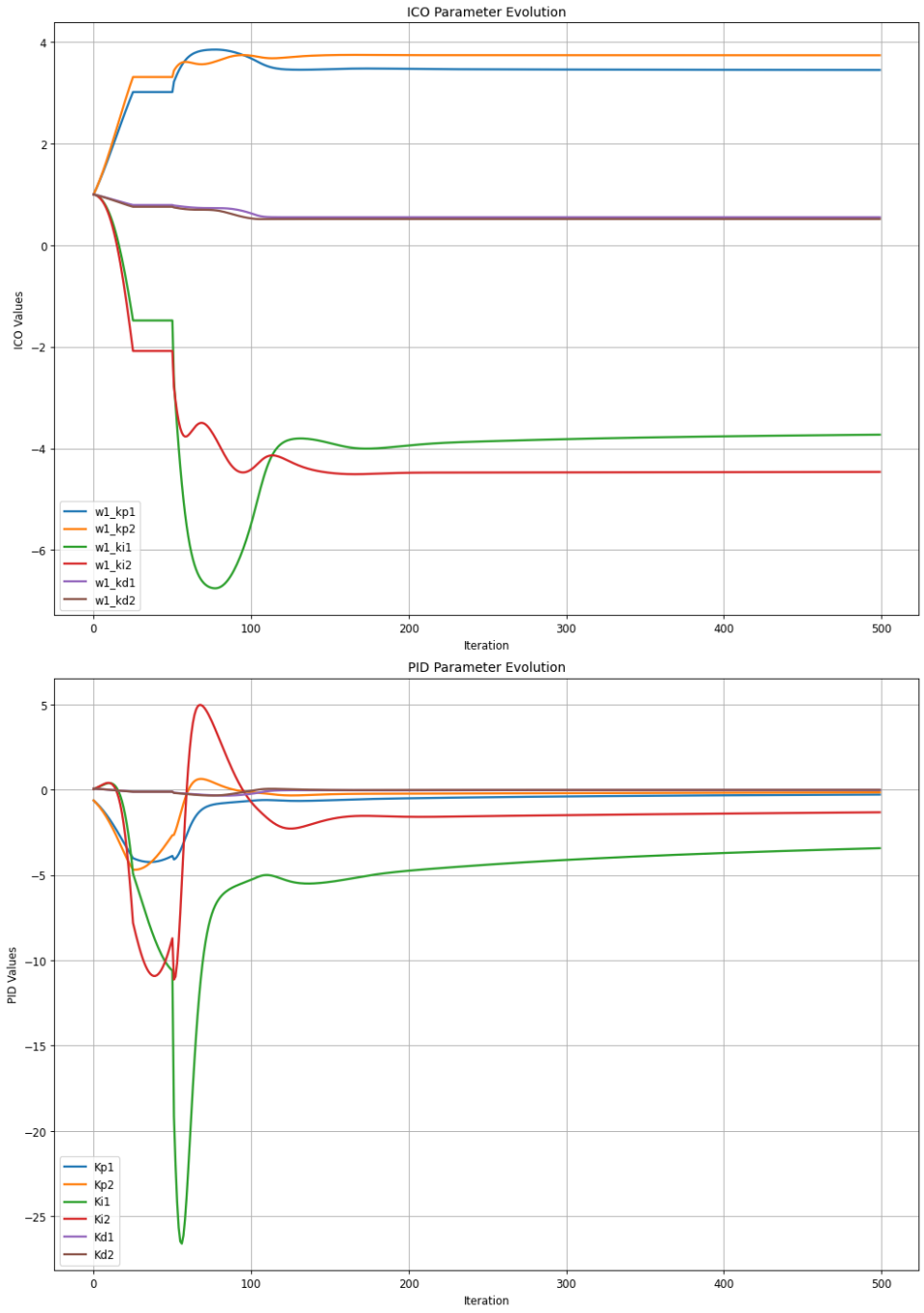


Fig. 53: Evolution of ICO and PID parameters for test scenario S8.1, using $R = 0.2\pi$.

S8.2 $R = 0.5\pi$

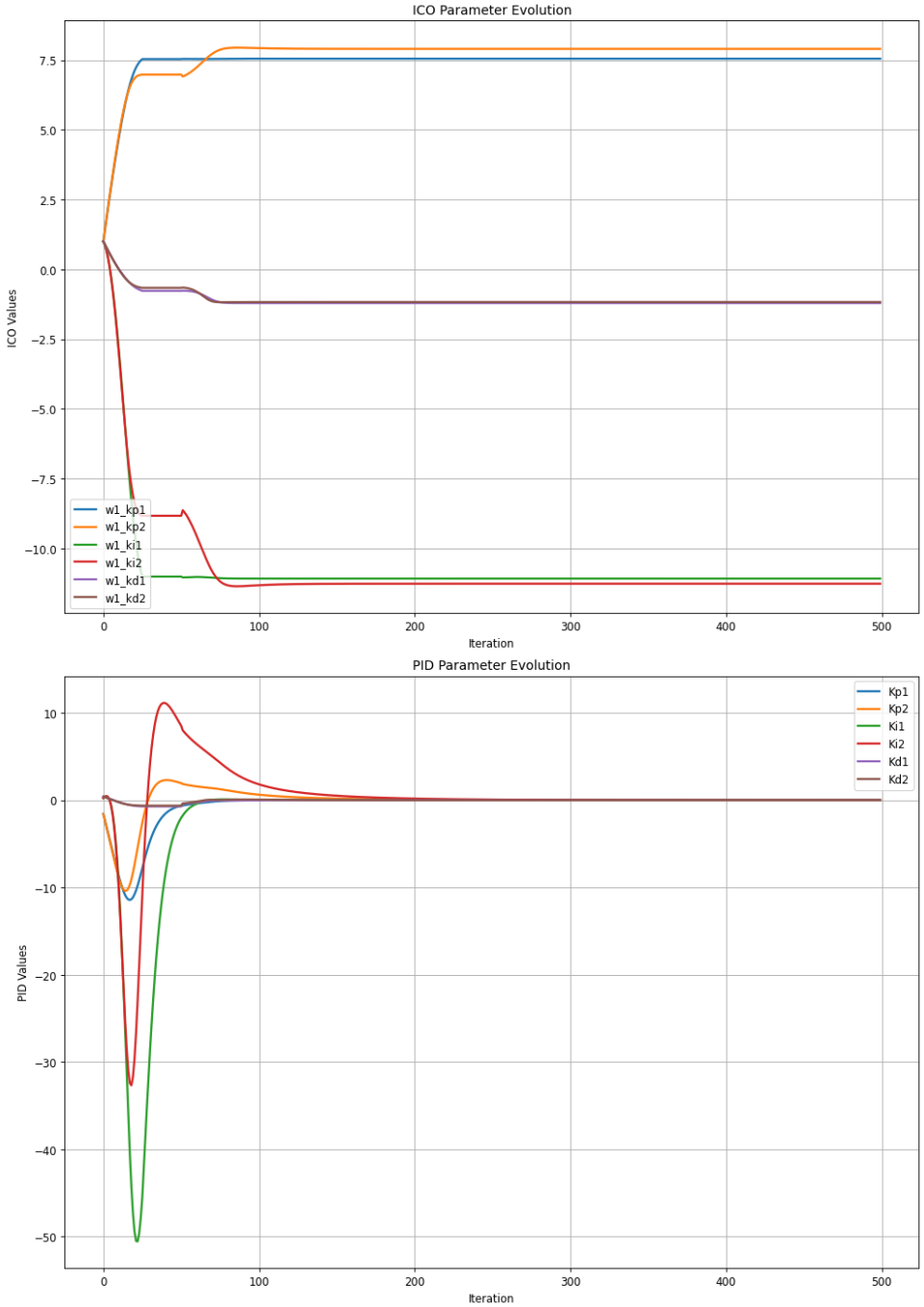


Fig. 54: Evolution of ICO and PID parameters for test scenario S8.2, using $R = 0.5\pi$.

S8.3 $R = \pi$

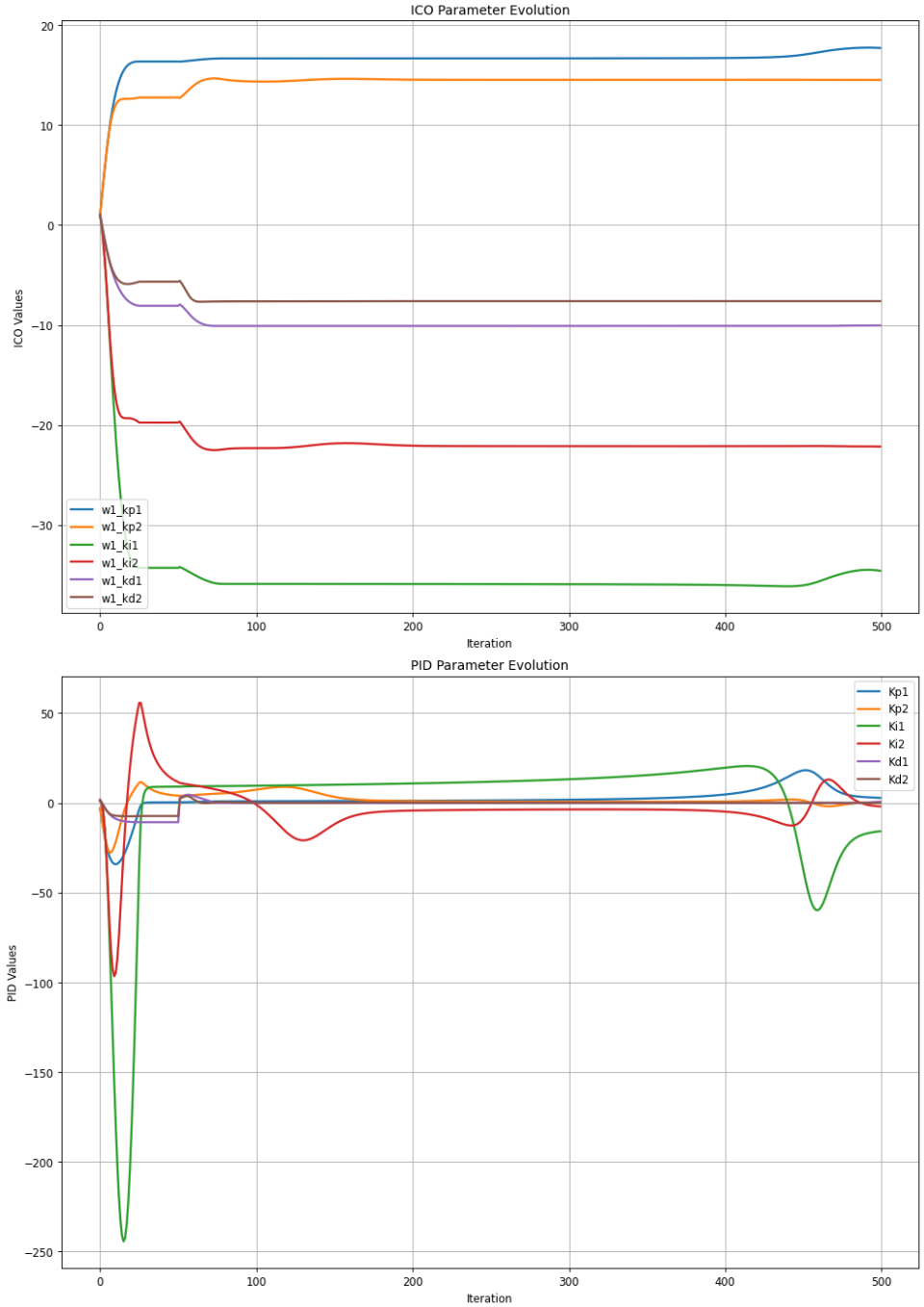


Fig. 55: Evolution of ICO and PID parameters for test scenario S8.3, using $R = \pi$.

ICO derivative filtering

S9.1 $R = 0.2\pi$

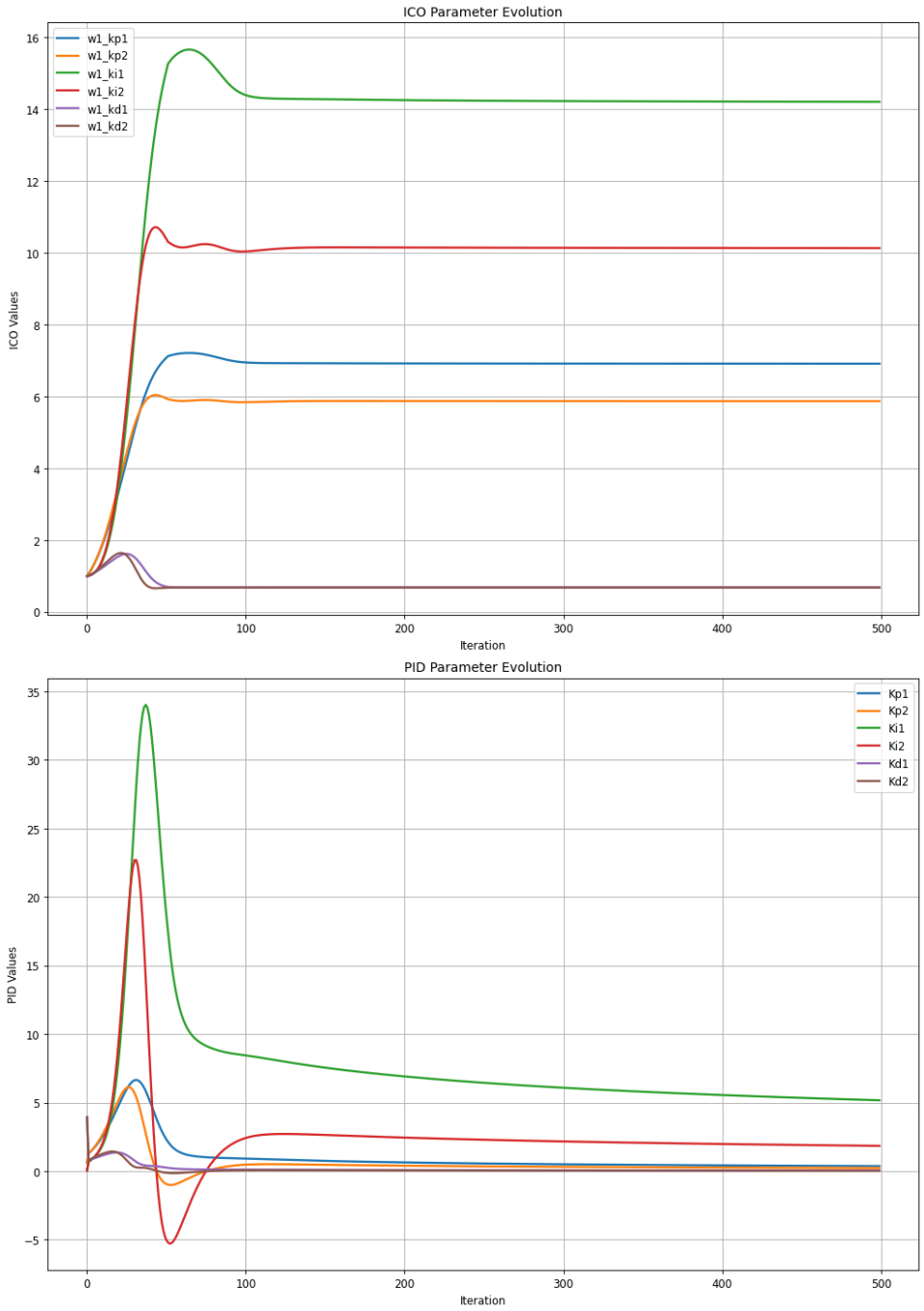


Fig. 56: Evolution of ICO and PID parameters for test scenario S9.1, using $R = 0.2\pi$.

