Automatic tuning of multiple PID motor controllers for a self-balancing platform via machine learning

Project and Presentation by:
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

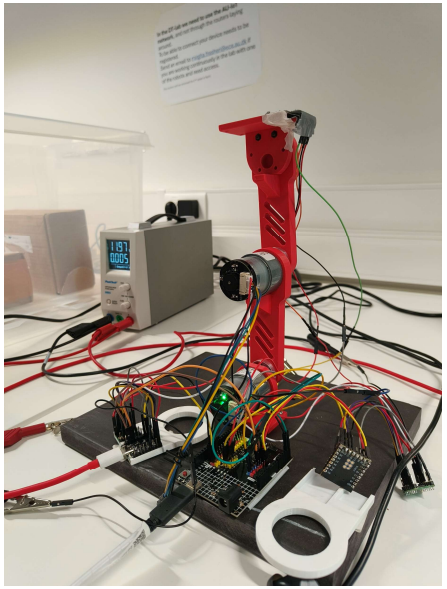
Order of Business

Presentation Focus:

- General outline of the project
- Present project nuances in formats not suitable for a written report
- Highlight topics deserving of more attention

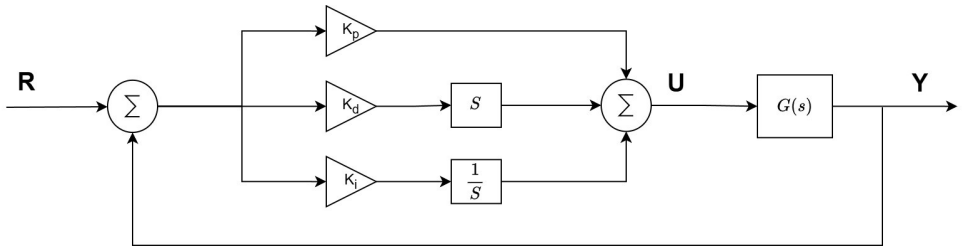
Content:

- Introduction & Problem Description
- Research Methodology & Materials
- Design & System Analysis
- Implementation & Testing
- Results
- Discussion
- Future Work



Introduction and Problem Description

- **PID controllers** remain industry standard for motor control but require manual tuning
- **Multi-motor systems** (e.g., robotic arms) suffer from interdependencies and nonlinearities
- Traditional tuning methods (e.g., Ziegler-Nichols, gain scheduling) are:
  - Time-consuming
  - Rely on accurate system models
  - Inflexible to dynamic environments
- **Challenge:** Online PID tuning for unknown or changing system dynamics
- **Goal**: Develop a learning-based approach to
  - Automatically tune **multiple PID controllers**
  - Handle disturbances and nonlinear behaviour
  - Operate without prior system modelling



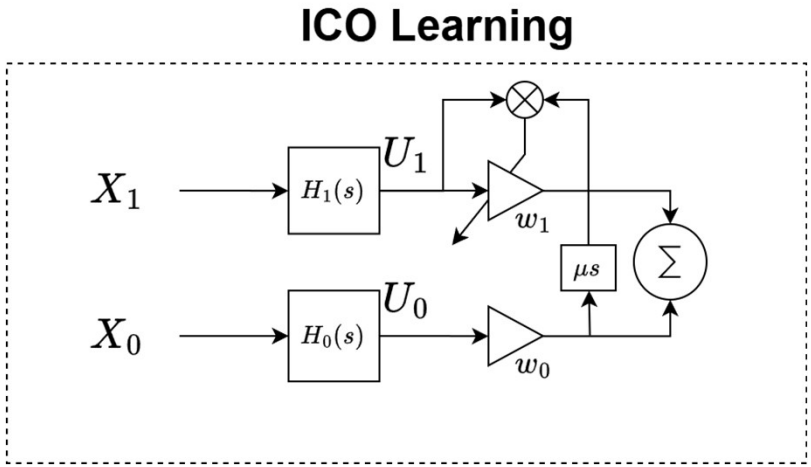
Research Methodology and Materials

Other intelligent control algorithms for auto tuning PID-gains?

- **Yes!**
- Often based on Reinforcement Learning (**RL**)
- Works by **minimizing** an error-based loss function via gradient descent
- Computationally heavy, especially if deployed with many agents or with a Deep Neural Networks **(DNN)** architecture for accurate parameter adaption

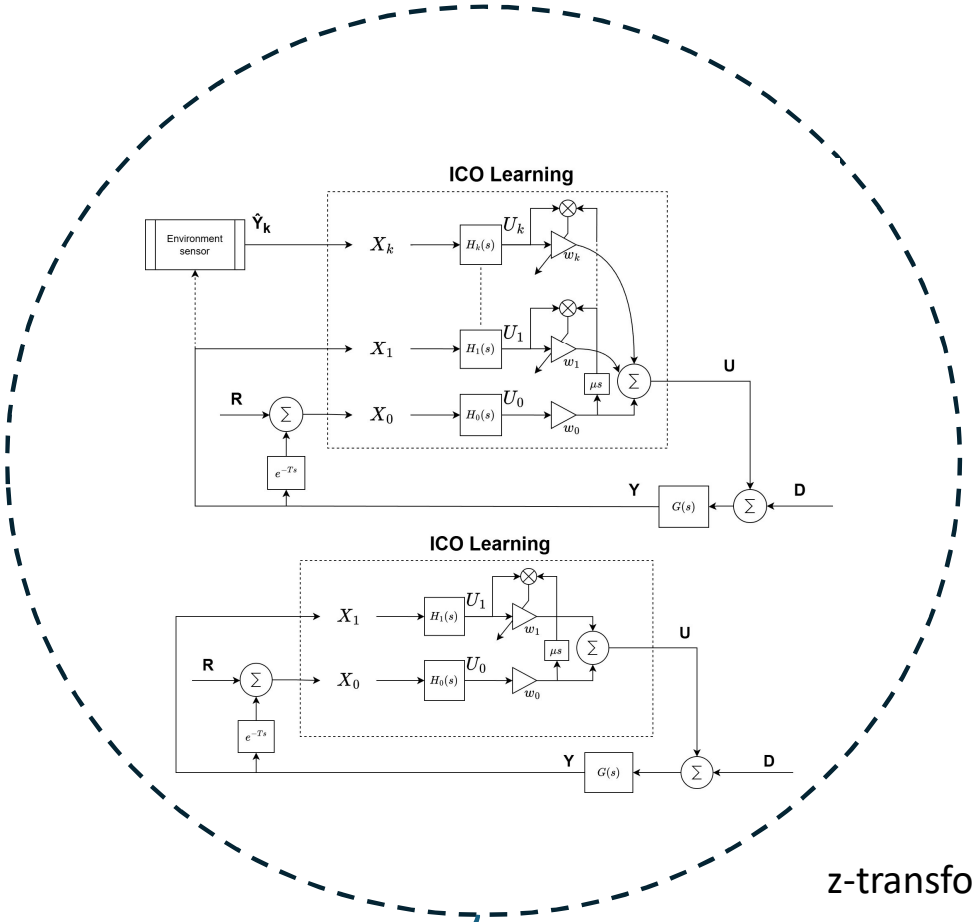
Proposed Solution: Use **Input Correlation (ICO) learning** for:

- Hebbian inspired weight adaptation
- Works by **maximizing** correlation between input signals in time
- Low computational cost
- Used before on embedded systems

