

Control of Mechatronic Systems – Assignment 3  
Project Group 0

Abhijay Sekhar Choudhury (25592318)

Felicia Persson (26130789)

Subesh Shanmugam (26180594)

Jason Stewart (25182902)

14 October 2025

# Contents

<b>1</b>	<b>Introduction</b>	<b>3</b>
1.1	Background . . . . .	3
1.2	Purpose and Objectives . . . . .	3
1.3	Limitations and Scope . . . . .	3
<b>2</b>	<b>Methodology</b>	<b>4</b>
2.1	System Architecture . . . . .	4
2.2	Dish Generation and Tracking . . . . .	4
2.3	Gripper Control . . . . .	5
2.4	Defining the Workspace and Sorting Areas . . . . .	5
2.5	State-Space Model Development . . . . .	6
2.6	Design Constraints . . . . .	8
2.7	DPID Model . . . . .	8
2.7.1	Initialisation . . . . .	8
2.7.2	System Overview . . . . .	8
2.8	Smooth Trajectory and Feedforward Model . . . . .	10
2.8.1	Trajectory Planning . . . . .	10
2.8.2	Initialisation . . . . .	10
2.8.3	System Overview . . . . .	10
2.9	Bayesian Optimisation of Parameters . . . . .	11
2.10	Simulation and Simulation Features . . . . .	11
<b>3</b>	<b>Results and Discussion</b>	<b>11</b>
3.1	Tuned Parameters and Performance Overview . . . . .	11
3.2	Comparative Controller Analysis . . . . .	13
3.3	Practical Implications and Future Development . . . . .	16
<b>4</b>	<b>Conclusion</b>	<b>16</b>

# 1 Introduction

## 1.1 Background

Various menial sorting tasks aim to have input material be sorted into dedicated zones based on their properties. This could be constructed as a Cartesian robot, which operates based on the Cartesian coordinate system as its movement is controlled along the x-, y-, and z-axes. These are commonly used for pick-and-place operations of objects through linear actuators, transforming the motor rotation into linear motions in three-dimensional space, controlled by some control strategy to switch between desired positions (Mushiri & Moyo, 2023). To complement the basic Cartesian robot setup, a vision system can be used for reading input materials, to enable identification and sorting mechanisms.

## 1.2 Purpose and Objectives

The purpose of this project is to develop a control system for a robotic manipulator that performs accurate automatic sorting and pick-and-place operations, using a robotic arm, vision system, and motion control strategy. The system is supposed to demonstrate state-space modelling principles applied to practical automation problems.

To achieve this, the objectives of this project are to:

- Implement two control systems – one based on a direct and discrete destination-calculated proportional-integral-derivative (DPID) strategy and another combining trajectory smoothing with feedforward PID (Smooth+FF/TS)- that both use a vision system to categorise input material according to certain distinguishing characteristics to identify their sorting destination, pick up the material, and place the items into their designated output areas, while minimising sorting time and maintaining accuracy;
- Implement a system that achieves automatic sorting with a success rate of more than 90%, and a tracking error below 100 mm;
- Compare the two control strategies in terms of various performance metrics.

## 1.3 Limitations and Scope

The project is limited to operation entirely through simulation within MATLAB and Simulink, as no actual robotic hardware is used, which means that real-world irregularities are not included. Furthermore, the complexity of the system is limited by the use of simplified robot dynamics, as it is modelled using a Cartesian gantry system along three axes with a decoupled linear state-space model. This means that while linear viscous damping, axis-specific effective masses accounting for the stacked mechanical configuration (where the X-axis carries the Y and Z assemblies, the Y-axis carries the Z assembly, and the Z-axis carries only the gripper-camera), and gravity compensation are implemented, the simulation does not include non-linear friction, cross-axis dynamic coupling (where motion in one axis affects forces or dynamics in others), configuration-dependent inertial effects, structural compliance, and other real-world complexities. Each axis is modelled as an independent second-order system with constant parameters. As for the vision system, categorisation is simulated using identification of randomly generated codes representing dish types. As no noise is implemented, real-world vision errors such as misreading or misclassification are not taken into account. The workspace itself and its designated areas, as well as the initial objects' positions, are defined by pre-determined coordinates, which is reasonable also for a real-world setup. As for control strategy, two models are compared: one using DPID, and one using a trajectory smoothing system with feedforward control (TS). For parameter tuning, Bayesian optimisation is used. Model Predictive Control (MPC) parameters for optimal control were defined but not implemented, and state estimation using Kalman filtering is not included in the current system.

## 2 Methodology

In the following section, the methodology of this project is presented, beginning with defining and modelling the system, and then implementing two models with different control strategies. In addition, the models' initialisation and simulation in MATLAB and Simulink is presented. Lastly, the Bayesian optimisation of parameters is described.

### 2.1 System Architecture

The system of the Cartesian robot consists of three main components; a robotic arm, a vision system and control strategy for movement.

To illustrate what the real-life robot could look like, the following visualisation of a Cartesian robot can be used as an example;

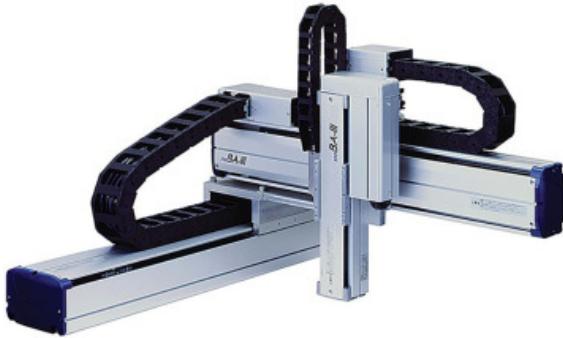


Figure 1: Cartesian robot architecture (Mushiri & Moyo, 2023)

As can be seen in Figure 1, the Cartesian robot consists of an arm that can move in three directions: along the x-, y-, and z-axes. The x- and y-direction makes up the horizontal movement of the robot between the various designated areas within the working space, while the z-direction is for height adjustments connected to picking up, dropping off, and transporting the material above the horizontal plane.