S9.2 $R = 0.5\pi$

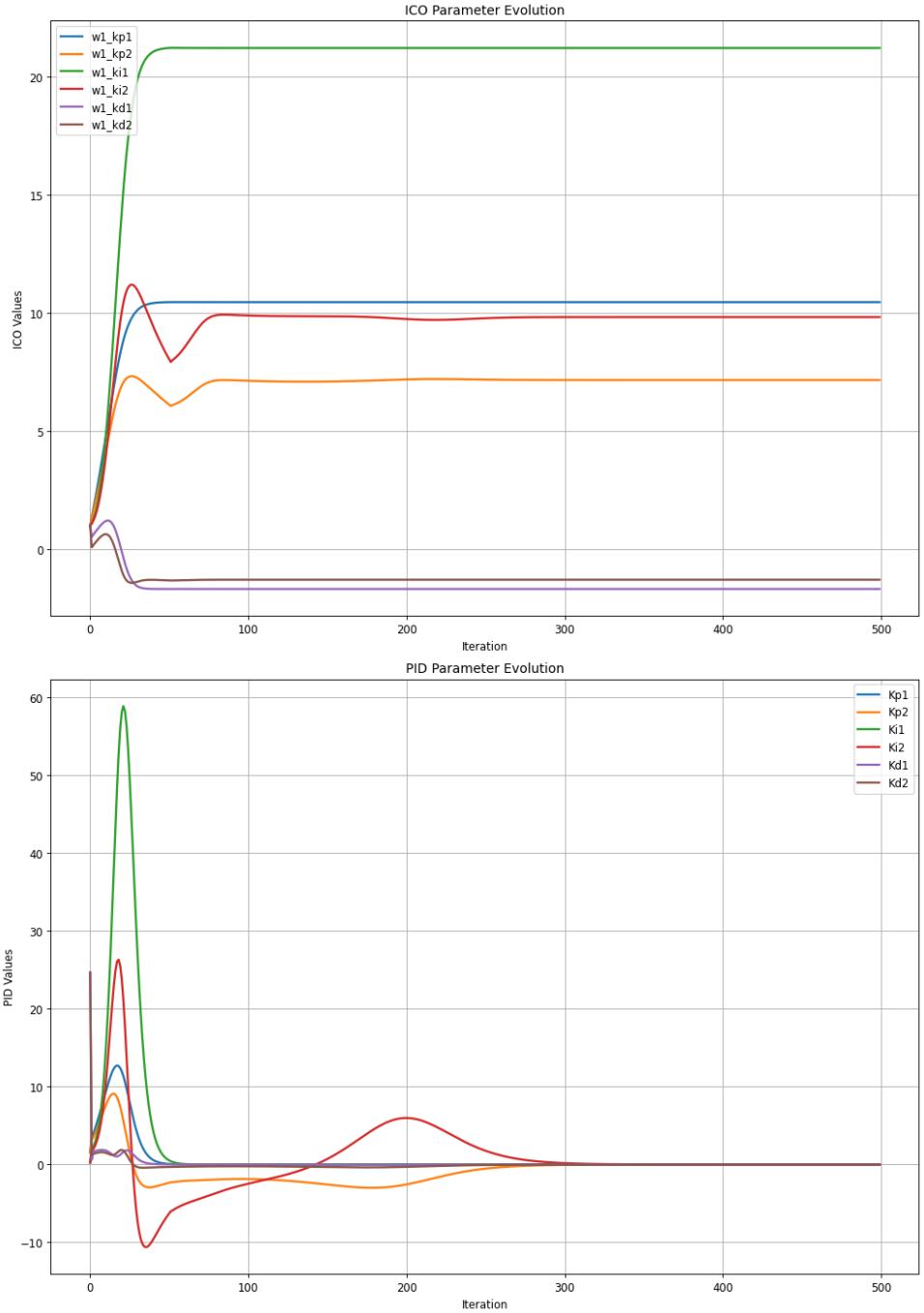


Fig. 57: Evolution of ICO and PID parameters for test scenario S9.2, using $R = 0.5\pi$.

S9.3 $R = \pi$

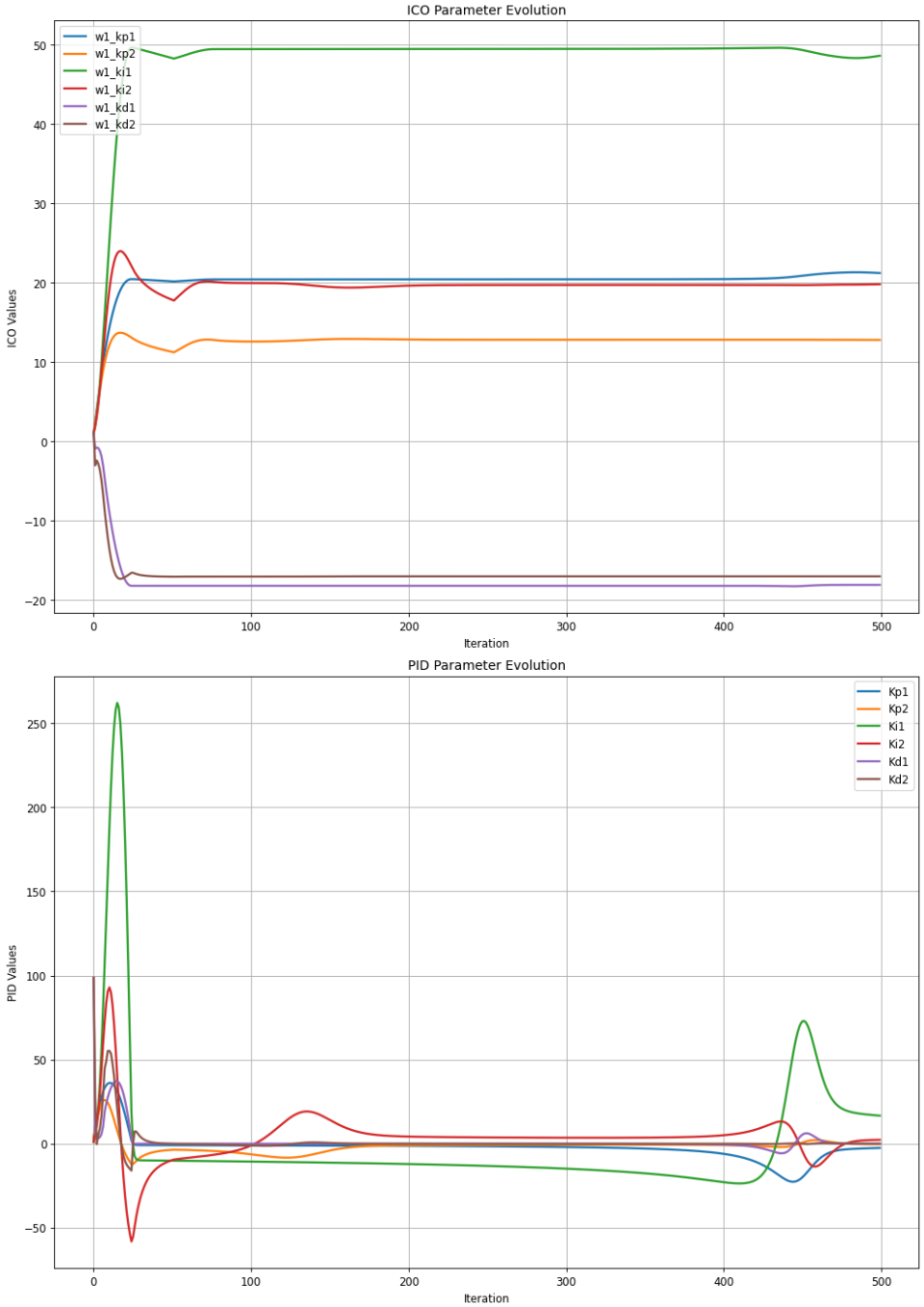


Fig. 58: Evolution of ICO and PID parameters for test scenario S9.3, using $R = \pi$.

7.5.7 Differing learning rates

This section of scenarios explores how the filtered systems behave at different tuned learning rates (μ) with a set disturbance, while the initial position is also the target position and using a sampling rate of 0,1s.

K_d model filtering

S10.1 $\mu = 0.1$**.**

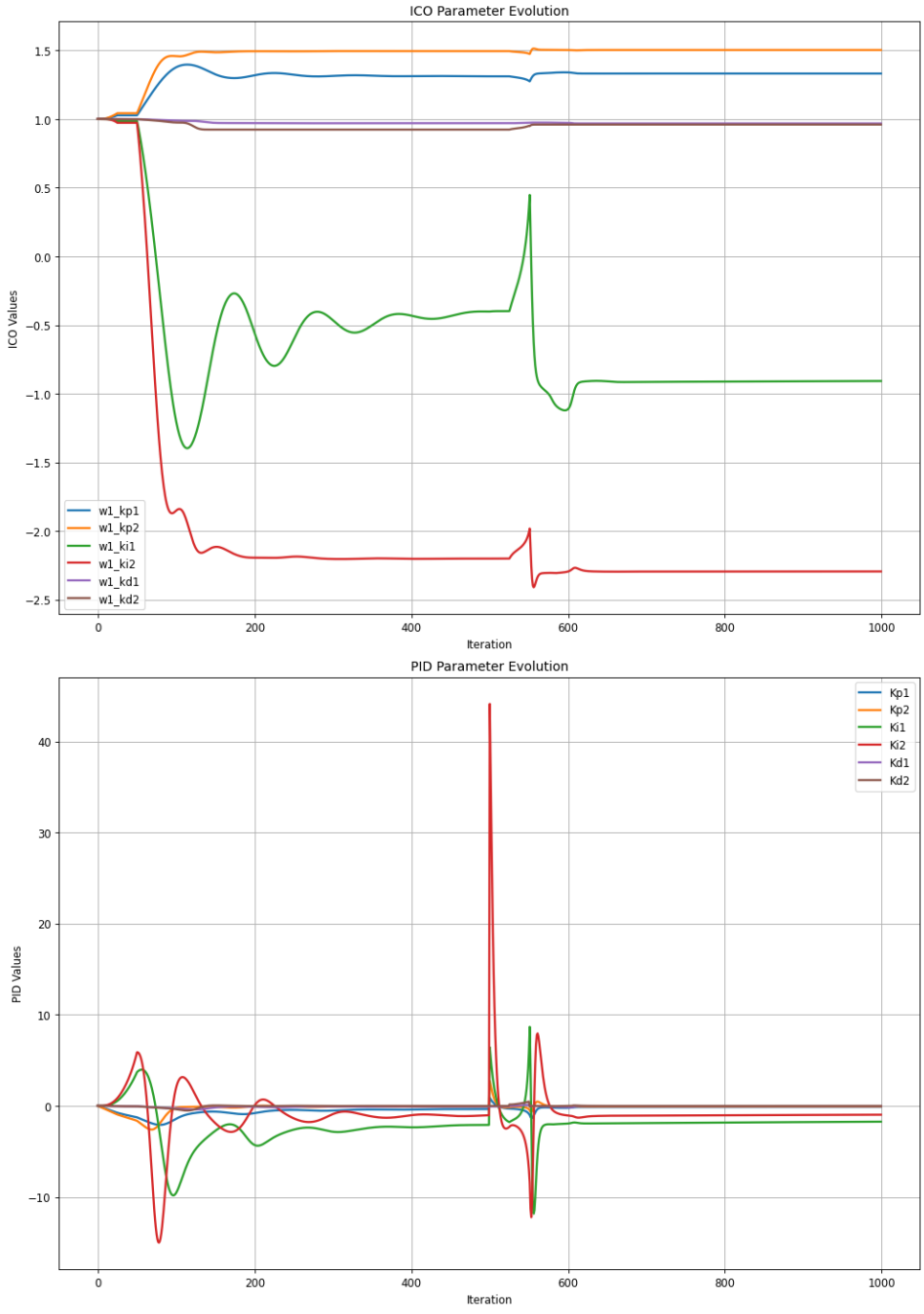


Fig. 59: Evolution of ICO and PID parameters for test scenario S10.1 with imposed disturbance and using $\mu = 0.1$.