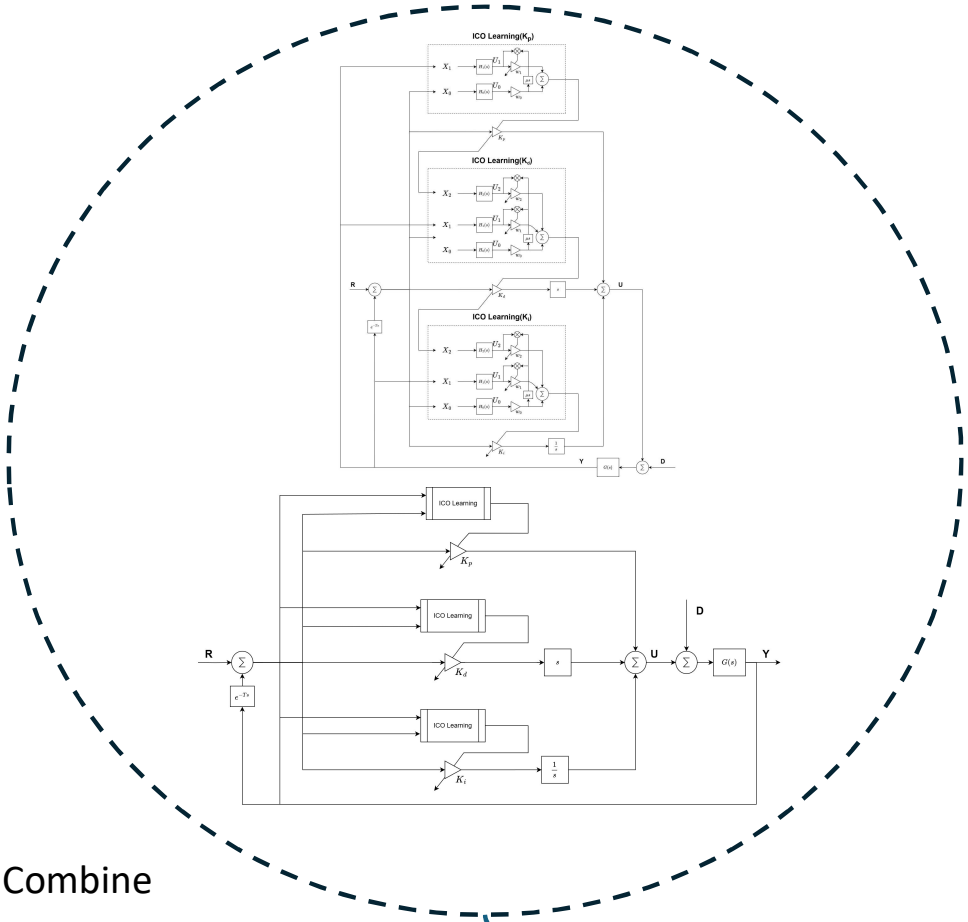


Design and System Analysis

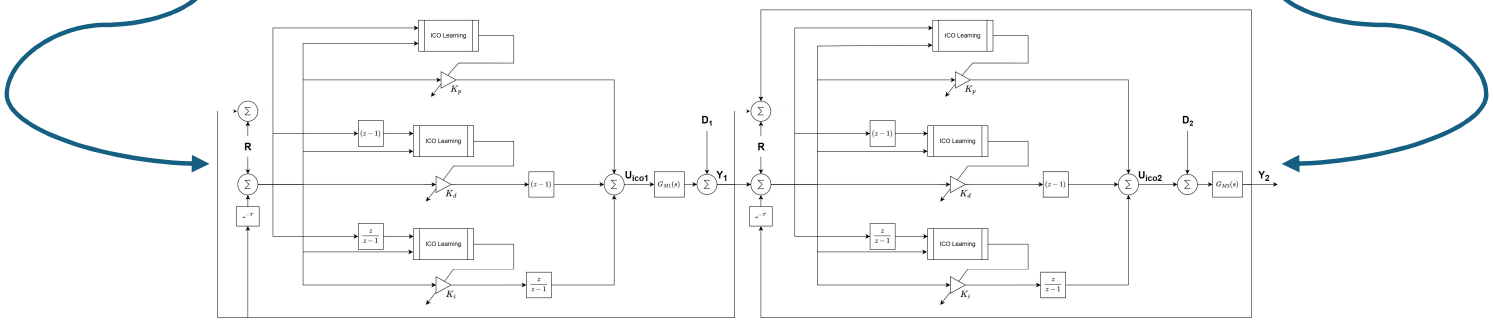
Intelligent Control View



Control Architecture View

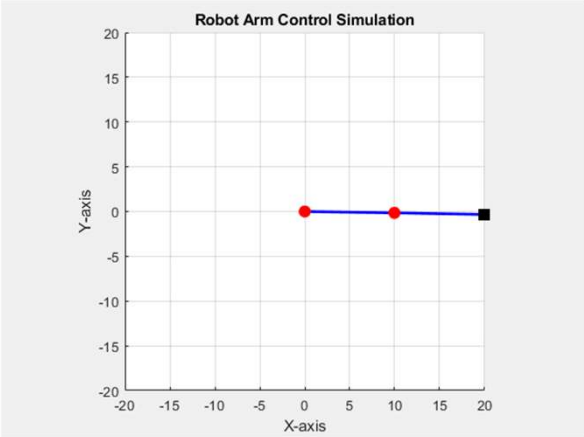


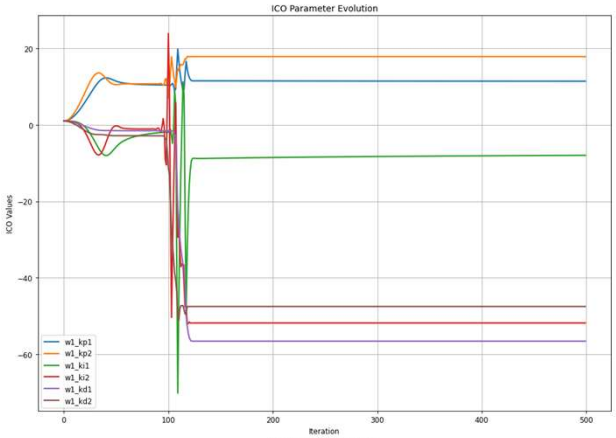
z-transform & Combine

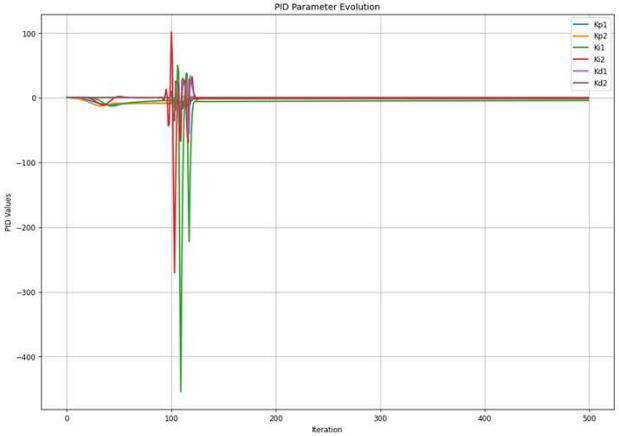


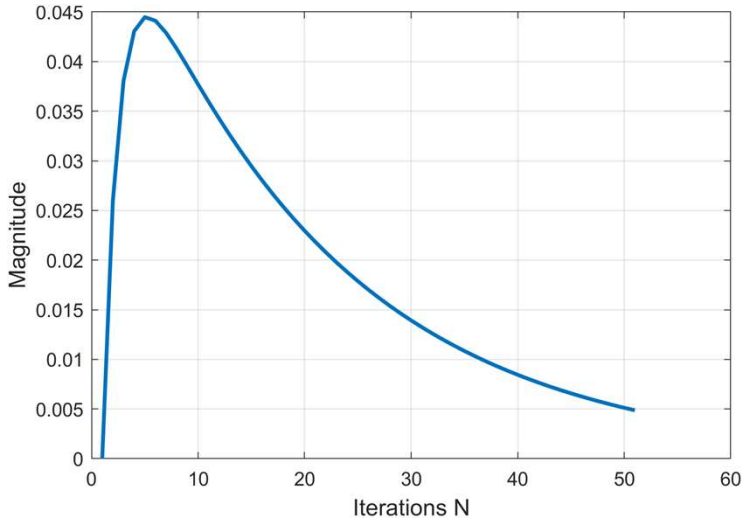
Design and System Analysis - Filtering and Stability