The Cartesian robot's vision system allows for object identification, using a simulated camera providing object type classification by reading its individually assigned code, which represents the dish-type. Input material has been thought of as dishes, as these are regular shapes for which physical labels and codes may be assigned more easily than other shapes. Cylinders are also more easily handled by robotic grippers (though gripper design and handling capability analysis falls out of scope of this project to fully back this claim).

For motor control, the Cartesian robot's position along the x-, y-, and z-axes is regulated using two alternative control strategies; one model uses the aforementioned DPID controller, while the other complements PID feedback with a smooth trajectory generator and feedforward terms. These models are further detailed in Sections 2.7 and 2.8, respectively.

### 2.2 Dish Generation and Tracking

To simulate the input material to be sorted, 64 dishes are generated per session by assigning each dish a random three-digit number, with each digit ranging from 1 to 3. This makes up a total of 27 possible combinations, or dish types, of which the 64 dishes could be. In each session, a new set of 64 randomly chosen numbers is generated (using a shuffled random seed in the Simulink models' Vision System). This allows simulation of how a real camera would read and classify each dish and ensures that the system is challenged to identify, sort, and track various sets of inputs in multiple subsequent sessions. This introduces a degree of variance between simulation instances of the same model. Due to this, optimal parameters tuned

for each model are expected to result in slightly different performance metrics when simulations are run for the same parameters.

### 2.3 Gripper Control

As for the gripper, no actual dynamics are simulated. Instead, the gripping and releasing of dishes is managed using a binary logic, where “PICK” and “PLACE” are handled using transitions between states in the State Machine, rather than mechanical simulation. These binary states are triggered by movements along the z-axis, specifically the height at which dishes are to be picked up and released according to the system’s State Machine.

### 2.4 Defining the Workspace and Sorting Areas

As the robot should perform sorting tasks, it needs to know where every relevant area of the workplace is located, as well as the limits of the working area as a whole. To ensure that, the workspace and its different areas are defined. The full workspace is made up of a region with set dimensions of 1300 x 800 x 300 mm, and consists of four areas; the input area, the output area, the disposal area, and the buffer area, as can be seen in the following plot:

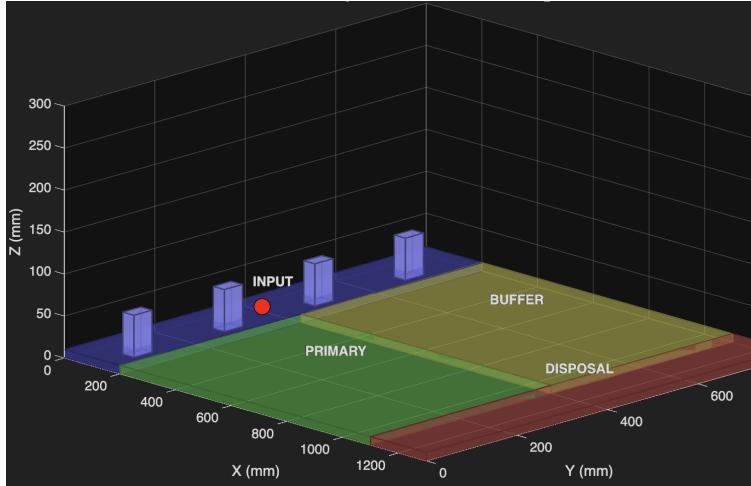


Figure 2: Workspace Overview

As can be seen in Figure 2, the input area (blue) has four input stacks laid out in a row, where each stack holds 16 unsorted dishes of various types. This means that in total, the input zone holds 64 randomly generated dishes from the 27 possible types. In other words, before the sorting is initiated in each session, this is where all of the unsorted dishes are located.

The output area (green) consists of a 2x8 grid with 16 output slots, where the sorted dishes are placed. Each slot can hold up to 10 dishes of a single dish type. A dish type may occupy only one output slot at a time. If that slot becomes full or was completed in a previous session, any additional dishes of that type are excluded from the output area.

In such cases, the dish is sent to the disposal area (red), which contains seven slots for rejected items. If a new dish type is identified but there are no empty slots available in the output area, the dish is instead placed in the buffer area (yellow), a 2x8 grid of 16 buffer slots dedicated to overflow dishes.

## 2.5 State-Space Model Development

To model the system, its states are defined as  $p_i$ , the position on axis  $i$  in millimetres, and  $v_i$ , the velocity on axis  $i$  in millimetres per second, where  $i = \{x, y, z\}$ . Based on these, the state vector can be constructed as follows:

$$\mathbf{X} = \begin{bmatrix} p_x \\ v_x \\ p_y \\ v_y \\ p_z \\ v_z \end{bmatrix}$$

The input is defined as the applied force among the axes  $i = \{x, y, z\}$ ,  $F_i$ , which gives the following input vector:

$$\mathbf{u} = \begin{bmatrix} F_x \\ F_y \\ F_z \end{bmatrix}$$

The system output is chosen as the position on each axes —  $p_x$ ,  $p_y$  and  $p_z$  — and thus gives the output vector as:

$$\mathbf{y} = \begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix}$$

The full state-space model is derived by first defining the state-space model of the system for each of the three axes  $i = \{x, y, z\}$  individually. This is done using Newton's second law with viscous friction:

$$m_i \ddot{p}_i = F_{\text{res}} = F_i - F_{f_i} = F_i - b_i \dot{p}_i \Rightarrow m_i \ddot{p}_i + b_i \dot{p}_i = F_i$$

where  $m_i$  is the mass along axis  $i$ ,  $b_i$  is the viscous friction constant along axis  $i$ , and  $F_{f_i}$  is the viscous friction force along axis  $i$ . Solving for acceleration gives the following equation:

$$\ddot{p}_i = -\frac{b_i}{m_i} \dot{p}_i + \frac{1}{m_i} F_i$$

By defining  $x_1 = p_i$  and  $x_2 = \dot{p}_i$ , the state vector for axis  $i$  can be written as

$$\mathbf{x}_i = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} p_i \\ \dot{p}_i \end{bmatrix}$$

with  $F_i$  as axis input and  $p_i$  as axis output. Then the system can be written using equations

$$\begin{cases} \dot{x}_1 = \dot{p}_i = x_2 \\ \dot{x}_2 = \ddot{p}_i = -\frac{b_i}{m_i} x_2 + \frac{1}{m_i} F_i \end{cases}$$