S10.2 $\mu = 1$**.**

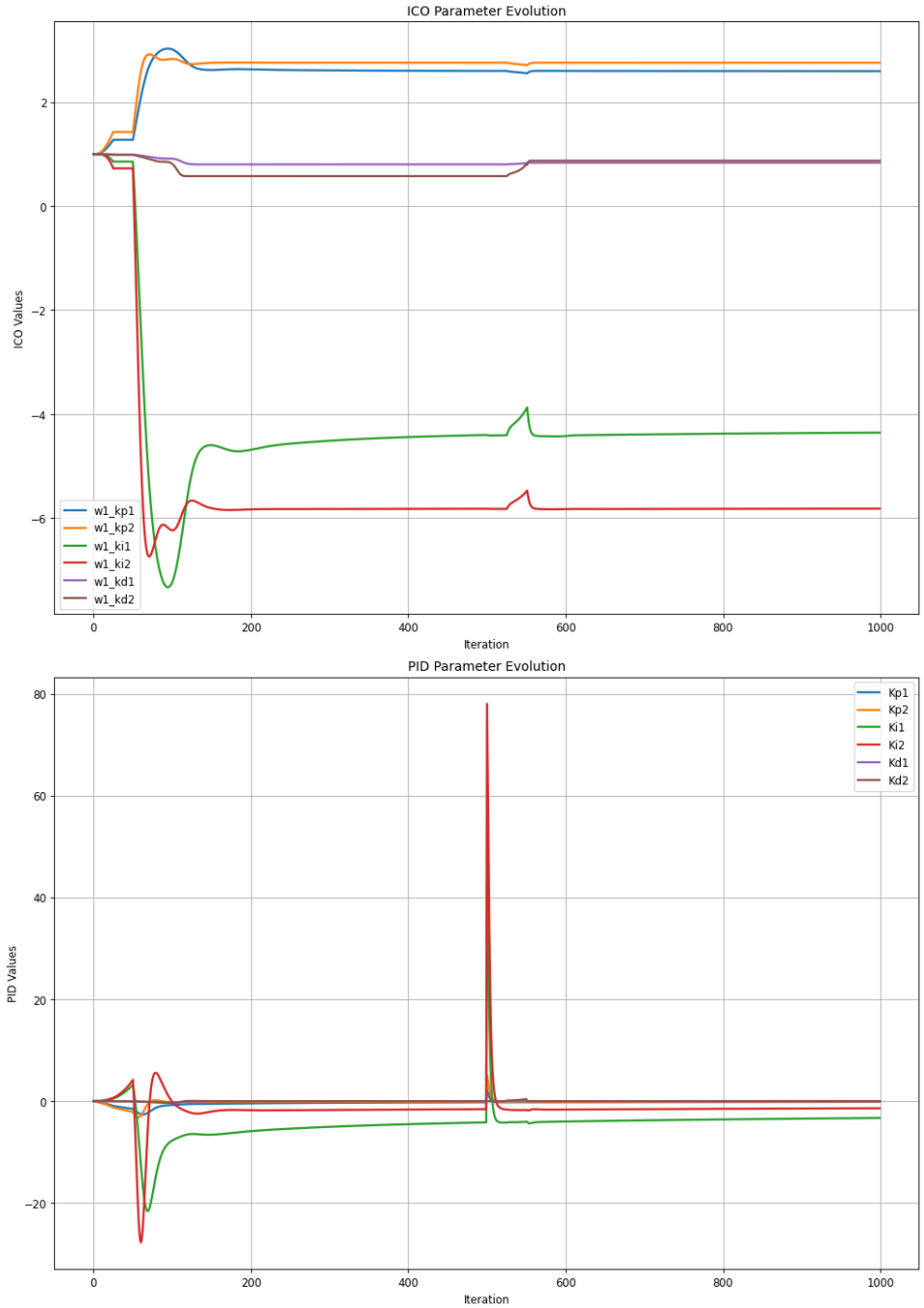


Fig. 60: Evolution of ICO and PID parameters for test scenario S10.2 with imposed disturbance and using $\mu = 1$.

S10.3 $\mu = 3$**.**

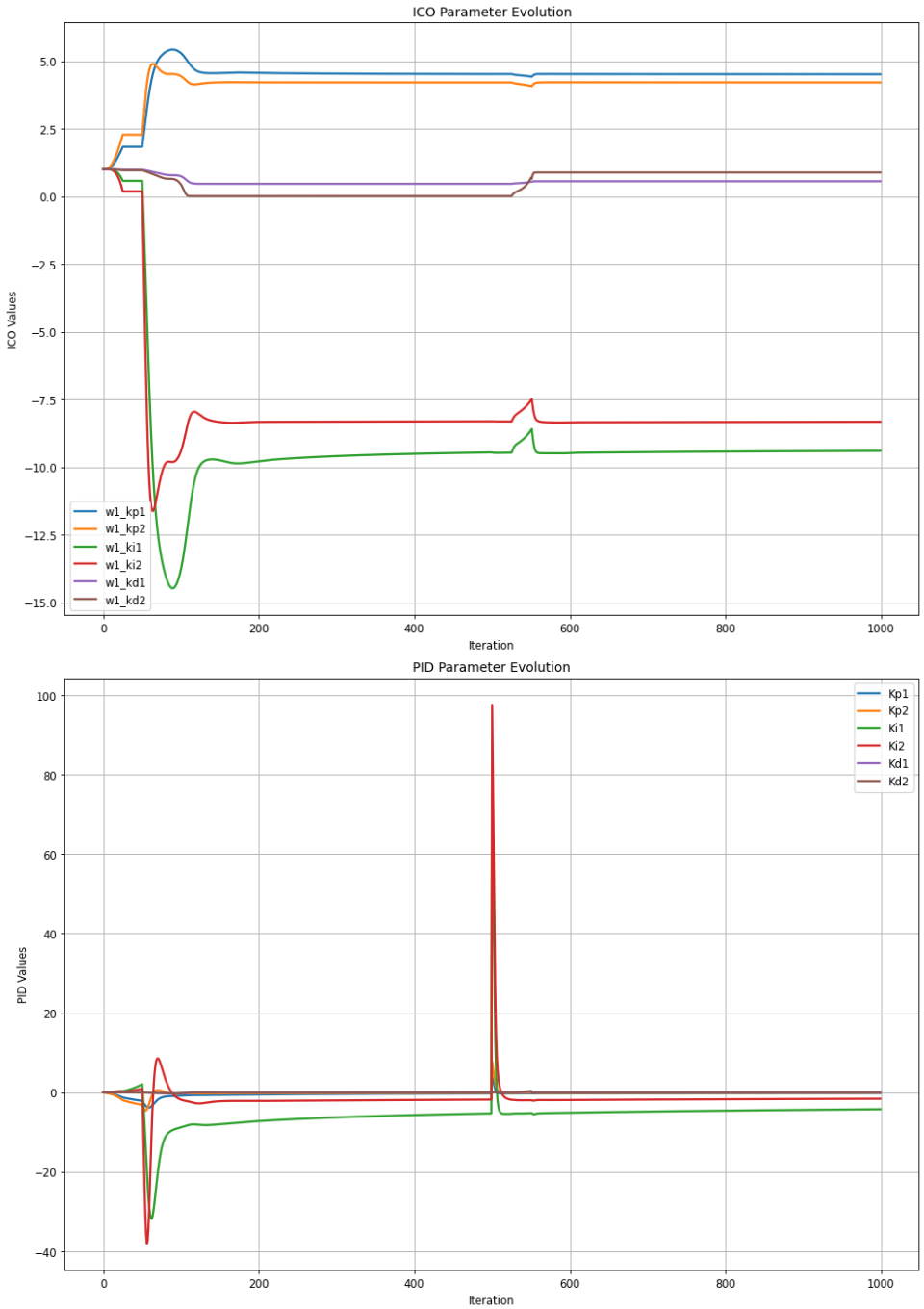


Fig. 61: Evolution of ICO and PID parameters for test scenario S10.3 with imposed disturbance and using $\mu = 3$.

ICO derivative filtering

S11.1 $\mu = 0.1$**.**

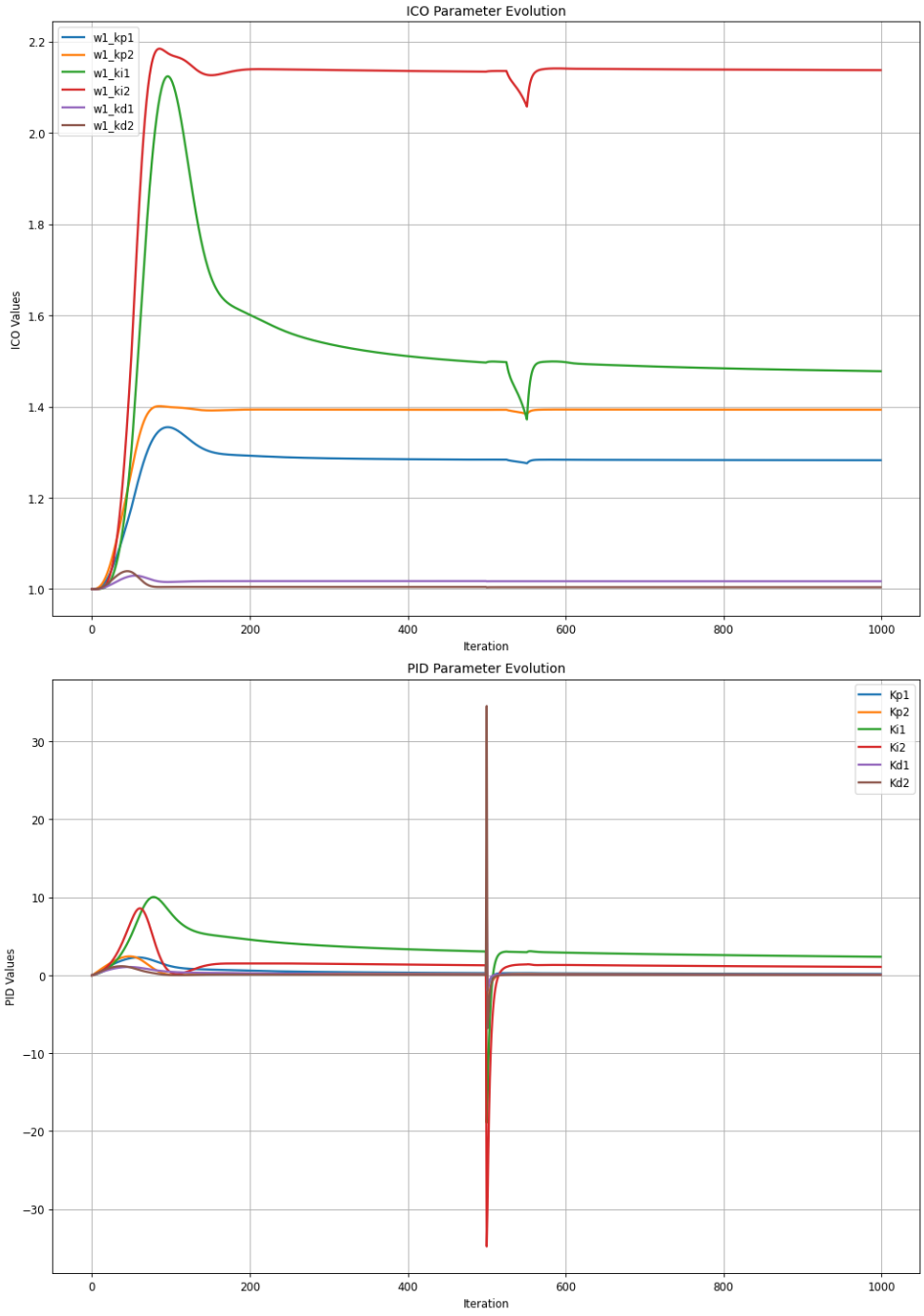


Fig. 62: Evolution of ICO and PID parameters for test scenario S11.1 with imposed disturbance and using $\mu = 0.1$.

S11.2 $\mu = 1$**.**

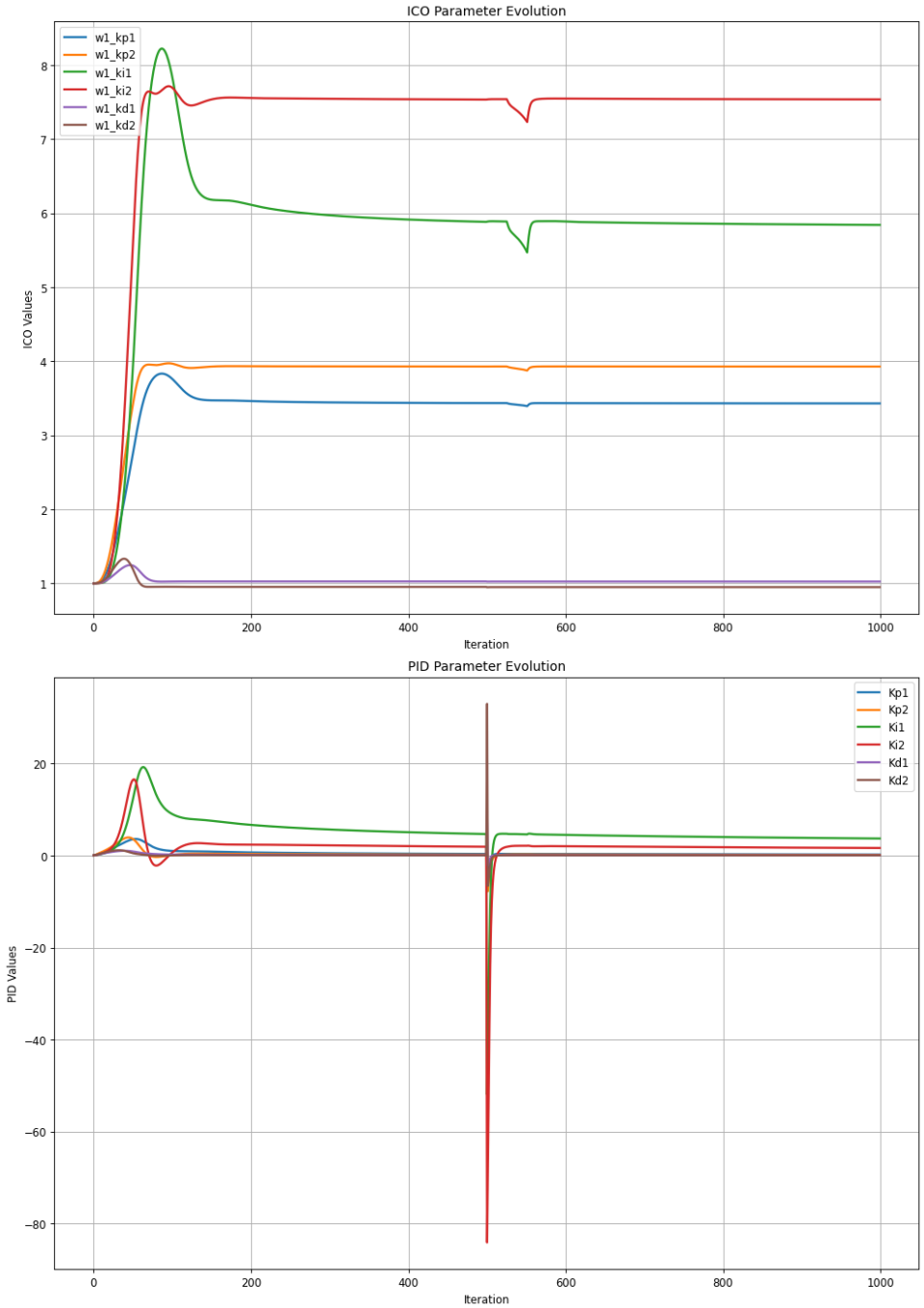


Fig. 63: Evolution of ICO and PID parameters for test scenario S11.2 with imposed disturbance and using $\mu = 1$.