S1.3









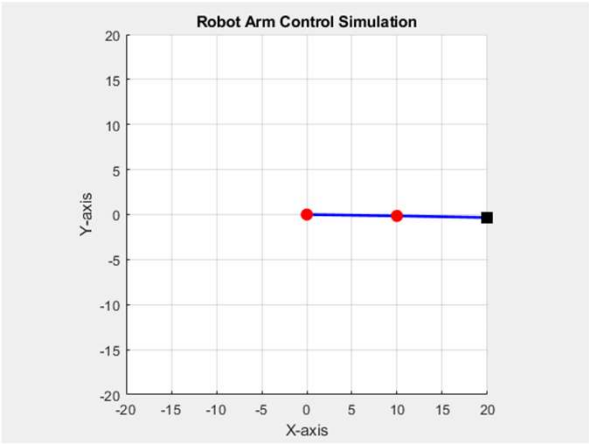
Deduce smoothening delay

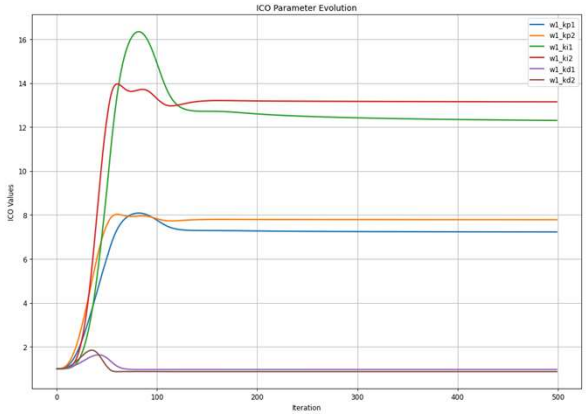
Filtering

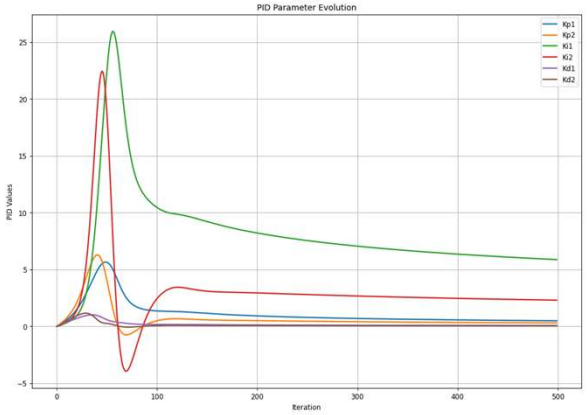
BIBO Stability in weight change response

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

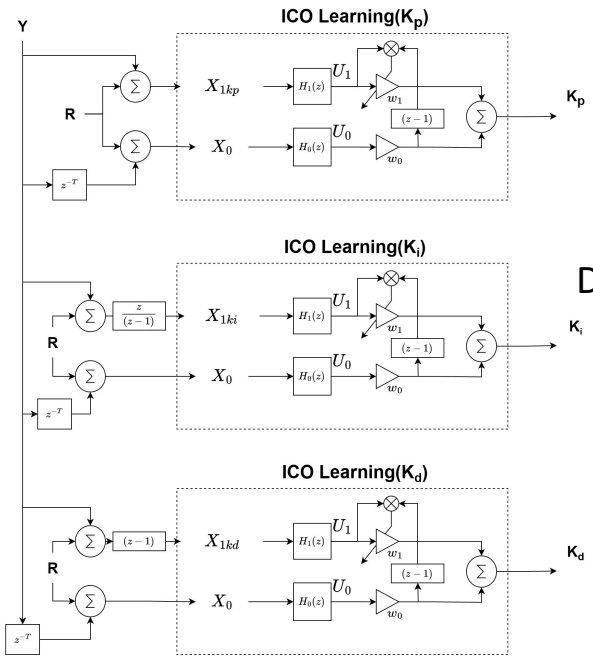
S5.3







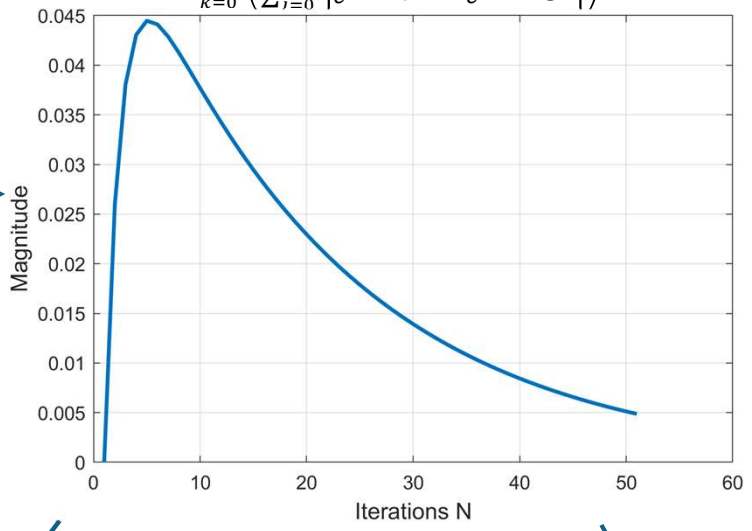
Design and System Analysis - Deduced Filter Models



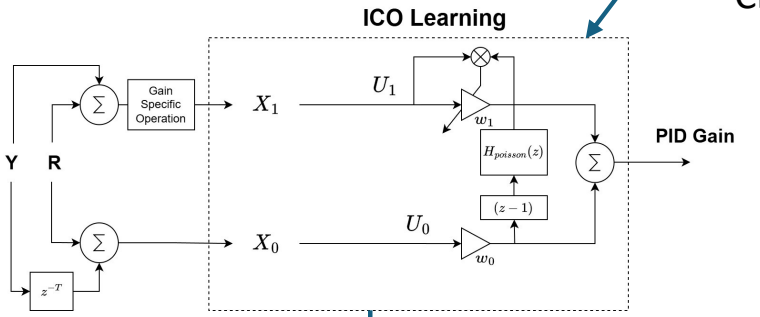
Design FIR Filter

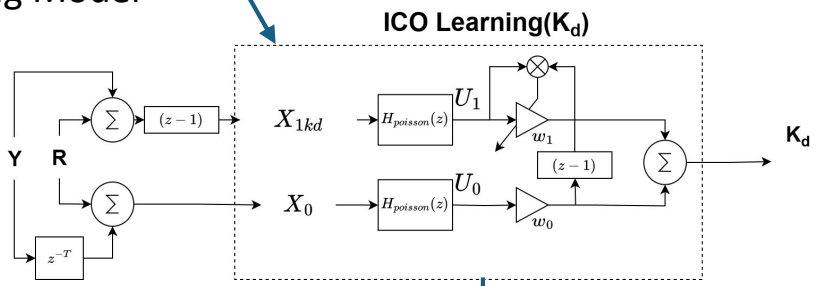
Poisson-like FIR Filter

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



Choose Filtering Model





Update Learning Rules

$$PID\_gain = w_0 x_0[n] + w_1 x_1[n]$$

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

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

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

Implementation and Testing

Pseudocode

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

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

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

3: **for** each time step n **do**

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

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

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

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

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

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

10: 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)