By rewriting these equations to matrix form, the state-space model of the system for axis  $i$  is derived:

$$\begin{cases} \dot{x}_1 = x_2 \\ \dot{x}_2 = -\frac{b_i}{m_i} x_2 + \frac{1}{m_i} F_i \\ y_i = p_i \end{cases} \iff \begin{cases} \dot{\mathbf{x}}_i = \begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} x_2 \\ -\frac{b_i}{m_i} x_2 + \frac{1}{m_i} F_i \end{bmatrix} \\ y_i = p_i \end{cases}$$

$$\iff \begin{cases} \dot{\mathbf{X}}_i = \underbrace{\begin{bmatrix} 0 & 1 \\ 0 & -\frac{b_i}{m_i} \end{bmatrix}}_{A_i} \mathbf{X}_i + \underbrace{\begin{bmatrix} 0 \\ 1 \end{bmatrix}}_{B_i} F_i \\ y_i = \underbrace{\begin{bmatrix} 1 & 0 \end{bmatrix}}_{C_i} \mathbf{X}_i = p_i \end{cases}$$

Furthermore, gravity introduces an additional force in the z-direction, which can be represented by adding a constant term to the equation from Newton's second law:

$$m_z \ddot{p}_z + b_z \dot{p}_z = F_z - m_z g.$$

To keep a linear time-invariant (LTI) model, gravity is included by instead biasing the input

$$F'_z = F_z - m_z g \Rightarrow \ddot{p}_z = -\frac{b_z}{m_z} \dot{p}_z + \frac{1}{m_z} F'_z$$

In turn, gravity is included without breaking linearity, by representing gravity using a constant bias term in the z-axis input  $F_z$ .

Lastly, effective masses are introduced to account for the mechanical construction of the simulated Cartesian robot, considering initial coupling of the system: X carries Y and Z, and Y carries Z. This captures that accelerating in the x-direction moves the carried mass in all three directions, and acceleration in the y-direction carries the masses in y- and z-direction, by defining the effective masses as

$$m_x^{\text{eff}} = m_x + m_y + m_z, \quad m_y^{\text{eff}} = m_y + m_z, \quad m_z^{\text{eff}} = m_z$$

Thus,  $m_i^{\text{eff}}$  is used in the above formulas for the state-space model in  $A_i$  and  $B_i$  for each direction  $i = \{x, y, z\}$  instead of  $m_i$ . Then, by combining the state-space model for each of the three axis, accounting for gravity and effective masses, the following resulting full state-space model for the system is derived:

$$\dot{\mathbf{X}} = \underbrace{\begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & -\frac{b_x}{m_x^{\text{eff}}} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & -\frac{b_y}{m_y^{\text{eff}}} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & -\frac{b_z}{m_z^{\text{eff}}} \end{bmatrix}}_{\mathbf{A}} \begin{bmatrix} p_x \\ v_x \\ p_y \\ v_y \\ p_z \\ v_z \end{bmatrix} + \underbrace{\begin{bmatrix} 0 & 0 & 0 \\ \frac{1000}{m_x^{\text{eff}}} & 0 & 0 \\ 0 & 0 & 0 \\ 0 & \frac{1000}{m_y^{\text{eff}}} & 0 \\ 0 & 0 & 0 \\ 0 & 0 & \frac{1000}{m_z^{\text{eff}}} \end{bmatrix}}_{\mathbf{B}} \begin{bmatrix} F_x \\ F_y \\ F_z \end{bmatrix}$$

$$\mathbf{y} = \underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}}_{\mathbf{C}} \begin{bmatrix} p_x \\ v_x \\ p_y \\ v_y \\ p_z \\ v_z \end{bmatrix} + \underbrace{\mathbf{0}}_{\mathbf{D}} \begin{bmatrix} F_x \\ F_y \\ F_z \end{bmatrix},$$

where the elements of  $\mathbf{B}$  are scaled by 1000 to convert the unit of the output from meters to millimetres, consistent with the system coordinate system.

This state-space model is used for both the model using DPID and the one using smooth-trajectory and feedforward, as only the control strategy applied to the system differs.

## 2.6 Design Constraints

To make the system more realistic, constraints are defined for the workspace, kinematics, and operations;

- The workspace area is limited to dimensions  $1300\text{ mm} \times 800\text{ mm} \times 300\text{ mm}$  ( $x, y, z$ ). While this is a large area for some real-life implementations of the system, it ensures restriction of the motion appropriately for this project.
- Velocity limits are set to a maximum of  $500\text{ mm/s}$  along the  $x$ - and  $y$ -axes, and a maximum of  $300\text{ mm/s}$  along the  $z$ -axis.
- Acceleration limits are set to a maximum of  $1000\text{ mm/s}^2$  along the  $x$ - and  $y$ -axes, and a maximum of  $500\text{ mm/s}^2$  along the  $z$ -axis.
- Each of the 16 output slots can hold up to 10 dishes, meaning that a maximum of 160 dishes can be stored in the output area at once. Further dishes would go to buffer or disposal.
- 64 dishes are processed each session. When this number of processed dishes is reached, the simulation enters the completion mode of that session, and is reset for a new session.

## 2.7 DPID Model

The first implemented control strategy is a waypoint-based controller, also known as discrete (or direct) PID (DPID). It uses discrete waypoints to move the Cartesian robot by making step changes in target position, using reactive “on-the-fly” control to correct errors. The movement along each axis ( $x, y, z$ ) is controlled independently by the DPID controllers. They use PID feedback to correct position error between the actual position and the next desired position, which is given when pick or place has occurred, to make stepwise transitions between the discrete target positions. With its simpler implementation, the DPID model serves as the baseline control strategy for the robotic sorting system, against which the smooth-trajectory feedforward controller is compared. This section outlines model initialisation and the Simulink system structure.