S11.3 $\mu = 3$**.**

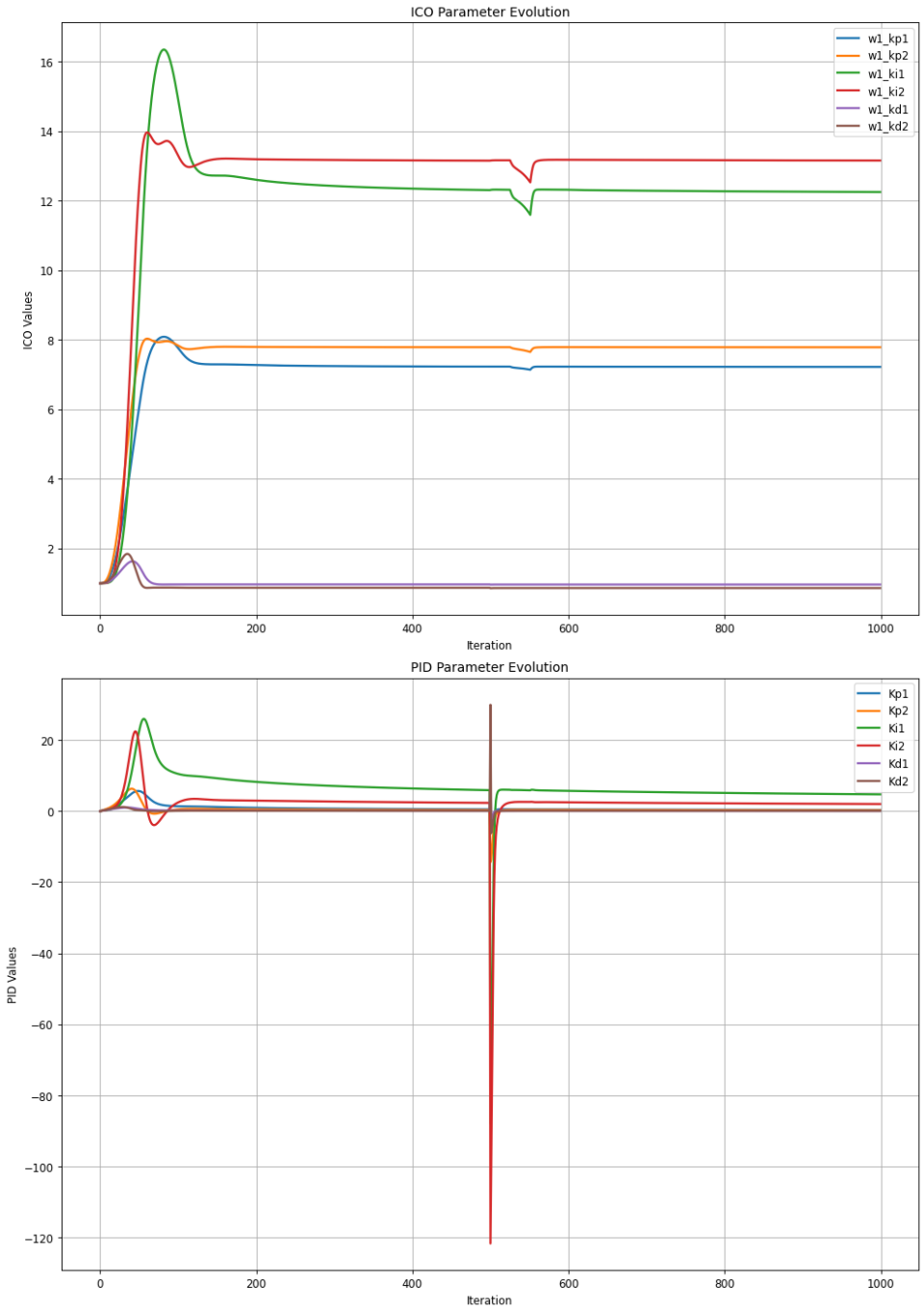


Fig. 64: Evolution of ICO and PID parameters for test scenario S11.3 with imposed disturbance and using $\mu = 3$.

7.5.8 Differing sample rates

This section of scenarios explores how the filtered systems behave at different tuned sampling rates (dt) and with imposed disturbance, while the initial position is also the target position and using a learning rate of 1.

K_d model filtering

S12.1 $dt = 0.05\ s$

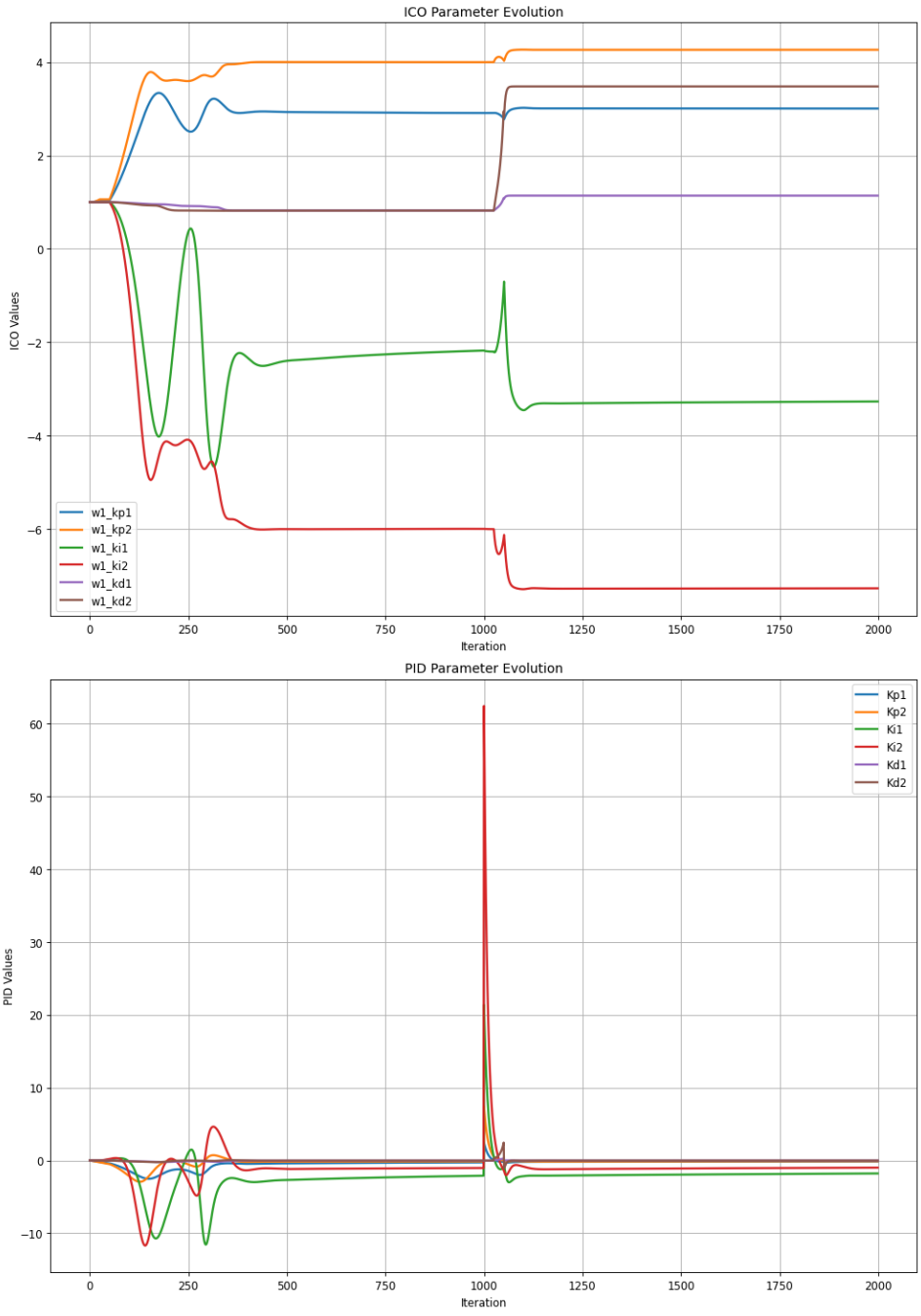


Fig. 65: Evolution of ICO and PID parameters for test scenario S12.1 with imposed disturbance and using $dt = 0.05$.

S12.2 $dt = 0.08\ s$

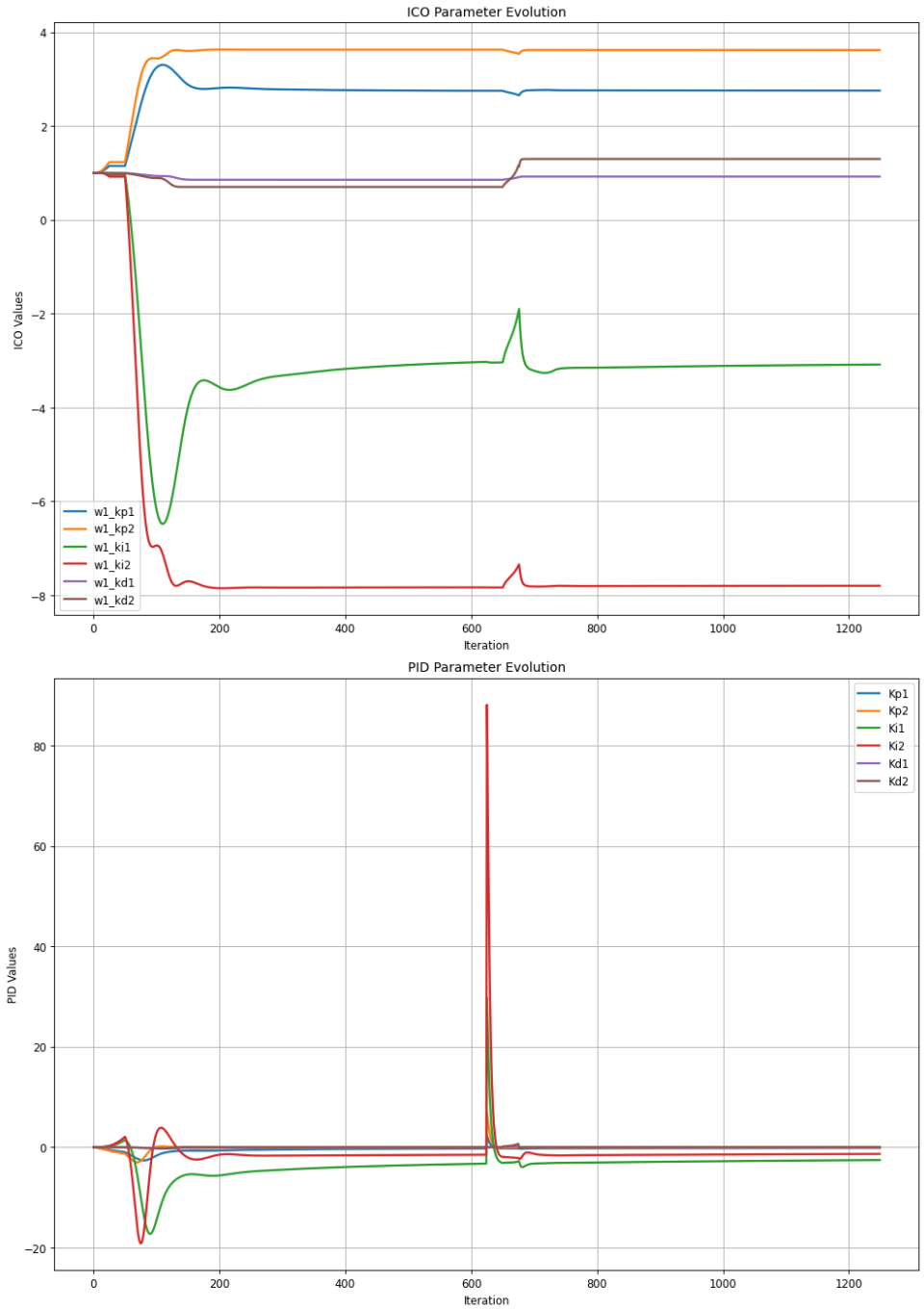


Fig. 66: Evolution of ICO and PID parameters for test scenario S12.2 with imposed disturbance and using $dt = 0.08$.