11: Update weights:

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

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

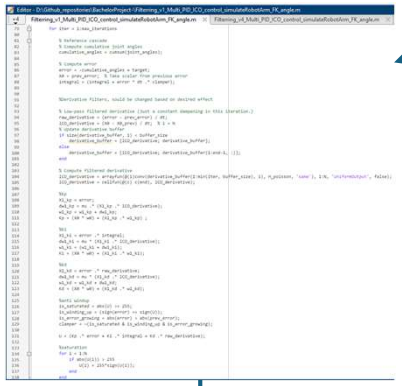
$$X_{1\_kd}[n] \leftarrow error[n] \cdot \Delta error[n]$$
$$\Delta w_{1\_kd} \leftarrow \mu \cdot X_{1\_kd}[n] \cdot \hat{d}_{ICO}[n]$$
$$w_{1\_kd} \leftarrow w_{1\_kd} + \Delta w_{1\_kd}$$

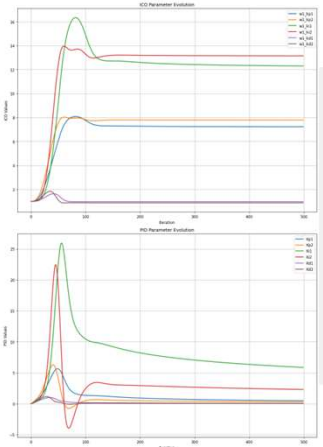
12: Compute PID-gain parameters:

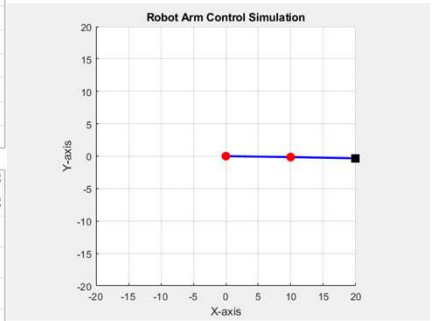
$$K_p \leftarrow w_0 \cdot X_0[n] + w_{1\_kp} \cdot X_{1\_kp}[n]$$
$$K_i \leftarrow w_0 \cdot X_0[n] + w_{1\_ki} \cdot X_{1\_ki}[n]$$
$$K_d \leftarrow w_0 \cdot X_0[n] + w_{1\_kd} \cdot X_{1\_kd}[n]$$