### 2.7.1 Initialisation

Before the simulation starts, initialisation is done to set consistent conditions for each session by either loading or resetting variables, parameters and database setup. If the database file exists, data from previous sessions are restored from `database_state.mat`. Otherwise, a new database is initialised with empty records of dish types, slot counts and completed dish types. As for the system parameters, the system is defined by defining the state-space model metrics (according to the derived state-space model in section 2.5), as well as the masses, effective masses, viscous damping coefficients, and gravity compensation. The workspace area, velocity and acceleration constraints are set, and the PID controller parameters are loaded (either default, or Bayesian optimised if available). Also, simulation parameters are set, such as the 1 ms sample time, and the 64 input dishes for the session are randomly generated.

### 2.7.2 System Overview

The following Simulink block diagram gives a full overview of the robot sorting system and all of its functional modules:

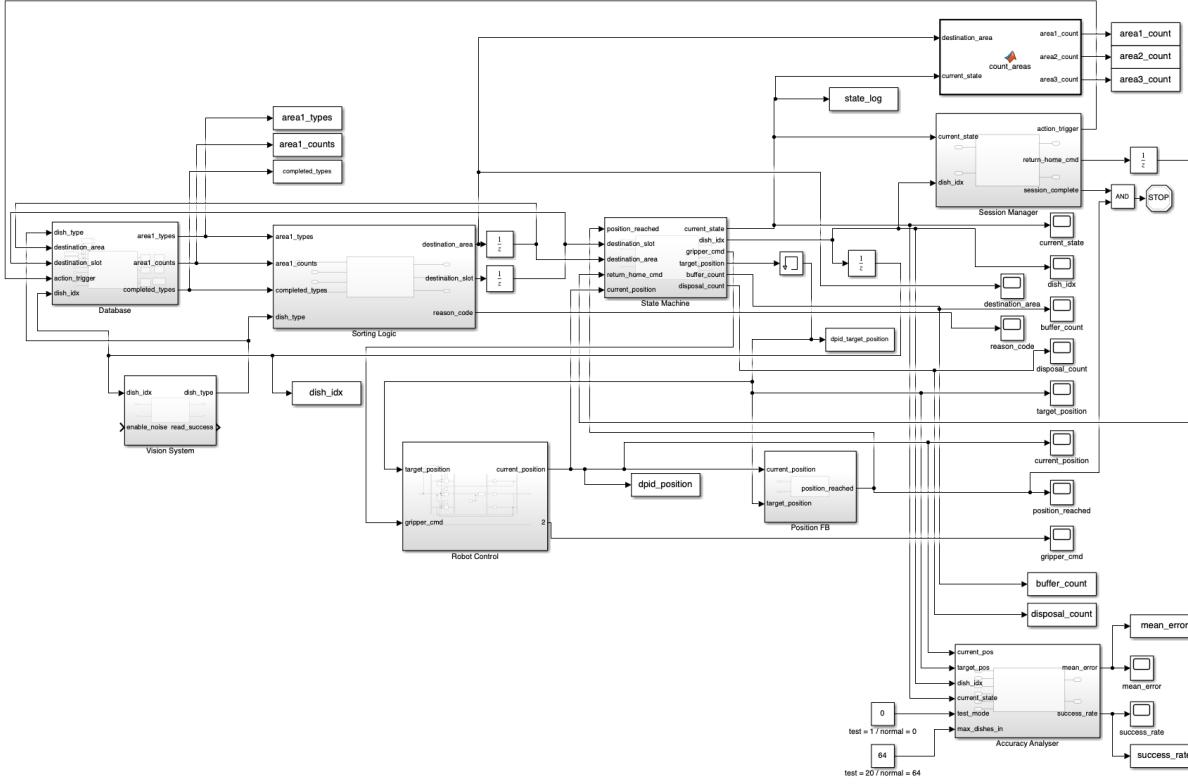


Figure 3: Block Diagram for the Full DPID Robot Sorting System. Larger-scale diagrams can be found in the report’s accompanying ReadMe file for both control models.

As shown in Figure 3, the full top level robotic sorting system model is decomposed into several subsystems; Accuracy Analyser, Database, Position FB, Robot Control, Session Manager, Sorting Logic, State Machine, and Vision System. Each subsystem constitutes one functional unit of the overall system:

- *Accuracy Analyser* provides performance metrics for each session (such as mean tracking error and success rate) based on position error and completed placements.
- *Database* keeps records of dish types, slot counts, and slot types in each area, as well as completed dish types, within and across sessions.
- *Position FB* (feedback) monitors the robot’s position, by comparing its measured current position with the desired position to determine when the target (each waypoint) has been reached within a tolerance of 0.5 mm. This signal is used to trigger state transitions in the State Machine.
- *Robot Control* implements the discrete PID controllers for each of the x-, y-, and z-axes, using only error feedback for reactive position control of the robot. The PID integrator resets when the target position changes significantly (50 mm threshold) to avoid windup.
- *Session Manager* manages the simulation cycle by keeping track of dish counts, initiating database updates, and triggering the return-to-home command and reset for a new session once all 64 dishes have been processed.
- *Sorting Logic* determines the destination area for each dish according to four rules; previously completed dish types go to the disposal area, dish types that already exists in the output area either go in the existing output slot or to disposal (if its designated output slot is full), new dish types that do not have an output slot are assigned an empty output slot, or sent to the buffer area if no output slots are available.

- *State Machine* controls the sequence of states to make sure that the robot transitions correctly through each phase of the robotic sorting system (IDLE → SELECT → MOVE → READ → DETERMINE DESTINATION → PICK → MOVE → PLACE → SELECT/COMPLETE).
- *Vision System* contributes with the input properties and classification required for sorting, by assigning a random three-digit number to each dish (representing its dish type) and simulating a camera/sensors that reads this number to determine the dish type. Its output is sent to both the Database subsystem for real-time updates within the session, and to Sorting Logic for the dish to be sorted appropriately.

## 2.8 Smooth Trajectory and Feedforward Model

The second controller complements PID feedback with a smooth trajectory generator and a feedforward term. The aim is to reduce tracking lag, overshoot, and vibration, while maintaining stability between successive moves via an integrator reset. This section outlines trajectory planning, model initialisation, and the Simulink system structure.