S12.3 $dt = 0.15\ s$

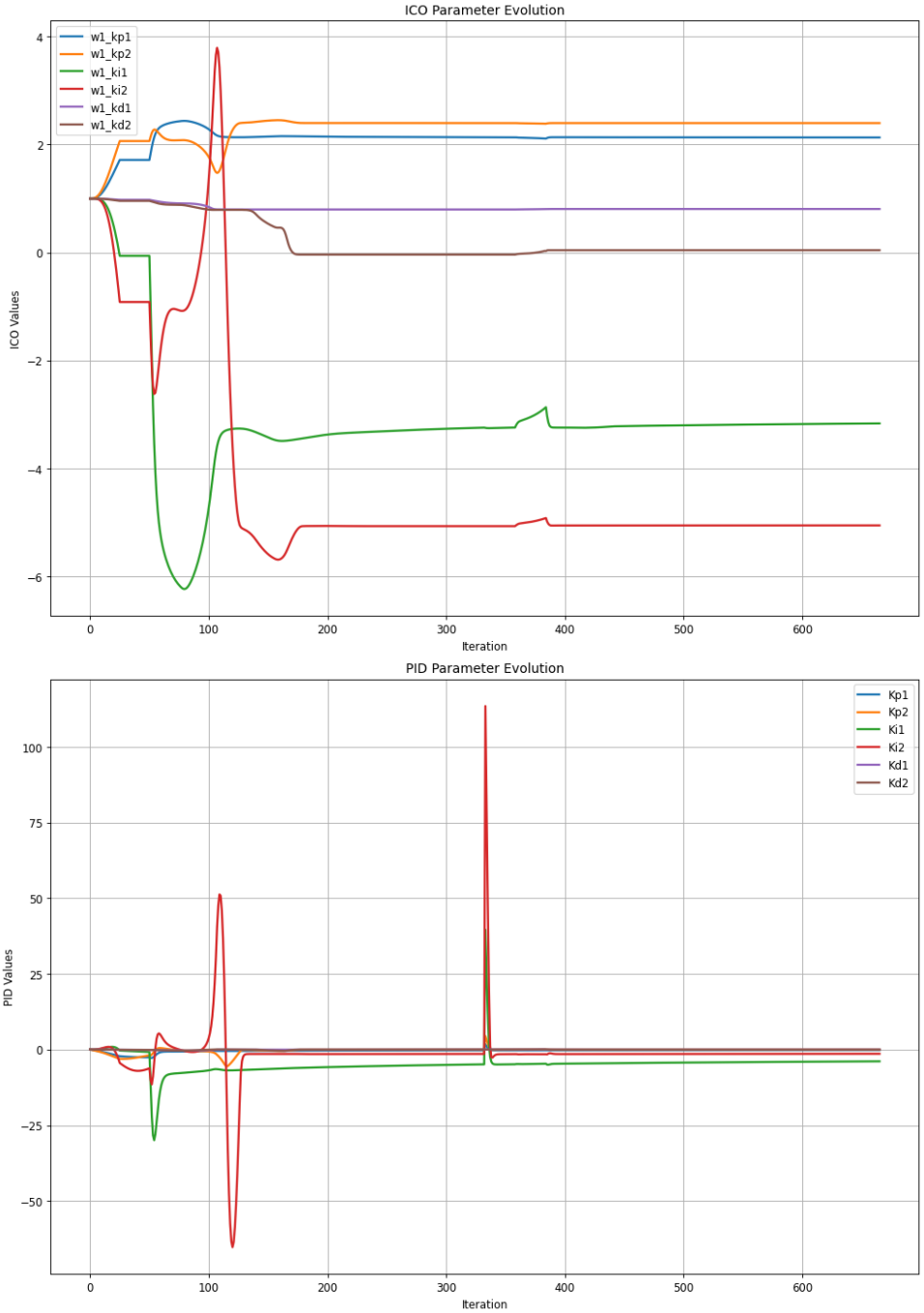


Fig. 67: Evolution of ICO and PID parameters for test scenario S12.3 with imposed disturbance and using $dt = 0.15$.

ICO derivative filtering

S13.1 $dt = 0.05\ s$

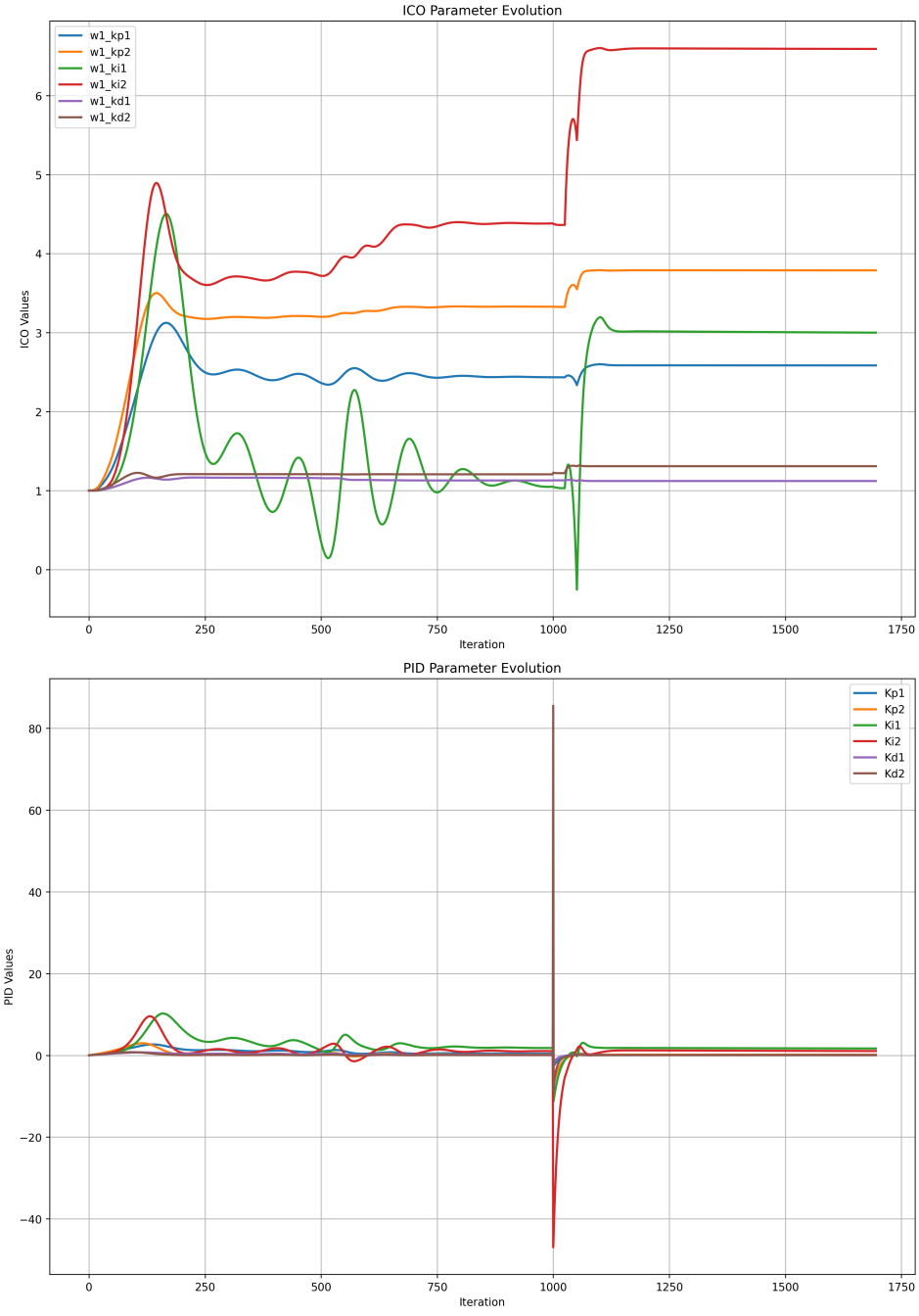


Fig. 68: Evolution of ICO and PID parameters for test scenario S13.1 with imposed disturbance and using $dt = 0.05$.

S13.2 $dt = 0.08\ s$

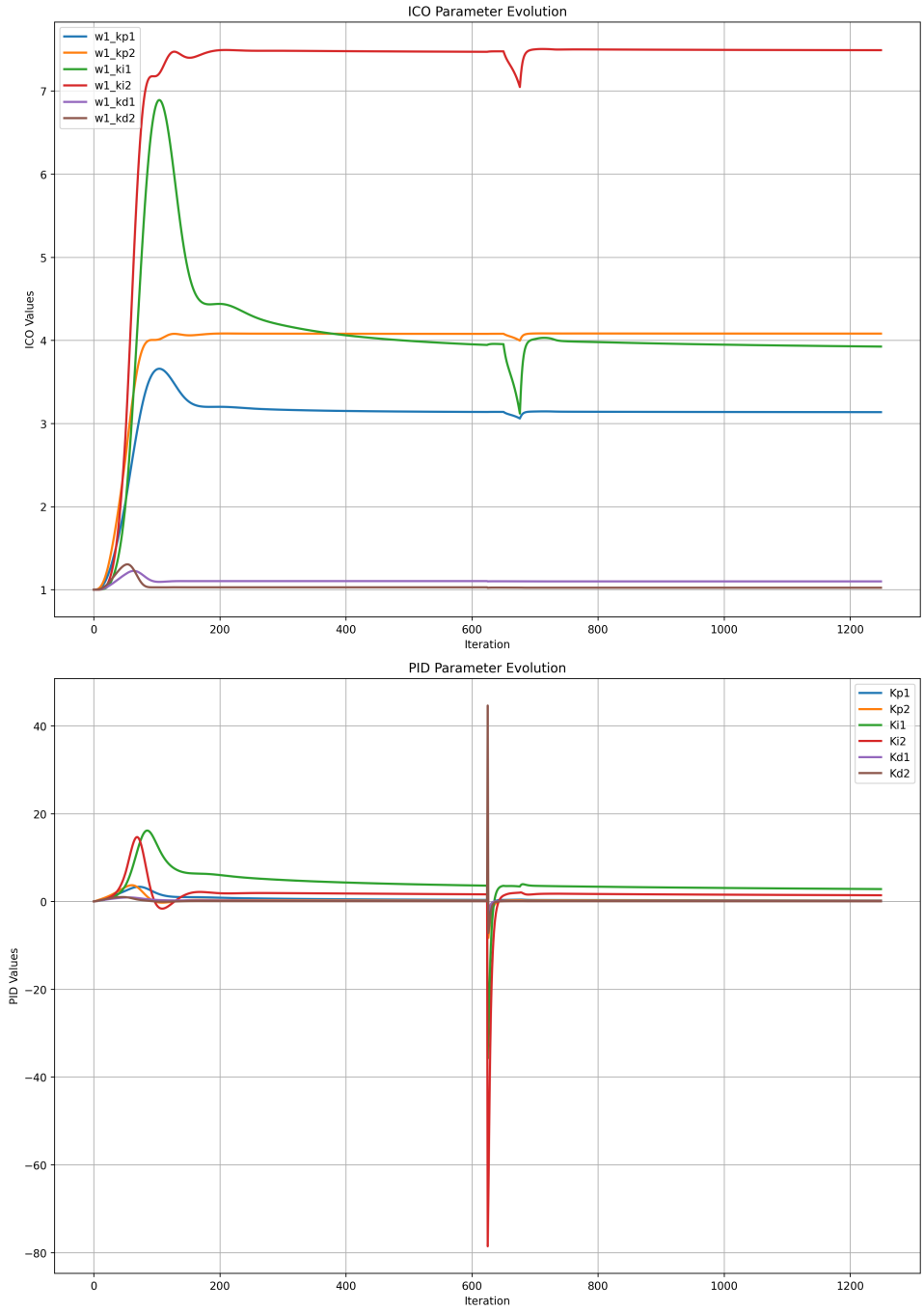


Fig. 69: Evolution of ICO and PID parameters for test scenario S13.2 with imposed disturbance and using $dt = 0.08$.

S13.3 $dt = 0.15\ s$

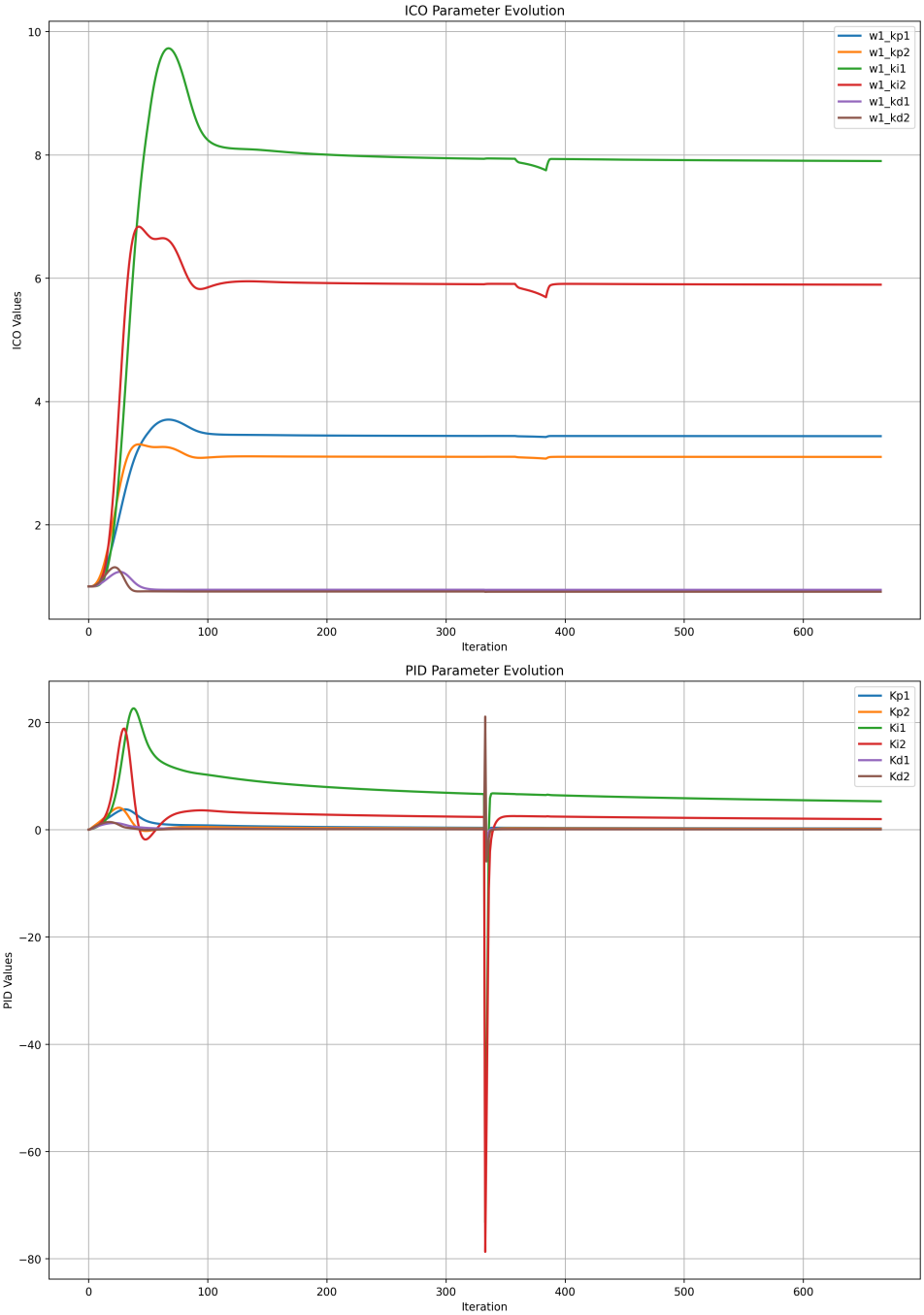


Fig. 70: Evolution of ICO and PID parameters for test scenario S13.3 with imposed disturbance and using $dt = 0.15$.