13: **end for**

Simulation Environment Implementation

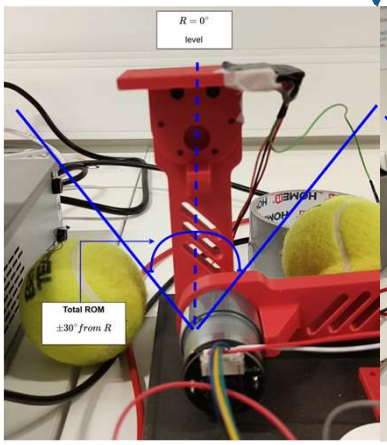


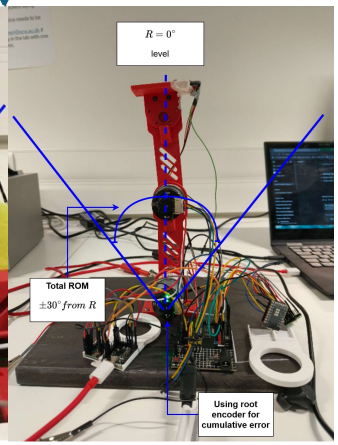




Hardware Environment Implementation

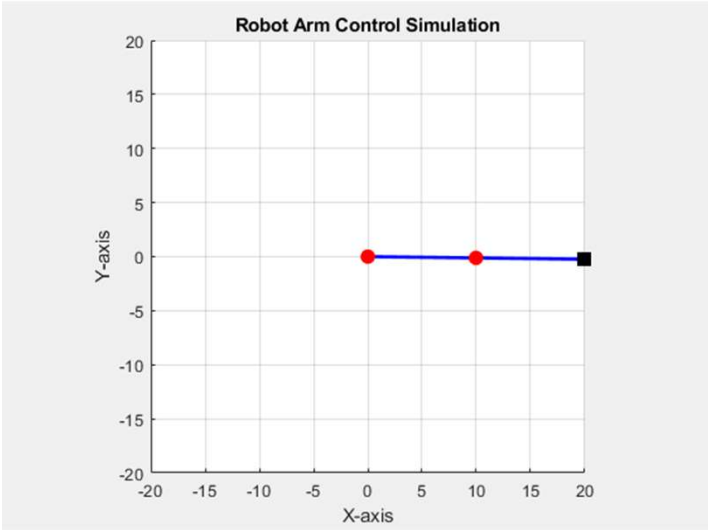
Design Test Criteria



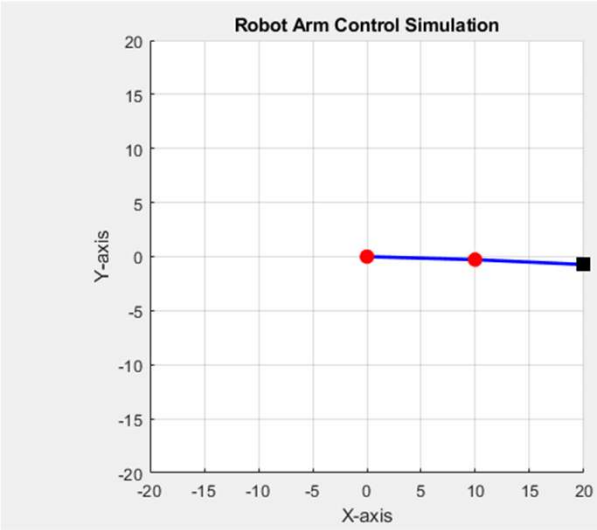


Results - Simulation

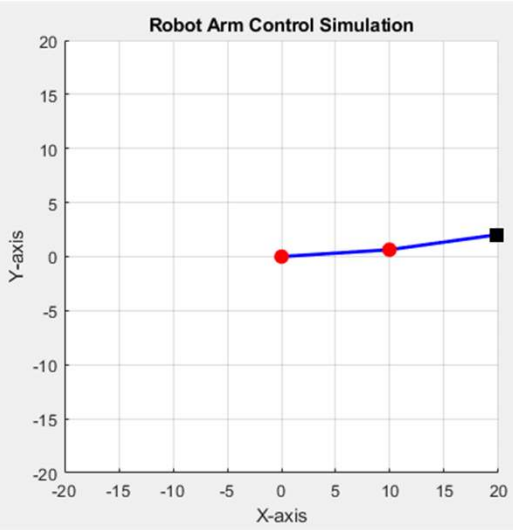
S3.3



S8.3