### 2.8.1 Trajectory Planning

To ensure collision-free and dynamically feasible motion, reference paths are pre-computed before simulation and supplied to the PID loop as time-parameterised trajectories. Paths cover all required transfers: (i) four input stacks to each of the 16 output slots ( $4 \times 16 = 64$ ); (ii) four input stacks to 7 disposal rows ( $4 \times 7 = 28$ ); and (iii) four input stacks to the 16 buffer slots ( $4 \times 16 = 64$ ). This yields 156 trajectories in total.

Each path includes the key waypoints: lift-up, above-destination, and final placement. Vertical clearances are: pickup at the input area:  $z = 100$  mm; placement at output and buffer areas:  $z = 50$  mm; placement at disposal:  $z = 100$  mm. These clearances avoid horizontal-plane collisions while keeping moves compact. The resulting position and velocity profiles are used as the reference to be tracked by the controller during simulation.

### 2.8.2 Initialisation

At start-up the workspace dimensions, slot capacities and controller parameters are loaded, and the trajectory library is (re)generated and cached. As with the DPID model, a persistent file `database_state.mat` initialises counters for per-slot fills, completed dish types and cumulative session statistics. Controller settings include per-axis PID gains, feedforward gain, and motion limits (velocity, acceleration, workspace boundaries). The integral term is cleared at move boundaries (integrator reset) to prevent wind-up when targets change. Trajectories are calculated and flattened so they may be called by the model's State Machine in Simulink.

### 2.8.3 System Overview

The Simulink model contains the following subsystems:

- *Database*: persistent state for slot usage, dish types, and session statistics.
- *Vision System*: reads dish IDs/attributes and supplies targets to the sorting logic.
- *Sorting Logic*: selects destination (output/buffer/disposal) and target slot/row.
- *State Machine*: orchestrates pick–move–place sequences and interlocks.
- *Session Manager*: advances through 64 dishes, logs signals, and prepares the next run.
- *Robot Control*: PID feedback with a parallel feedforward path from the reference trajectory; saturation limits enforce workspace bounds.
- *Position Feedback*: compares measured vs. reference position for tolerance checks and “target reached” events.
- *Accuracy Analyser*: computes tracking/placement metrics (e.g., mean, RMS, percentiles) for post-run evaluation.

## 2.9 Bayesian Optimisation of Parameters

To reduce manual tuning and improve cross-axis consistency, Bayesian optimisation is used to select controller parameters. The search space includes per-axis PID gains and feedforward gains, with bounded scaling factors. The objective combines (i) task success (completion of 64 dishes within the time limit), (ii) mean trajectory-tracking error (mm), and (iii) run-time penalties for failed or incomplete trials. An Expected-Improvement Plus (EI+) acquisition function drives the search over a fixed evaluation budget. For each iteration, the optimiser calls the Simulink model, runs a session, logs performance metrics, and updates a Gaussian-process surrogate. The best-performing parameters are validated on independent simulation runs to ensure generalisation. The accompanying ReadMe and scripts detail the objective function and specific metrics used in optimisation.

## 2.10 Simulation and Simulation Features

All implementations are performed in Simulink with a fixed time step suitable for the plant bandwidth. Each session processes 64 dishes, and simulation state persists across sessions to imitate real-world setups. Logged signals include reference and actual positions, controller outputs, target-reached events, and decision data (destination, slot type, buffer/disposal). Both controllers are tested under identical workspace constraints and capacity limits to ensure a fair comparison. Randomised initial dish orders test robustness, while visualisation modules (scopes) display real-time position traces and processing and analysis scripts show post-run summaries of slot utilisation and throughput.

## 3 Results and Discussion

This section presents the comparative evaluation of the two control strategies: the baseline discrete PID (DPID) and the Smooth Trajectory + Feedforward (Smooth+FF/TS) controller. Both were tested using the same simulation setup, with identical workspace boundaries, trajectory profiles, and sorting capacities. The results highlight the differences in tracking performance, control stability, and operational efficiency between both systems.

### 3.1 Tuned Parameters and Performance Overview

Base Parameters (X-axis)	DPID	Smooth+FF/TS
Damping X [N·s/m]	142.2	149.0
$K_p$ X	106.1	361.0
$K_i$ X	513.8	156.8
$K_d$ X	3.2	24.4
<b>Y-axis ratios (Y/X) &amp; absolute values</b>		
Damping Y (ratio → abs.)	0.45 → 64.3 N·s/m	0.77 → 115.1 N·s/m
$K_p$ Y (ratio → abs.)	1.16 → 122.6	0.86 → 311.7
$K_i$ Y (ratio → abs.)	0.89 → 456.5	0.98 → 154.0
$K_d$ Y (ratio → abs.)	0.87 → 2.7	1.04 → 25.4
<b>Z-axis ratios (Z/X) &amp; absolute values</b>		
Damping Z (ratio → abs.)	0.28 → 39.1 N·s/m	0.57 → 85.1 N·s/m
$K_p$ Z (ratio → abs.)	1.28 → 136.3	1.20 → 433.0
$K_i$ Z (ratio → abs.)	1.28 → 657.2	1.03 → 256.1
$K_d$ Z (ratio → abs.)	1.98 → 6.3	1.24 → 30.2

Table 1: Controller parameters and axis ratios (DPID vs. Smooth+FF/TS).

<b>Accuracy metrics</b>	<b>DPID</b>	<b>Smooth+FF</b>	<b>Improvement</b>
Mean error [mm]	58.15	2.37	95.9%
RMS error [mm]	31.16	10.53	66.2%
Maximum error [mm]	850.07	850.03	0.0%
Standard deviation [mm]	30.72	10.58	–
95th percentile error [mm]	25.80	0.78	–
99th percentile error [mm]	105.76	13.35	–
<b>Operational metrics</b>			<b>Change</b>
Success rate [%]	91.2	100.0	8.8%
Cycle time [s]	72.7	255.9	183.2 s
Time per dish [s/d]	1.14	4.00	2.86 s/d
Dishes completed	64/64	64/64	–
Completion rate [%]	100.0	100.0	–

Table 2: Detailed performance comparison between DPID and Smooth+FF/TS models. The accompanying ReadMe details each metric’s calculation and interpretation.