7.5.9 Differing initial error

This section of scenarios explores how the filtered systems behave at different setpoints (R) / initial errors ($E(0)$) with imposed disturbance, and with the learning rate of 1 and a sampling rate of 0,1 seconds.

K_d model filtering

S14.1 $R = 0.2\pi$

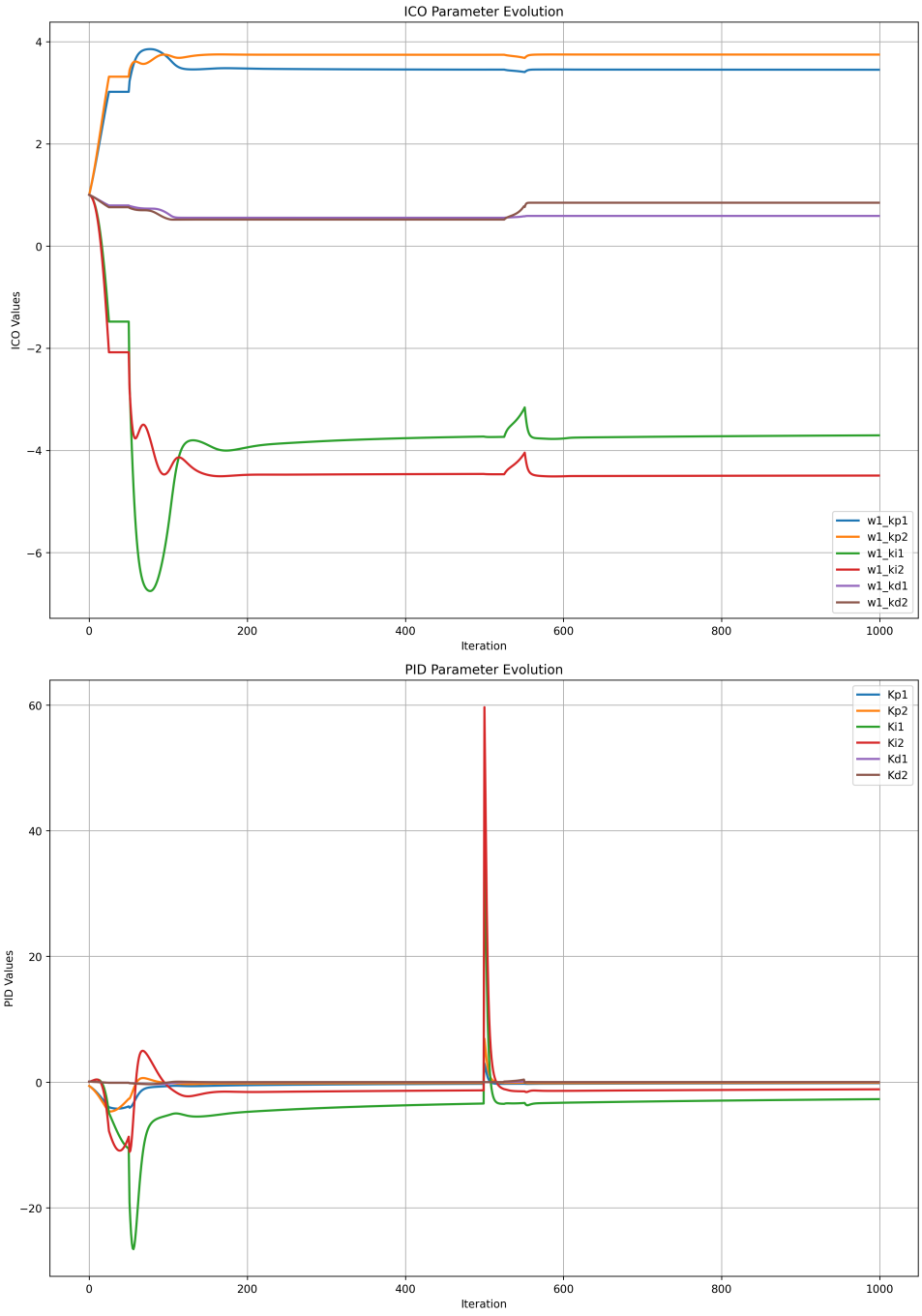


Fig. 71: Evolution of ICO and PID parameters for test scenario S14.1, using $R = 0.2\pi$.

S14.2 $R = 0.5\pi$

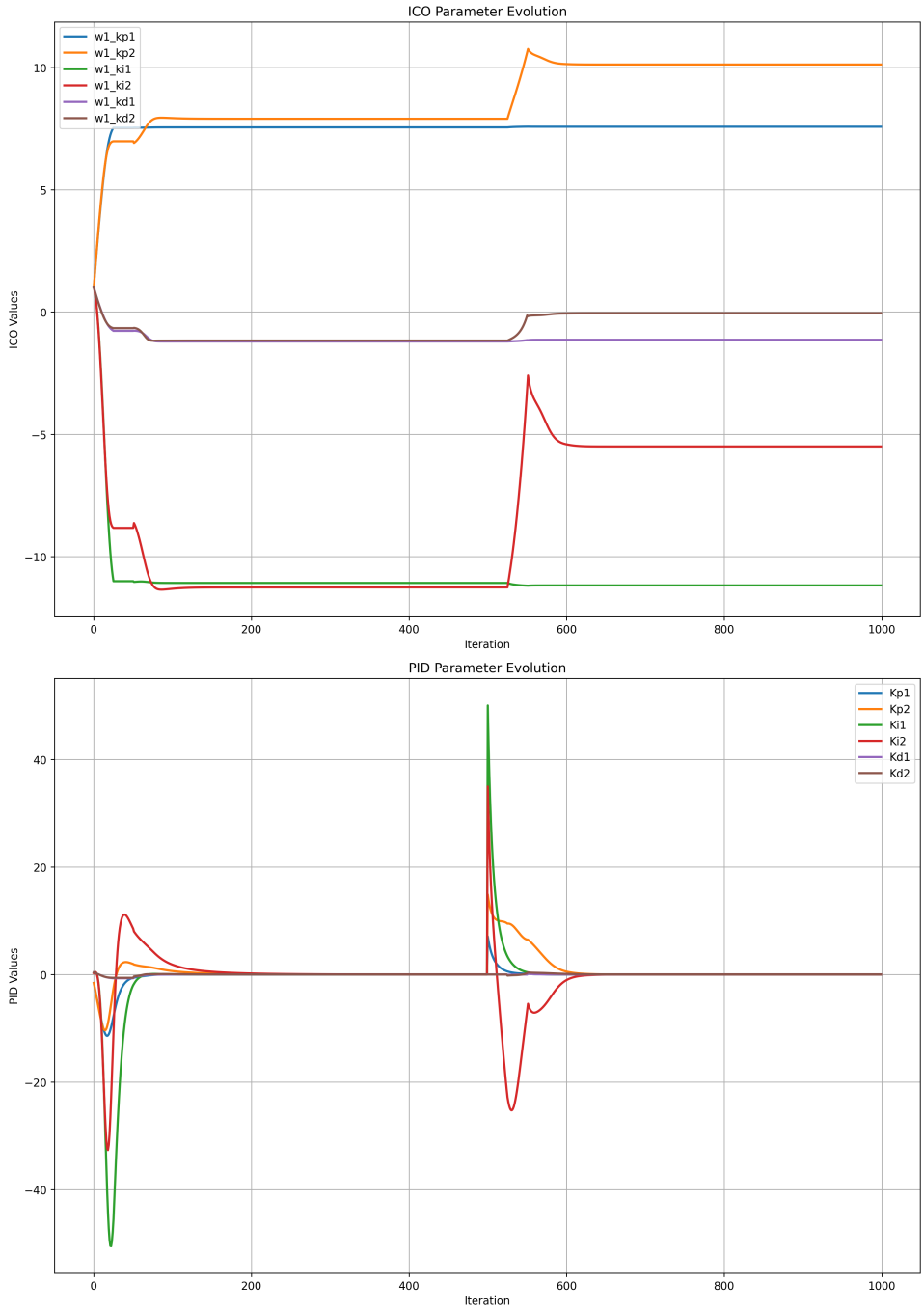


Fig. 72: Evolution of ICO and PID parameters for test scenario S14.2, using $R = 0.5\pi$.

S14.3 $R = \pi$

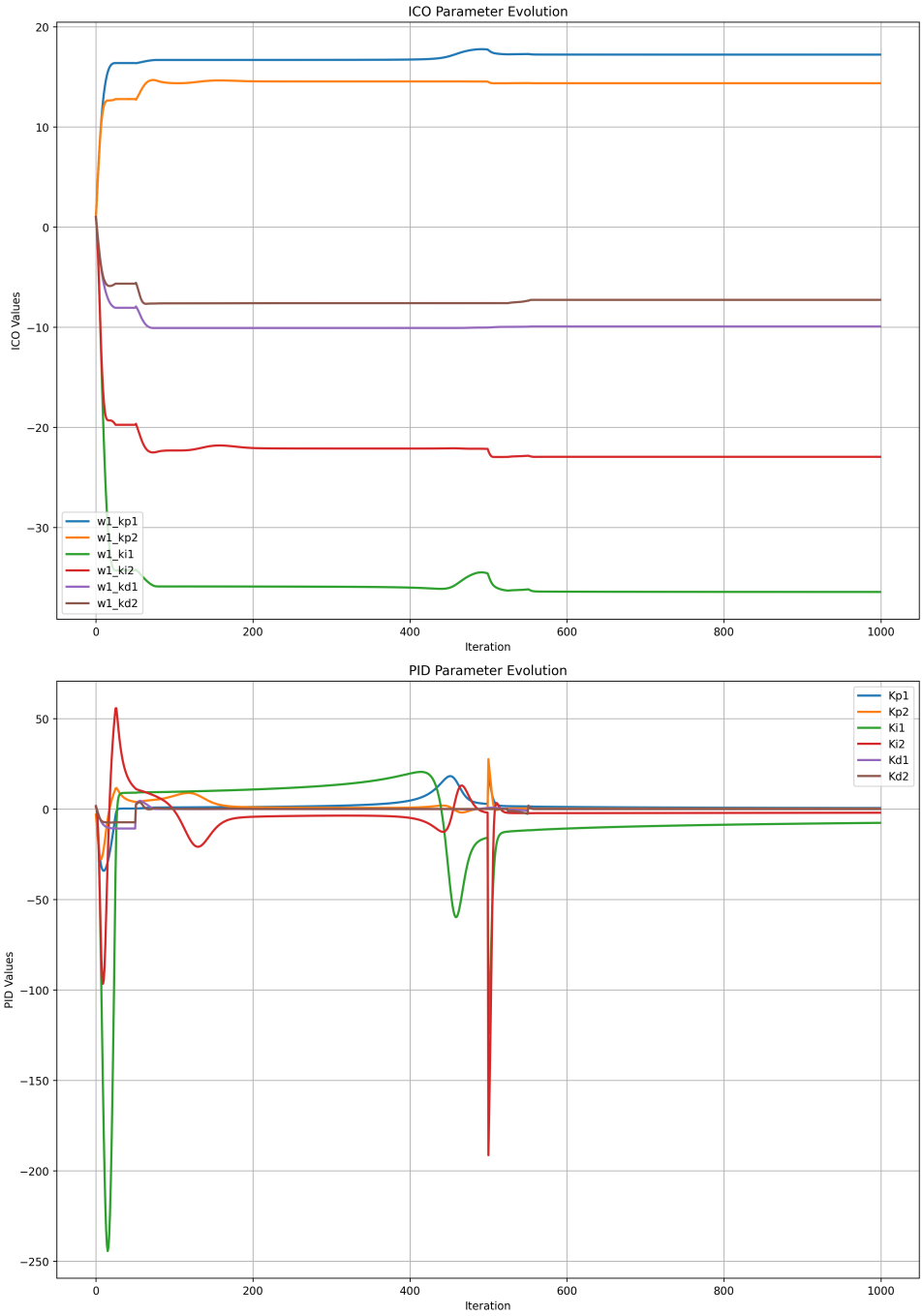


Fig. 73: Evolution of ICO and PID parameters for test scenario S14.3, using $R = \pi$.

ICO derivative filtering

S15.1 $R = 0.2\pi$

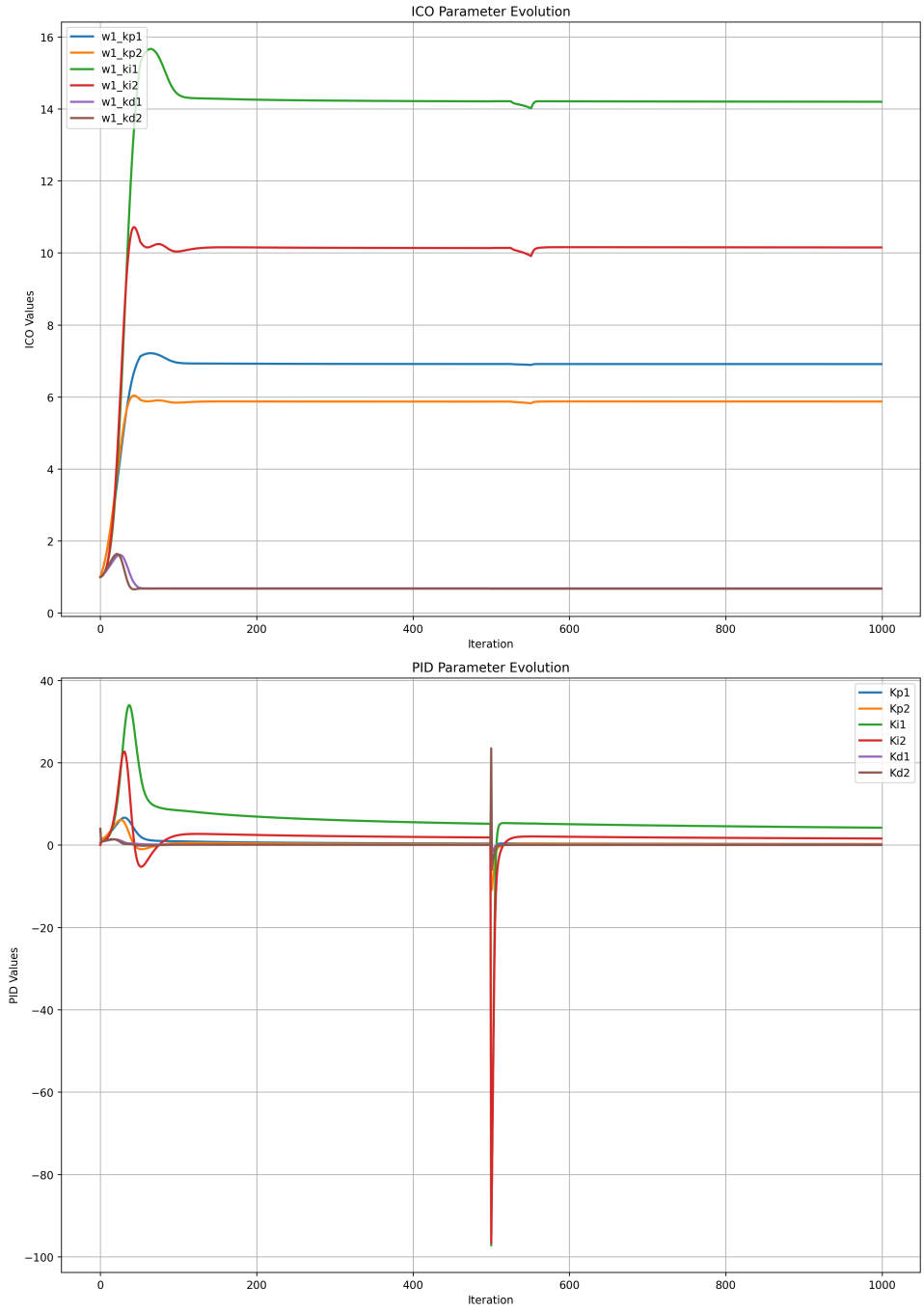


Fig. 74: Evolution of ICO and PID parameters for test scenario S15.1, using $R = 0.2\pi$.

S15.2 $R = 0.5\pi$

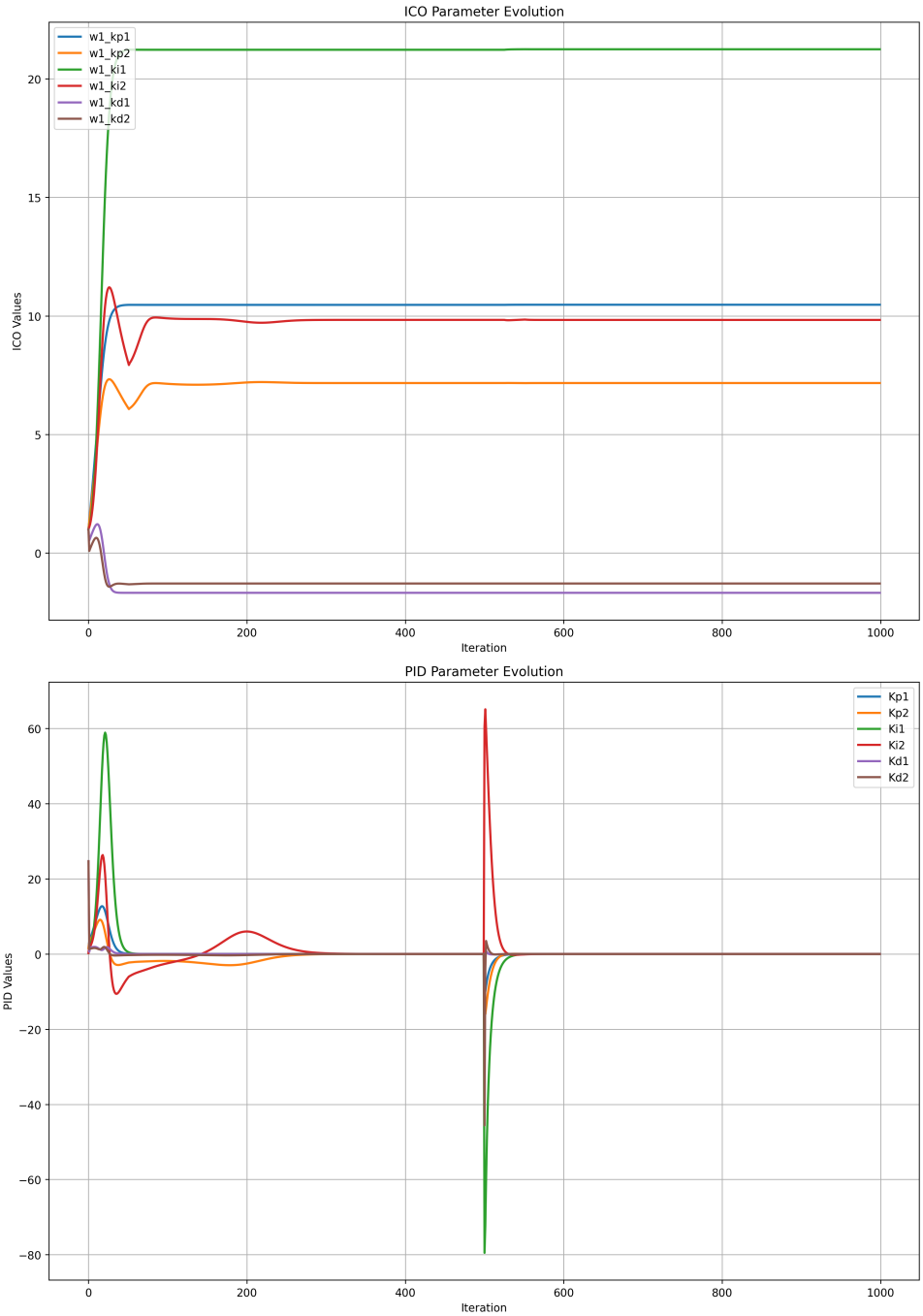


Fig. 75: Evolution of ICO and PID parameters for test scenario S15.2, using $R = 0.5\pi$.

S15.3 $R = \pi$

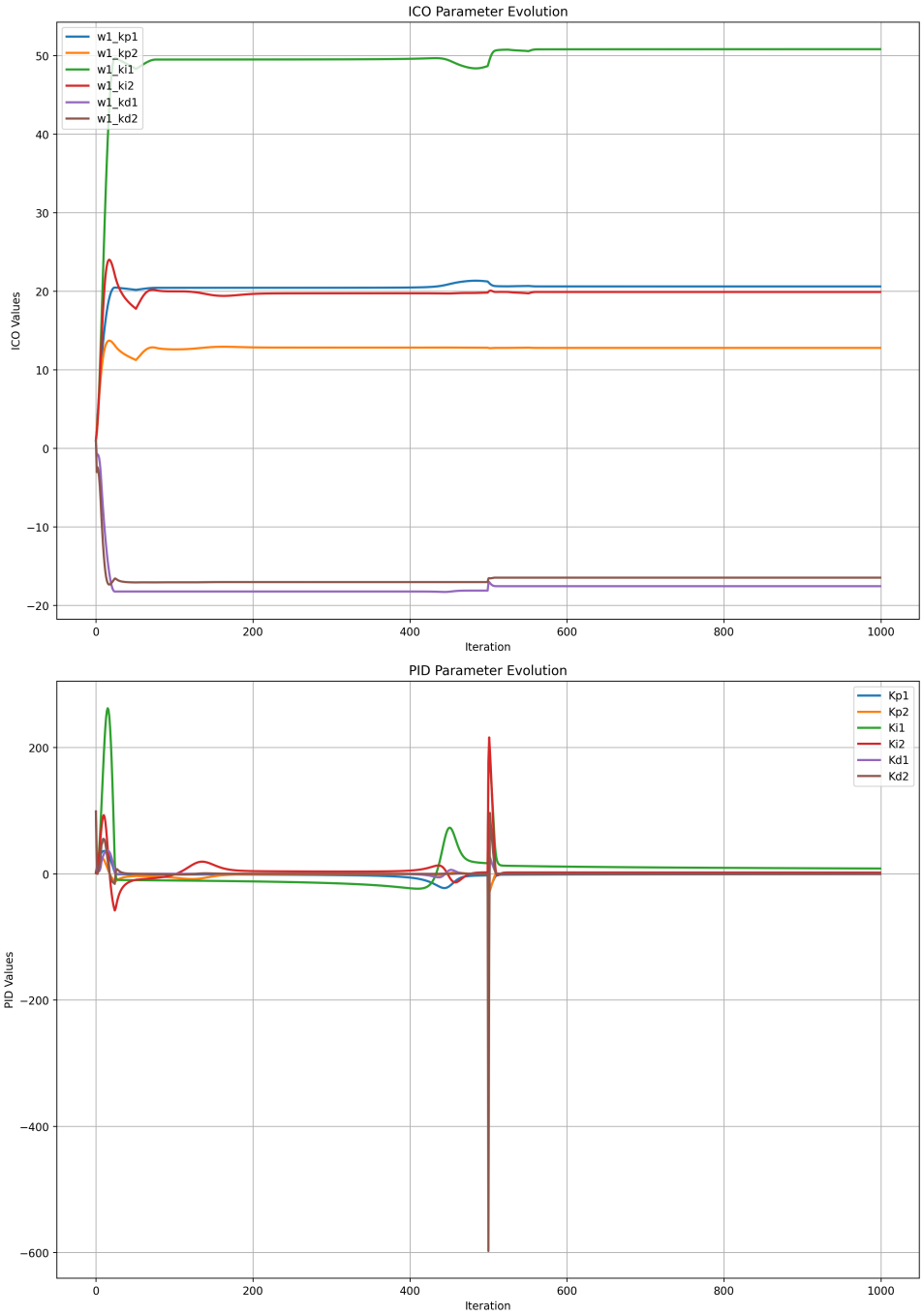


Fig. 76: Evolution of ICO and PID parameters for test scenario S15.3, using $R = \pi$.

K_d model filtering

S16.1 Normal periodic disturbances every 10 seconds.

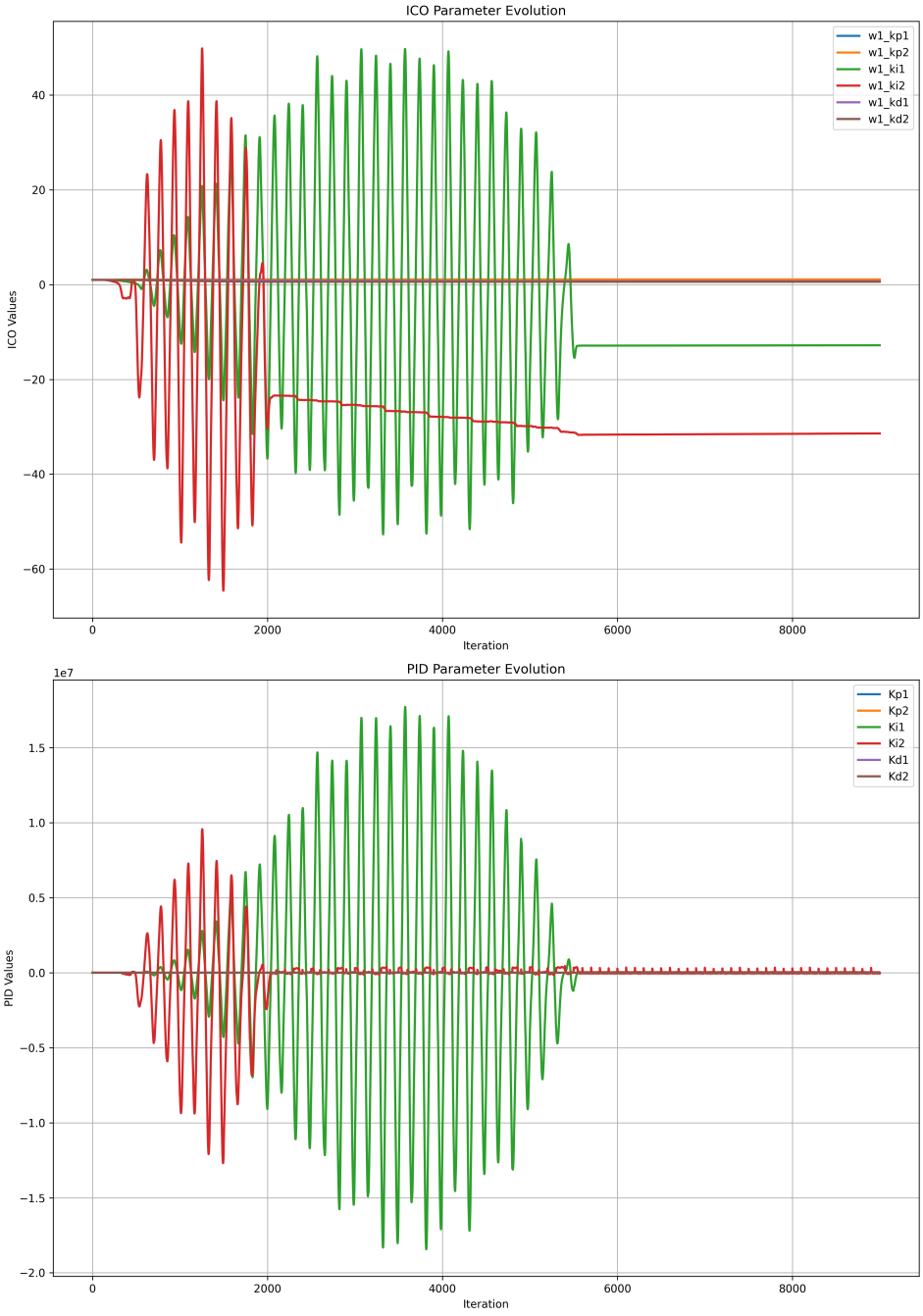


Fig. 77: Evolution of ICO and PID parameters for test scenario S16.1, where the system is subject to periodic disturbances.