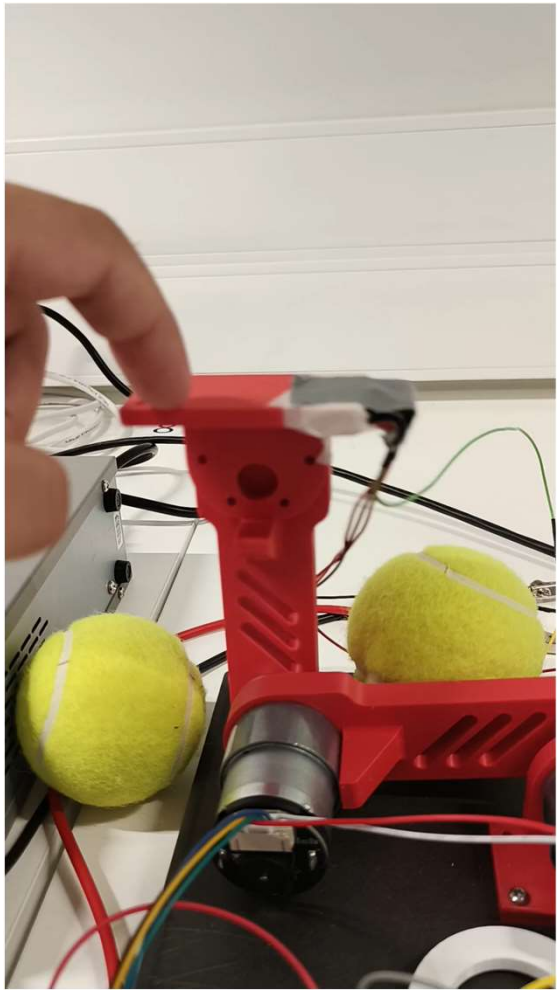


S9.3



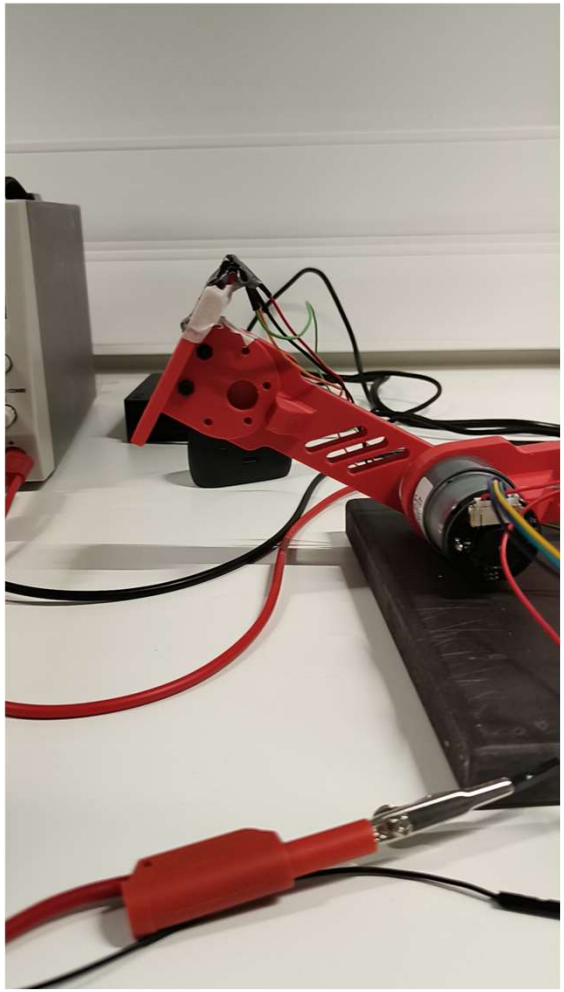
Results - Embedded System

Additional Test 1



MS.2

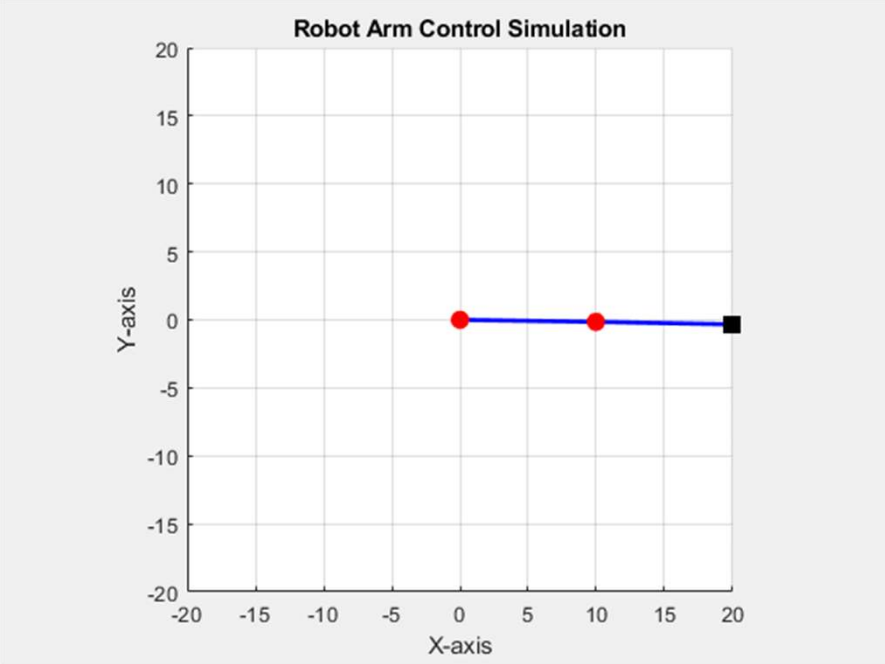
Additional Test 3

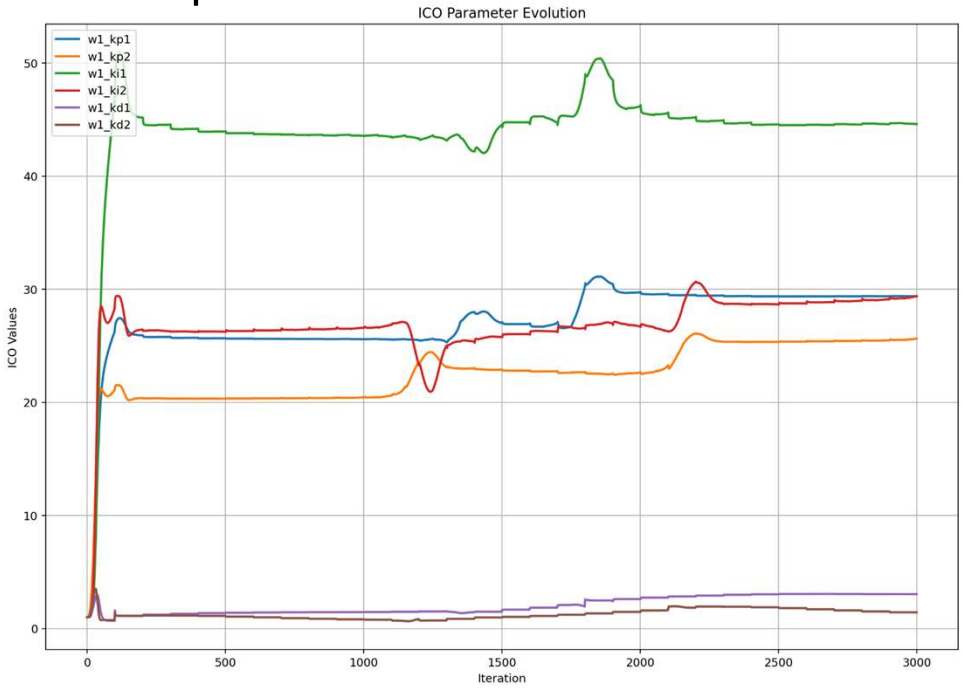


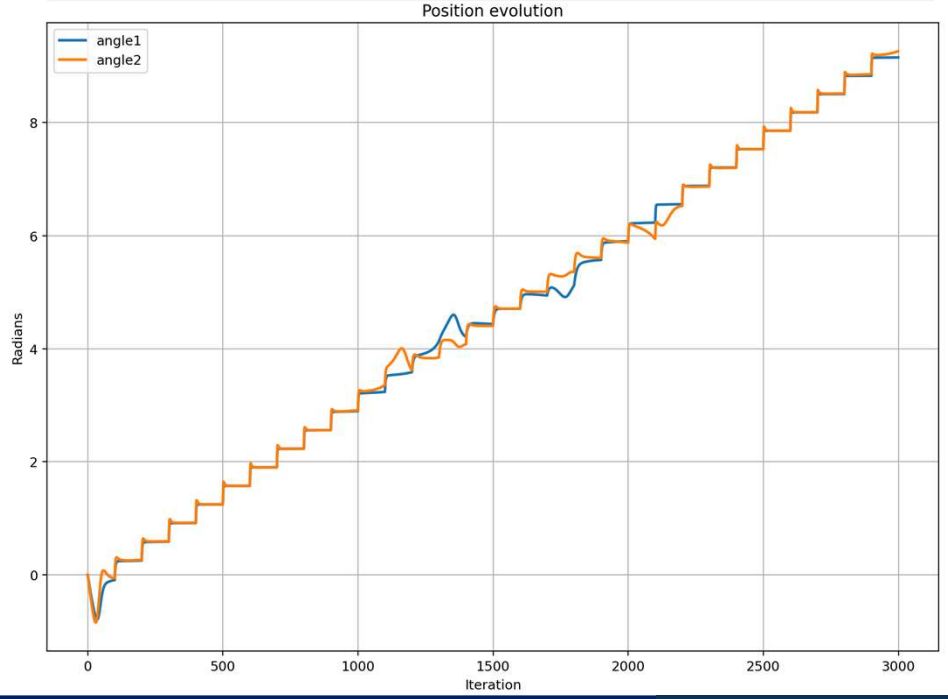
Results - An Added Insight

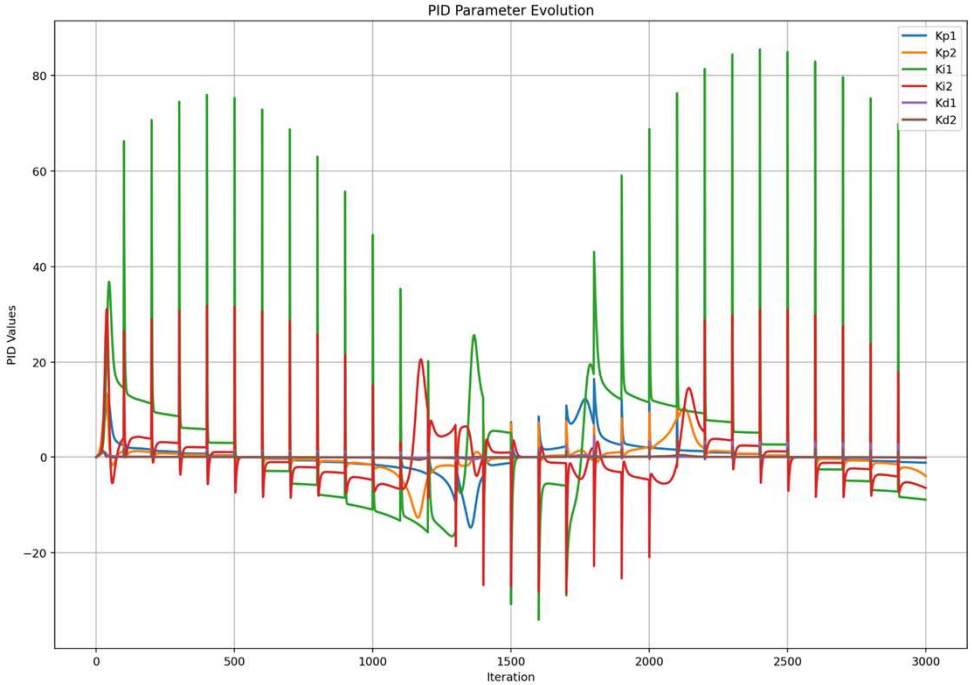
$\mu = 0.1,\ dt = 0.1\ s$

Analog Clock Example