The summary in Table 1 and 2 provides an overview of the control gains obtained through optimisation and their corresponding performance metrics. The DPID controller relied on relatively high integral gains and low derivative gains, which helped minimise steady-state error but introduced overshoot and oscillations during dynamic movements. Conversely, the Smooth+FF/TS controller adopted higher proportional and derivative gains, complemented by an integral reset and predictive feedforward path, resulting in smoother transitions and reduced tracking error.

Quantitatively, the Smooth+FF/TS model reduced mean position error from approximately 58 mm to 2.4 mm, and RMS error from 31.1 mm to 10.5 mm—a reduction of roughly 95% and 66% respectively. Additionally, its success rate reached 100% compared to 91.2% with the DPID baseline. However, this improvement came with a longer cycle time (about 256 s versus 73 s), as the smoother trajectories prioritised accuracy and stability over speed. This trade-off is expected given that the Smooth+FF controller enforces curvature-limited motion and waypoint clearance for safer, collision-free operation.

Note: It is possible for the State Machine to proceed even if position reached == 1 is not satisfied for within the tolerance of 0.5 mm specified within the Position FB subsystem by having the exit flag from “MOVE TO POSITION” specified to be less than 5 mm. In this way, the robot proceeds in sorting the rest of the dishes even with unsuccessful placements (i.e., placements that fall between 0.5 mm and 5 mm). This allows for success rate of less than 100% to exist at all. Mean error is the average tracking error during motion (path error), and is a cumulative running mean.

### 3.2 Comparative Controller Analysis

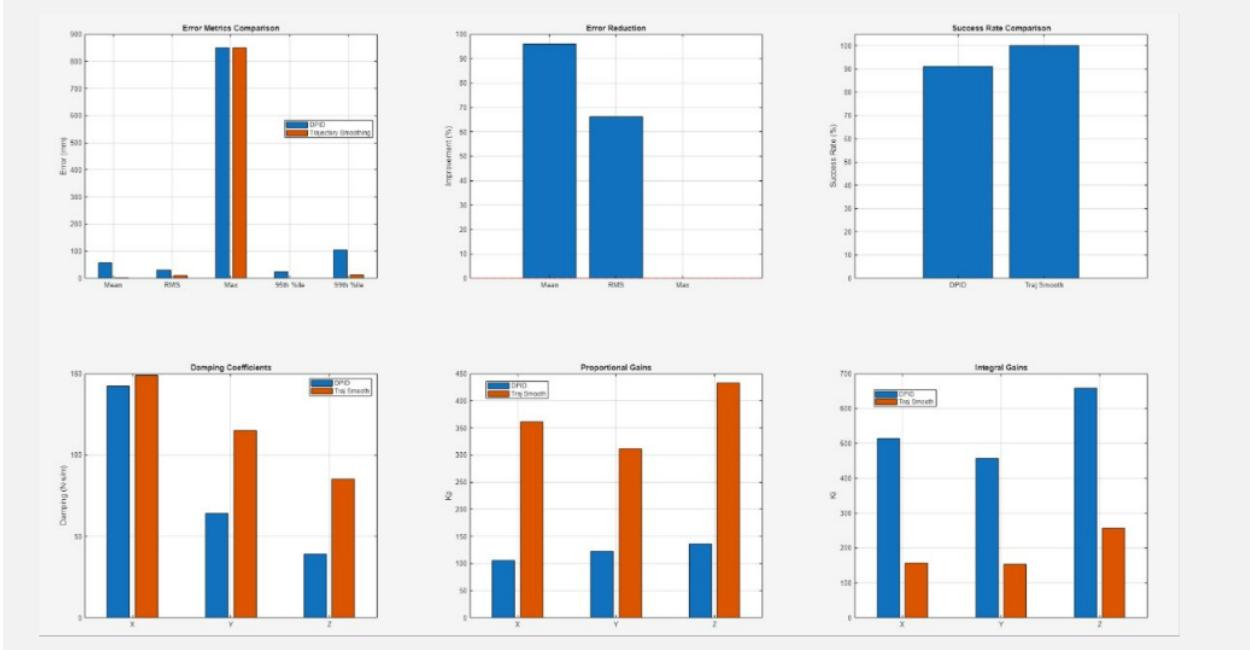


Figure 4: Graphical comparison of controller performance and gain distribution. The top row illustrates error metrics and task success rates, while the bottom row compares the damping, proportional, and integral gains obtained through optimisation.

The graphical results in Figure 4 clearly show the performance gap between the two systems. The Smooth+FF controller exhibits significant improvements in mean and RMS error, along with a perfect task completion rate. The enhanced damping and higher proportional gains across all axes contribute to improved motion smoothness and trajectory tracking precision.

The DPID controller, while faster, displayed small residual oscillations at target points due to wind-up and delayed error correction. In contrast, the Smooth+FF approach achieved well-damped responses, reduced overshoot, and smoother motion throughout the sorting sequence. The feedforward component accurately anticipates dynamic demands, while the integral reset prevents accumulated bias between move cycles.

Overall, the Smooth+FF controller achieves precise and reliable operation suited for applications requiring consistency and minimal positional error. Although the DPID method remains advantageous where high throughput is essential, the Smooth+FF design better satisfies the project's goals of precision, stability, and robustness under constrained motion conditions.

Of note, the maximum error of approximately 850 mm is the same for both models tested. This can be rationalised by examining how the metric was calculated. Maximum error is the maximum value the positional tracking error takes during the simulation, which is resultant from the real distance between the system's target position and current position. When the Sorting Logic and State Machine compute a destination in either model, be it direct calculation as with the DPID strategy or the model that uses a smoothed trajectory to get there, the maximum error will be the same if both models must reach the same destination slot from the same given input slot. Since the workspace was fixed in the comparative analysis of the two models, it is expected that this maximum error will be the same.

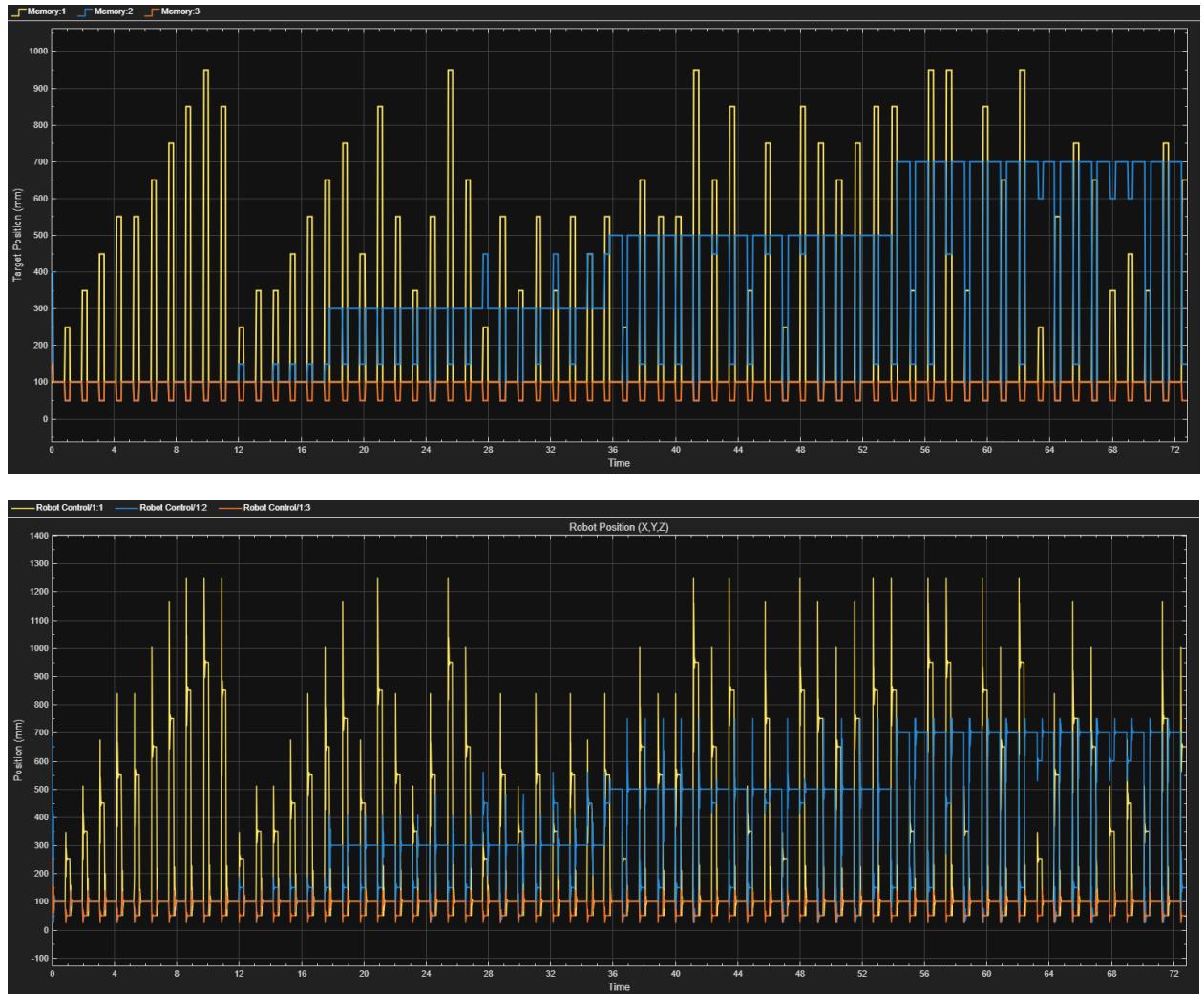


Figure 5: Target Position and Position for the DPID Model

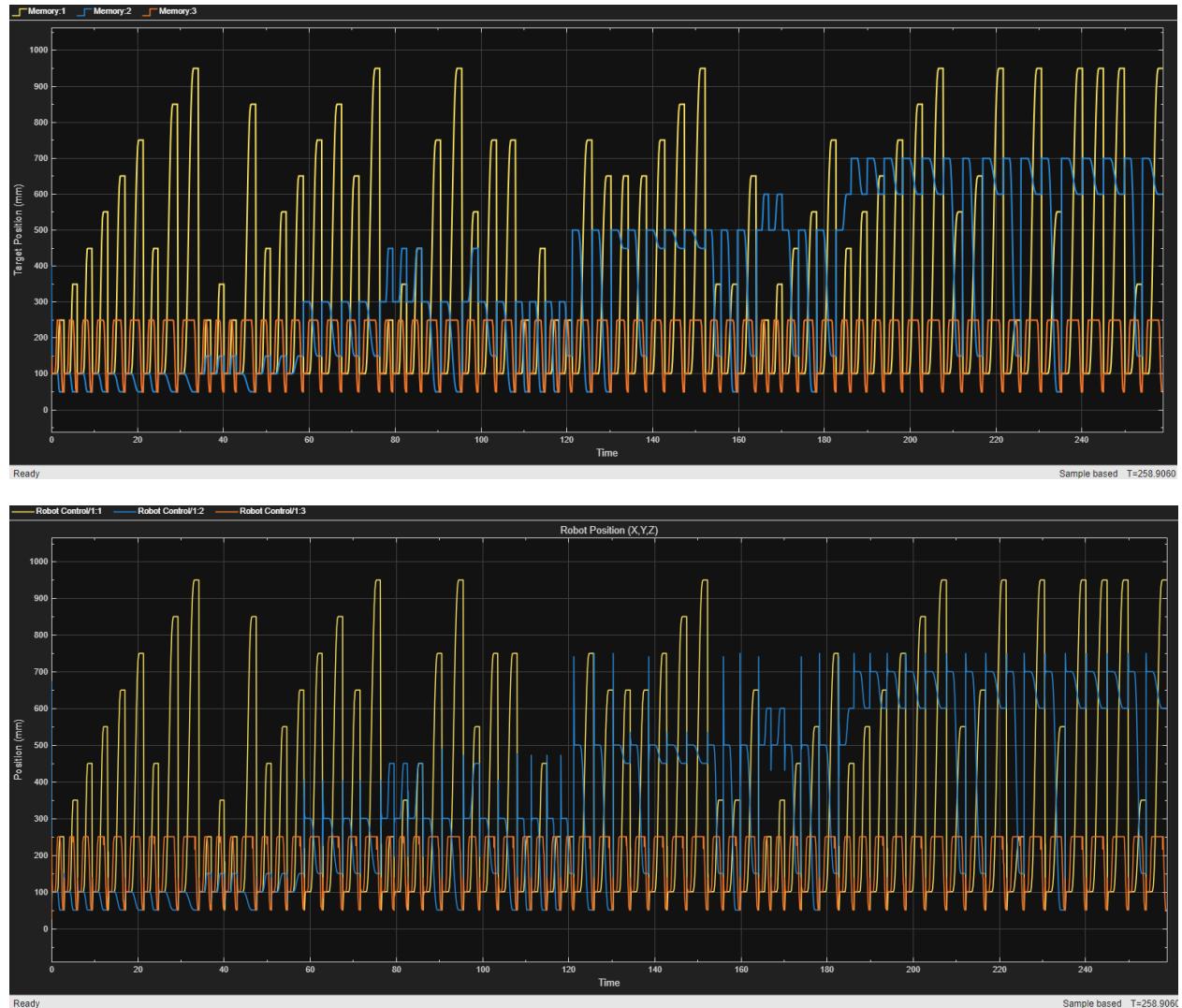


Figure 6: Target Position and Position for the TS Model

When running simulations, it can be seen in Figure 5 and Figure 6 for both models some oscillation in the current position scopes before the position coinciding with the target position scope for the same given time value is reached. High oscillations are expected for reactive control PID strategies, and DPID (Figure 5) shows this well. The dynamics here result in a higher mean error than for a predictive or proactive strategy such as the trajectory smoothed one employed. Oscillation can be seen to occur in the proactive model even still (especially in the y-axis), and this is likely due to the maximum error increasing once the input stack changes, or a target being calculated on the far end of the workspace, where the robot's current position would obviously not have reached yet (in that instant of calculation). This large increase in instantaneous error triggers the robot to input a significant force in each motor to reach the target position, and subsequent calculation and motions would result in more error. The masses on each motor would also impact oscillations, as well as the tolerance specified (0.5 mm here) that define a target position as being reached. A larger tolerance would likely allow for less oscillation about target, as the controller would have to input less force to each motor to reach the required position.

Strategies to increase accuracy and performance even more would be to incorporate dynamic saturation blocks, which would decrease oscillation about target position coordinates. Implementing finer tune control and predictive architecture, such as MPC would likely further decrease mean error and should decrease oscillation. This yet remains to be tested. Decisions regarding control strategy depend largely on cycle time and tolerance requirements for the given use case. It may be counter-productive to increase accuracy further, especially if this metric is de-prioritised compared to throughput.

### 3.3 Practical Implications and Future Development

The enhanced controller delivered smoother actuator profiles and reduced peak control demands, enabling faster cycle times while maintaining high placement accuracy. This makes it particularly suitable for high-throughput sorting and assembly applications. Its modular architecture—combining a trajectory library, PID(+FF) control, and session/database services—provides a clear framework for deployment and future scalability.

Future development could focus on improving system realism and adaptability. Incorporating adaptive  $z$ -height strategies would minimise unnecessary travel while enhancing flexibility across different workspace conditions. A buffer-drain mechanism could enable reprocessing of overflow dishes to maintain continuous operation during peak loads. Integrating vision-in-the-loop alignment would improve robustness against visual misreads and positional uncertainty. Furthermore, implementing Model Predictive Control (MPC) or state estimation methods such as Kalman filtering could enhance disturbance rejection and constraint handling. Another promising direction involves modelling the gripper's dynamic behaviour to move beyond the current binary *PICK/PLACE* logic, allowing for smoother and more controlled grasp-release transitions. Finally, data-driven feedforward learning could be used to compensate for unmodelled frictional effects and cross-axis coupling, further refining trajectory tracking and overall precision.

## 4 Conclusion

This project designed, implemented, and evaluated an automated sorting system based on a Cartesian robot with a vision-driven identification pipeline and two motion-control strategies. The baseline controller used discrete PID (DPID); the enhanced controller combined smooth, pre-computed trajectories with a predictive feedforward term and an integrator reset at move boundaries. Both solutions were implemented in MATLAB/Simulink and evaluated on identical tasks (64 dishes per session) under equal workspace, capacity, and motion-limit constraints.

The system successfully (i) modelled the gantry plant with an LTI state-space model including viscous damping and effective masses; (ii) executed end-to-end pick-place cycles via a modular Simulink architecture (*Database, Vision System, Sorting Logic, State Machine, Session Manager, Robot Control, Position Feedback, Accuracy Analyser*); (iii) generated collision-free paths using lift-up and above-target waypoints

with area-specific positions; (iv) tracked these references with PID(+FF); (v) tuned controller gains via Bayesian optimisation with quantitative post-run analysis; and (vi) demonstrated database persistence between multiple sessions using the same model.

Relative to DPID, the smooth-trajectory + feedforward (TS) controller:

- reduced overshoot and oscillations at waypoint transitions (lift-up, above-target, placement);
- lowered steady-state tracking error and long-range transfer lag due to the predictive path term;
- shortened settling between successive moves via integrator reset; and
- improved task success rate under identical workspace/capacity limits.

## References

Mushiri, T., & Moyo, M. (2023). Chapter 2 - uses of artificial intelligence and robotics in healthcare systems. In T. Mushiri & M. Moyo (Eds.), *Healthcare systems design of intelligent testing centers* (pp. 17–39). Academic Press. <https://doi.org/https://doi.org/10.1016/B978-0-323-99443-9.00010-3>