S16.2 Ramp disturbances every 10 seconds

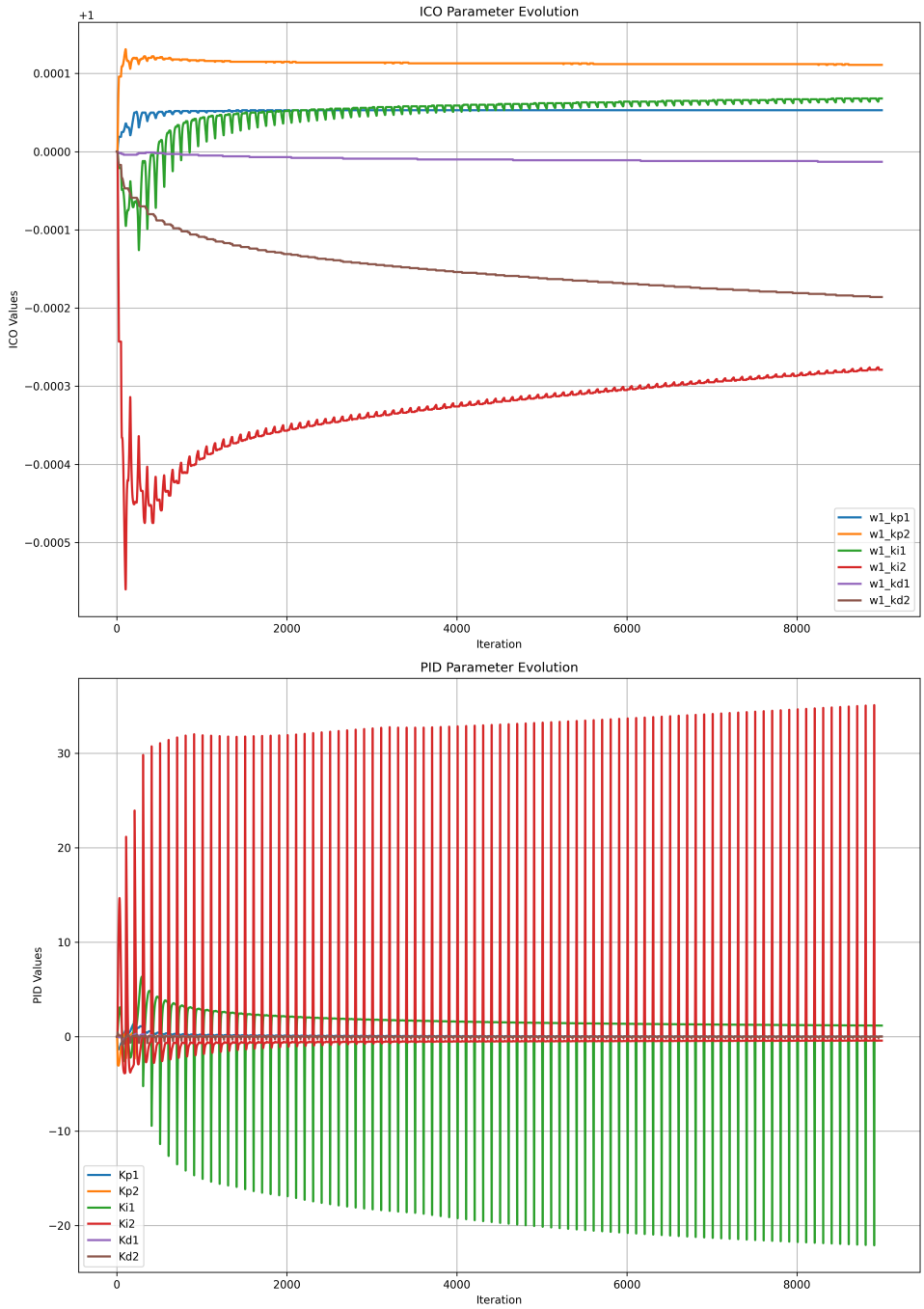


Fig. 78: Evolution of ICO and PID parameters for test scenario S16.2, where the system is subject to periodic disturbances.

ICO derivative filtering

S17.1 Normal periodic disturbances every 10 seconds.

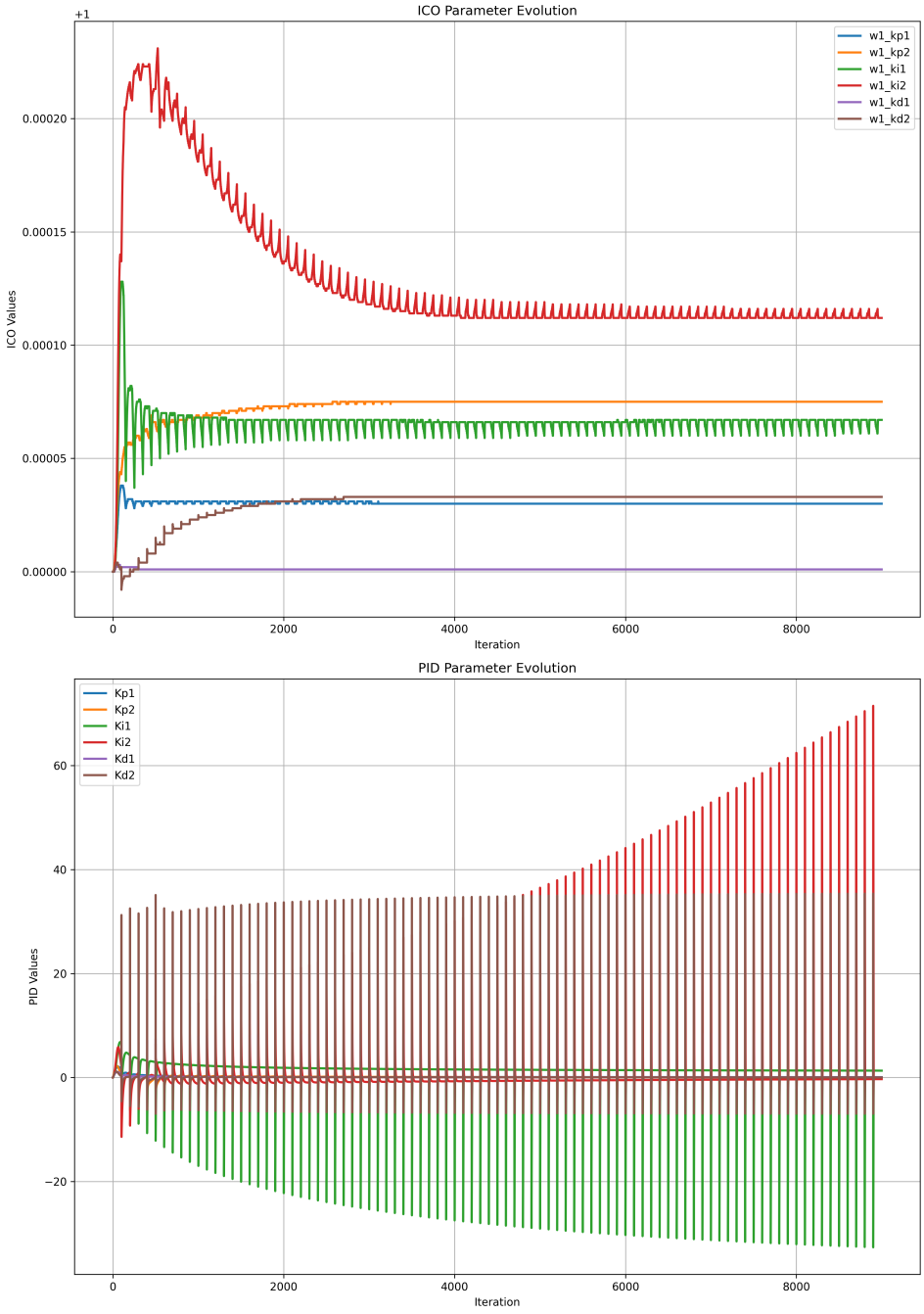


Fig. 79: Evolution of ICO and PID parameters for test scenario S17.1, where the system is subject to periodic disturbances.

S17.2 Ramp disturbances every 10 seconds

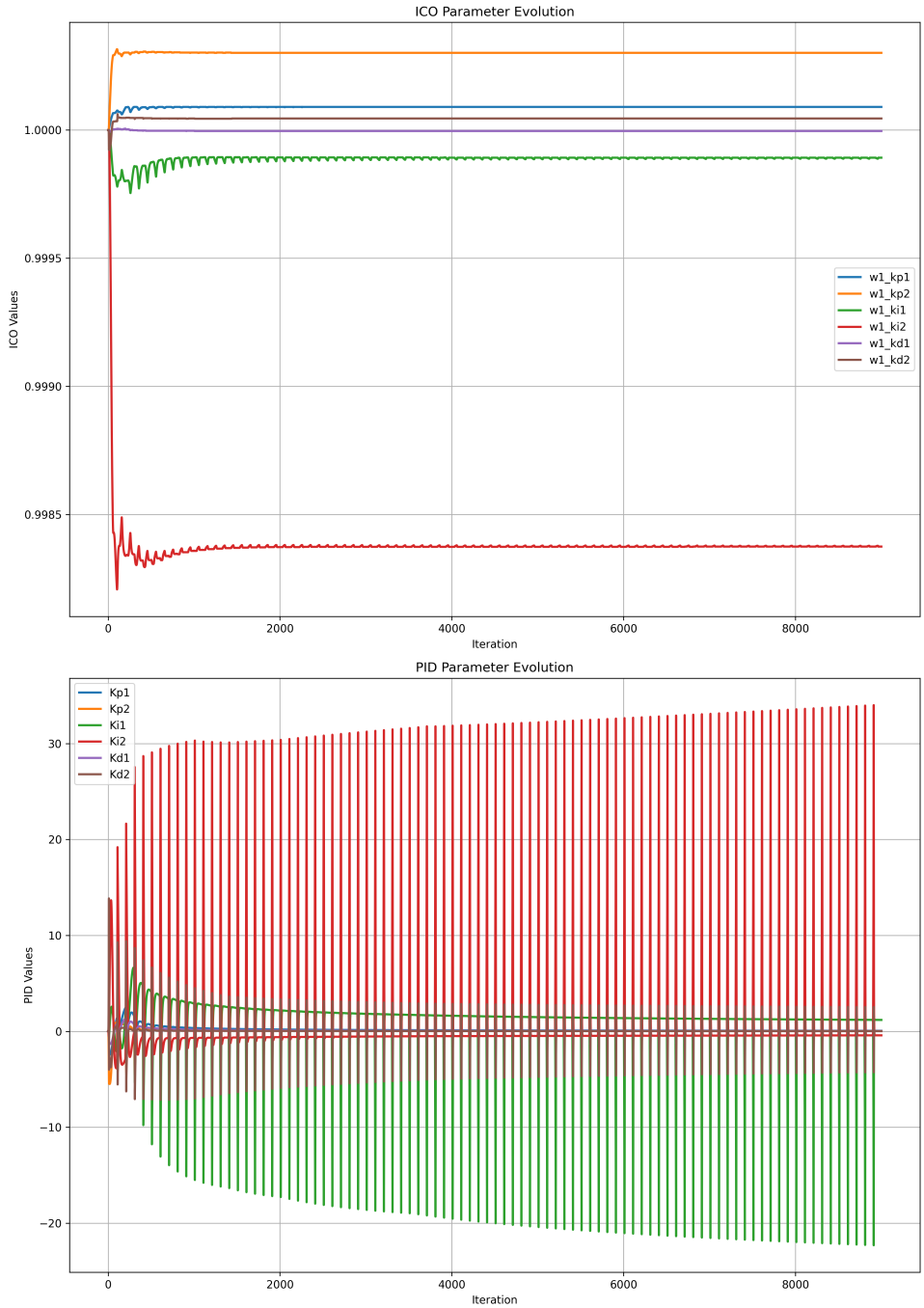


Fig. 80: Evolution of ICO and PID parameters for test scenario S17.2, where the system is subject to periodic disturbances.

7.6 Appendix 6: Miscellaneous

It is possible to recreate every test scenario with the associated MATLAB code from the GitHub page [20]. Otherwise, animated gifs, labeled for each scenario, can be found in the separately attached zip file.
Pictures and videos of the embedded system are also available in the attached ZIP.