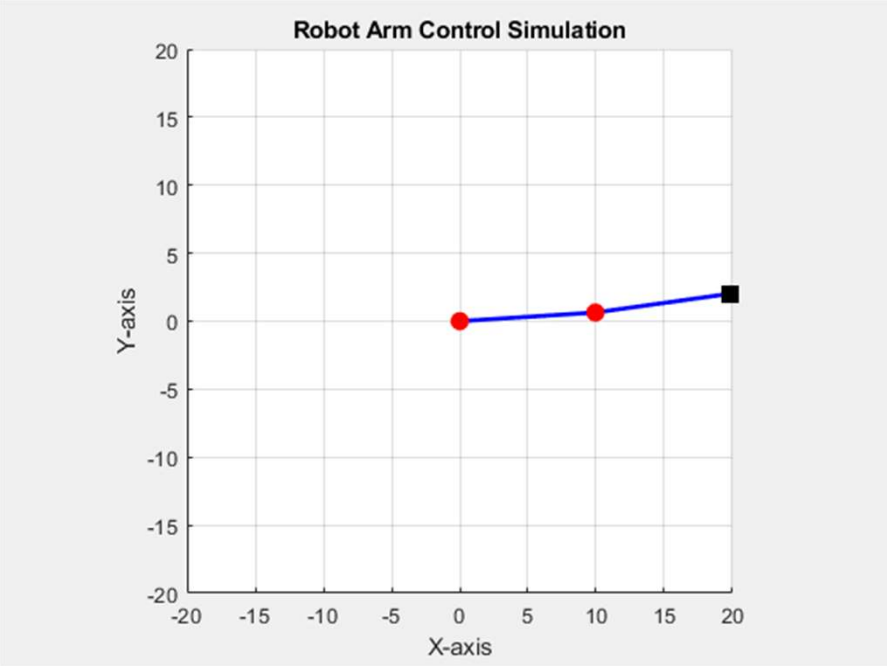


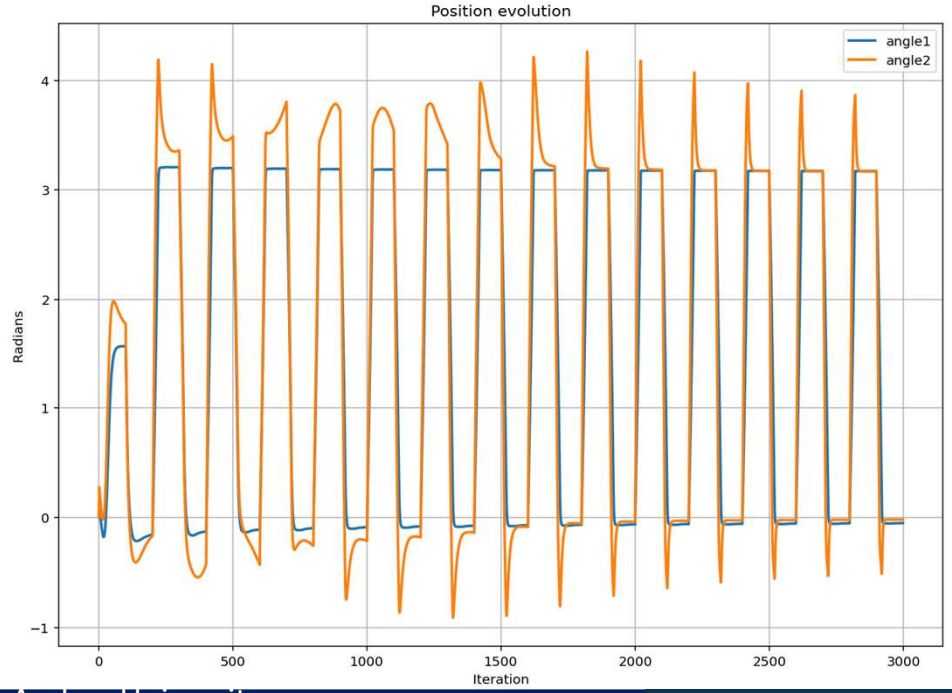


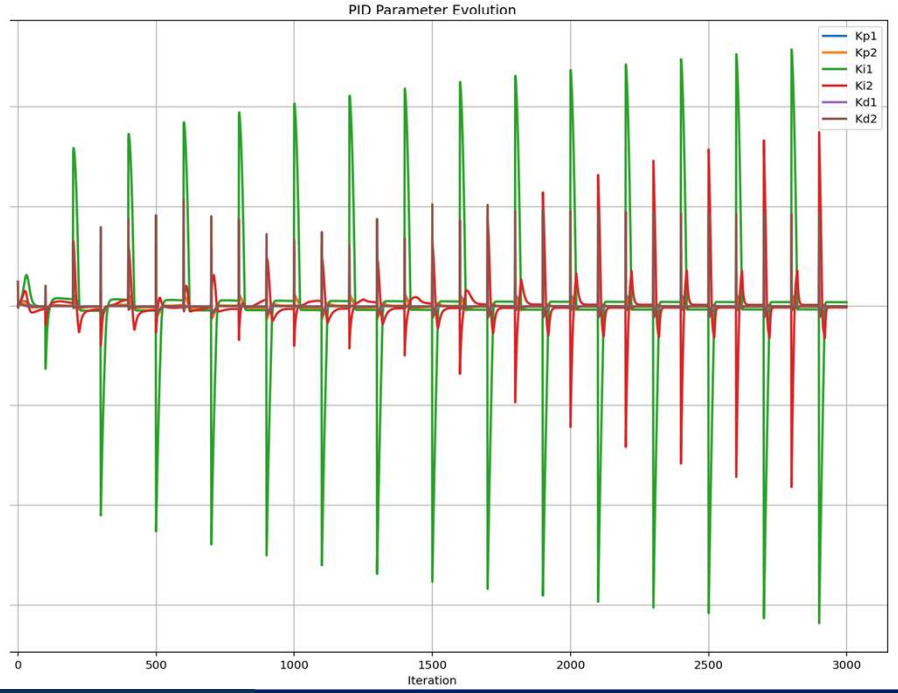
Results - An Added Insight

$\mu = 0.1,\ dt = 0.1\ s$

Metronome Example

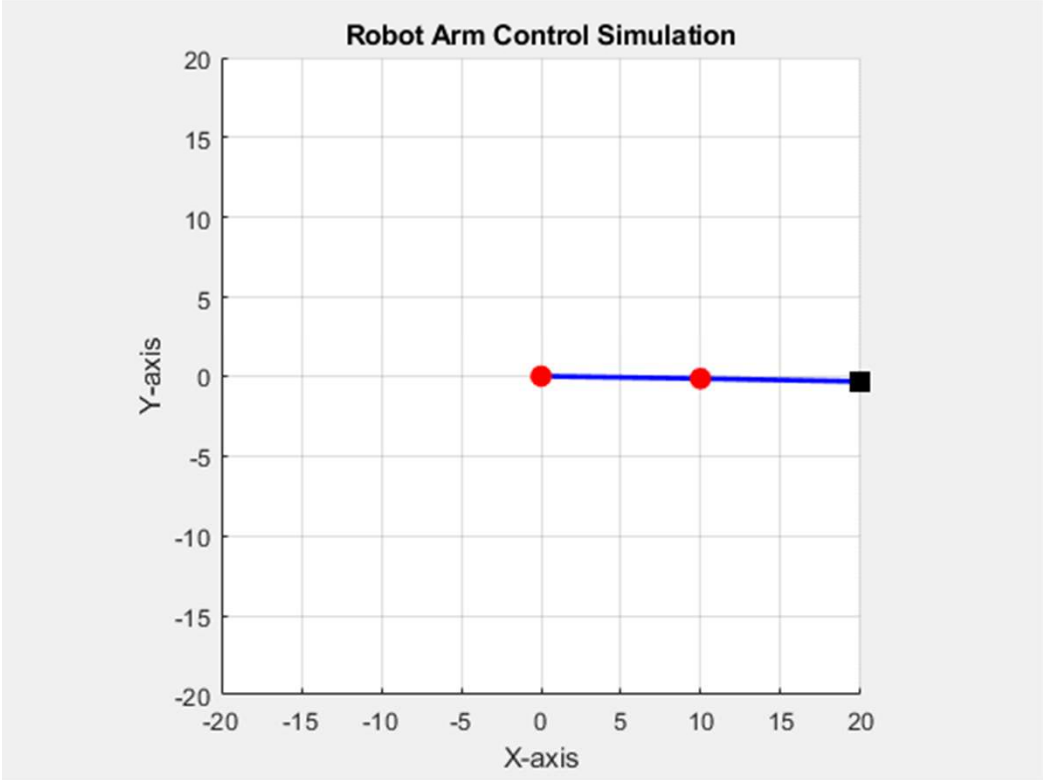


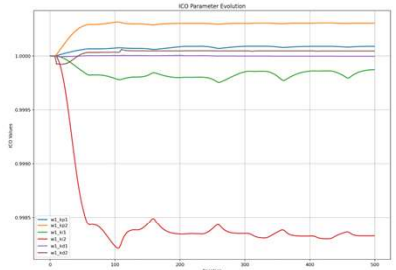


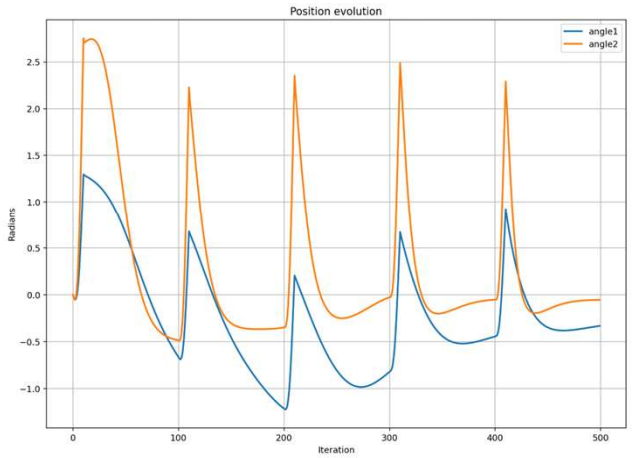


Results - An Added Insight

Untrained:

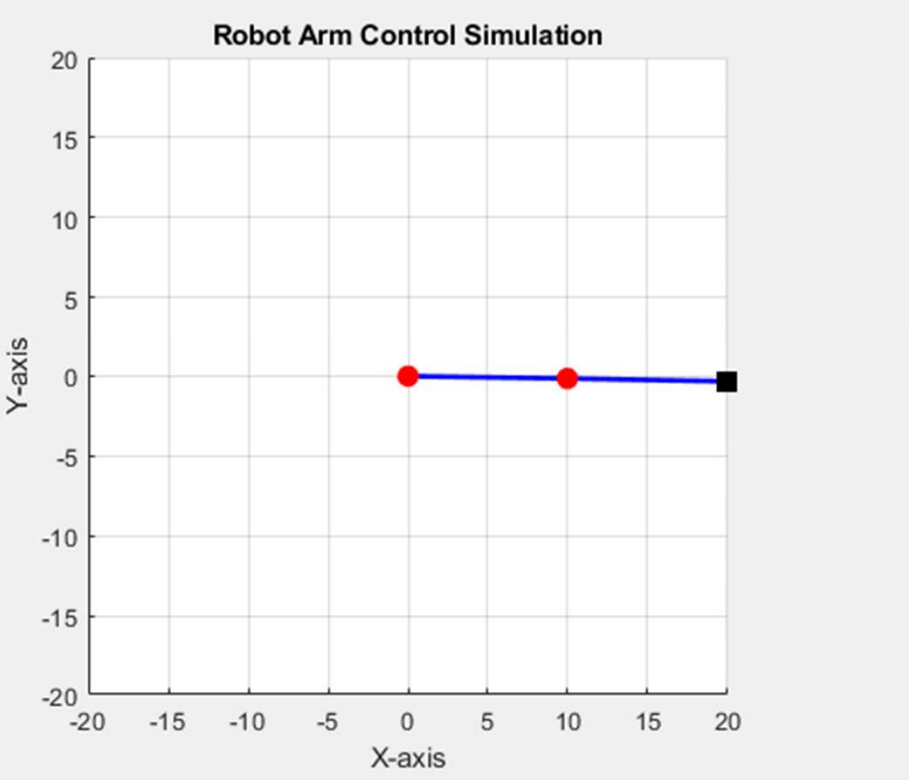


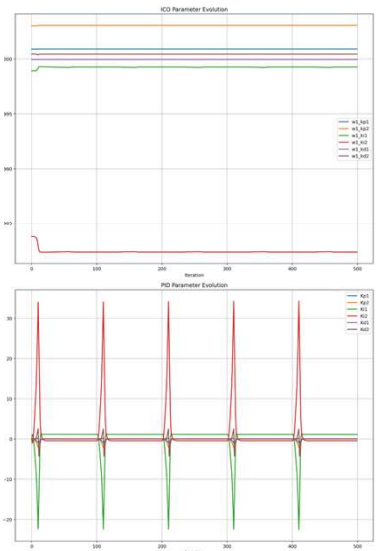


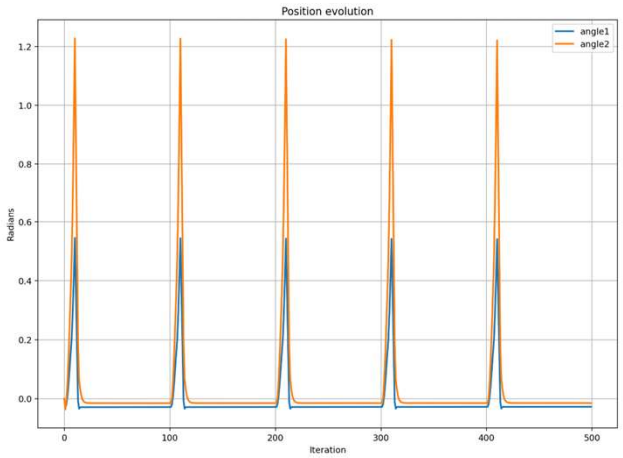


$\mu = 0.0001,\ dt = 0.1\ s$

Trained for 10000 iterations (16 min 40 s):







$\mu = 0.0001,\ dt = 0.1\ s$

Discussion

- Is ICO-PID capable of adapting PID-gain parameters?
  - Adapting to disturbances and gravitational non-linearities?
- Is filtering needed?
  - Working as intended?
- Computation cost efficient compared to other PID auto tuning algorithms?
  - Intelligent control?
- Implementation feasible?
  - Calibration sequence?
  - Necessity for offline training?

Future Work

Hardware:

- Create an embedded system capable of experiencing the full effect of the nonlinear dynamics
- Testing and deployment on multiple arms

Software:

- Develop a standard framework for saving and reapplying states from previous sessions
- Test in alternate nonlinear environments and assert feasibility

References

A. Anthony, “Bsc project: Automatic tuning of multiple pid motor controllers for a selfbalancing platform via ml.” Github Repository Link, 2025.

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

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.

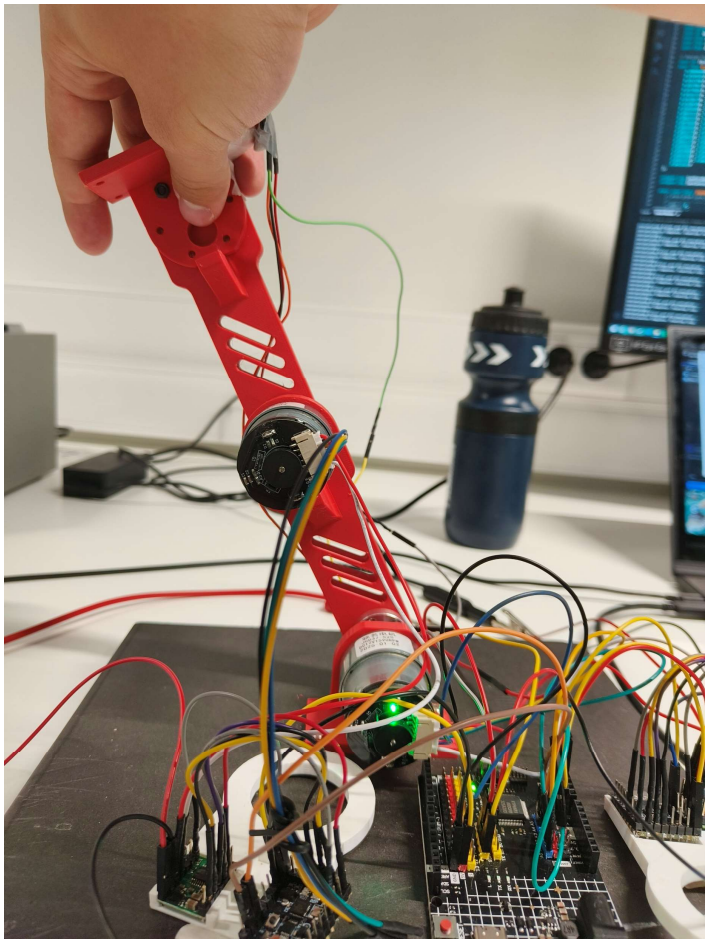
F. Wörgötter and B. Porr, “Temporal sequence learning, prediction, and control: a review of different models and their relation to biological mechanisms,” Neural computation, vol. 17, no. 2, pp. 245–319, 2005.

B. Porr and F. Wörgötter, “Isotropic sequence order learning,” Neural Computation, vol. 15, no. 4, pp. 831–864, 2003.

K. J. Åström, “Ziegler-nichols auto-tuners,” 1982.

Drawio, Tool used to create figures for the documentation, https://draw.io

Max physically feasible angle of motor:



DS.1: