Project Report: Inverted Pendulum

COMP0216 - Systems Engineering for Real-time Systems

Spring 2025

Group Members: Daniel Helmi, Muhammad Maaz, Arjun Bhaduri, Rehan Agrawal

University College London

April 4, 2025

Contents

1 Introduction **3**

2 Project Planning **3**

2.1 Mind Map . . . 3
2.2 Work Breakdown Structure (WBS) . . . 3
2.3 Risk Breakdown Structure (RBS) . . . 3
2.4 A-B-M Estimates, Expected Time, and Standard Deviation . . . 3
2.5 Slack . . . 4
2.6 Project Network Diagram . . . 5
2.7 Critical Path and Completion Time (CP, CT) . . . 5
2.8 Gantt Chart . . . 5
2.9 RACI Matrix . . . 6
2.10 Probability of Completion . . . 6

3 Risk Management **6**

3.1 Failure Modes and Effects Analysis (FMEA) . . . 6
3.2 Redundancies and Compensatory Measures . . . 6
3.3 Ishikawa (Fishbone) Diagram . . . 7
3.4 Power–Interest Grid . . . 7
3.5 Risk Matrix . . . 8
3.6 Security Threats, Vulnerabilities, and Mitigation . . . 8
3.7 Applied Lean Methods . . . 9

4 System Requirement Specification **9**

4.1 Dynamics and Kinematics Requirements Consistency . . . 9
4.2 Logical Requirements Consistency . . . 9
4.3 Mealy Machine . . . 10
4.4 Model-Based System Engineering (MBSE) . . . 10
4.5 Reliability and Failure Time Metrics . . . 11

5 Modelling and Simulation **12**

5.1 Equations of Motion . . . 12
5.1.1 Nonlinear Dynamics . . . 12
5.1.2 State-Space Representation . . . 13
5.1.3 Linearized Model . . . 13
5.2 Controller Design and Implementation . . . 13
5.2.1 PID Controller . . . 13
5.2.2 Pole Placement Controller . . . 14

5.2.3 Nonlinear Controller 14
5.3 Python Implementation Details 14
5.3.1 Pendulum Physics Module 14
5.3.2 Sensor Noise and Filtering 14
5.4 Results and Benchmarking 15

6 Prototyping **16**
6.1 Electrical Diagram (Schematics) 16
6.2 Mechanical (CAD) Design 18
6.3 Controller Design Approaches 19
6.3.1 System Modeling and Linearization 19
6.3.2 PID Controller 19
6.3.3 Pole Placement Controller 20
6.3.4 LQR Controller 20

7 Overview **20**
7.1 Benchmarking and Results 20
7.1.1 LQR Controller 20
7.1.2 Pole Placement 21
7.1.3 PID Analysis 24
7.1.4 PID_with_failure 25
7.1.5 Failure Metrics 26
7.1.6 Overall Controller Performance Comparison 28

8 Conclusion **28**

9 References **29**

10 Appendix **30**
10.1 Ethical Considerations 30
10.2 Figures and Code Snippets 30
10.3 Planning and Results Video Link or Screenshot 36
10.4 Code Submissions 36

1 Introduction

This project creates an inverted pendulum with the aim of testing different control strategies. System behaviour analysis uses state-space representation, non-linear dynamics, and linearised models. Python implemented PID, LQR, and non-linear controllers. Real-world performance improves with sensor noise and filtering. Electrical schematics, CAD, and PCB layouts construct and test the system.

We were inspired by online inverted pendulum systems. University of Michigan Control Tutorials for MATLAB and Simulink model the inverted pendulum. [1].

2 Project Planning

2.1 Mind Map

The mind map, found in the appendix (Fig. 46).

2.2 Work Breakdown Structure (WBS)

A figure of the WBS can be found in the appendix (Fig. 47)

2.3 Risk Breakdown Structure (RBS)

The Risk Breakdown Structure can be found in the Appendix (Tab. 12)

2.4 A-B-M Estimates, Expected Time, and Standard Deviation

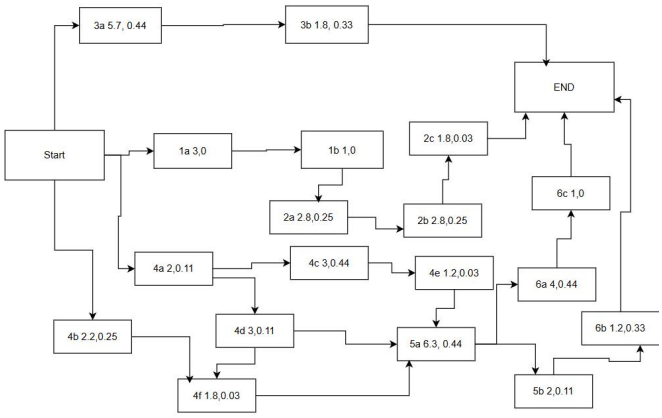

Figure 1: Activity Chart Showing Task Sequences, Expected Times, and Variance

Example Path:

- A sample path: `Start → 1a → 1b → 2a → 2b → 4c → 4e → 5a → 6a → 6c → END`
- This sequence includes high-variance tasks such as `5a (6.3 days, 0.44)` and `6a (4 days, 0.44)`, which are likely candidates for the critical path.

Activity	Optimistic Time	Most likely time	Pessimistic Time	Expected Time (1dp)	Variance (2dp)	Standard Deviation (2dp)	Immediate Predecessor
1A	3	3	3	3	0	0	
B	1	1	1	1	0	0	1a
2A	1	3	4	2.8	0.25	0.5	1a,b
B	1	3	4	2.8	0.25	0.5	2a
C	1	2	2	1.8	0.03	0.16	2a,b
3A	3	6	7	5.7	0.44	0.67	
B	1	2	2	1.8	0.03	0.16	3a
4A	1	2	3	2	0.11	0.33	
B	1	2	4	13	0.25	0.5	
C	1	3	5	3	0.44	0.67	4a
D	2	3	4	3	0.11	0.33	4a
E	1	1	2	1.2	0.03	0.16	4c
F	1	2	2	1.8	0.03	0.16	4a-e
5A	5	6	9	6.3	0.44	0.67	4a-e
B	1	2	3	2	0.11	0.33	5a
6A	2	4	6	4	0.44	0.67	5a
B	1	1	2	1.2	0.03	0.16	5b
C	1	1	1	1	0	0	5a

Figure 2: Time Estimates for Project Activities Using A–B–M and PERT

Examples:

- Activities like **1A, B, and 1** have no uncertainty (zero variance), indicating deterministic durations.
- Activities like **3A, 4C, and 5A** have higher standard deviations, meaning they introduce more schedule risk.
- The "Immediate Predecessor" column identifies which activities must be completed before the current one can start, helping define the dependency graph and calculate the project's overall duration.

2.5 Slack

Activity	Duration	Earliest Start (ES)	Earliest Finish (EF)	Latest Start (LS)	Latest Finish (LF)	Float (Slack)
1a	3	0	3	0	3	0
1b	1	3	4	3	4	0
2a	2.8	4	6.8	4	6.8	0
2b	2.8	4	6.8	4	6.8	0
2c	1.8	4	5.8	6.8	8.6	2.8
3a	5.7	0	5.7	5.8	11.5	5.8
3b	1.8	5.7	7.5	11.5	13.3	5.8
4a	2	0	2	6.8	8.8	6.8
4b	2.2	0	2.2	6.8	9	6.8
4c	3	6.8	9.8	6.8	9.8	0
4d	3	9.8	12.8	9.8	12.8	0
4e	1.2	6.8	8	8.8	10	2
4f	1.8	12.8	14.6	12.8	14.6	0
5a	6.3	12.8	19.1	12.8	19.1	0
5b	2	8	10	10	12	2
6a	4	19.1	23.1	19.1	23.1	0
6b	1.2	10	11.2	12	13.2	2
6c	1	23.1	24.1	23.1	24.1	0

Figure 3: List of Activities with Associated Slack

Critical Path and Completion Time from Slack

The critical path can also be derived using the slack values in this table. As shown, the path: `1a → 1b → 2a → 4c → 4d → 4f → 5a → 6a → 6c` consists entirely of tasks with zero slack, confirming it as the critical path. The total completion time along this path is **24.1 days**, which aligns with the earlier analysis.

2.6 Project Network Diagram

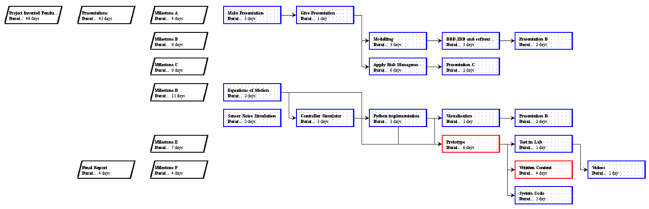

Figure 4: Network Diagram Representing Task Dependencies and Critical Path

Critical Path Insight:

- *Sensor Noise Simulation* → *Controller Simulator* → *Python Implementation* → *Prototype* → *Test in Lab* → *Written Content*

2.7 Critical Path and Completion Time (CP, CT)

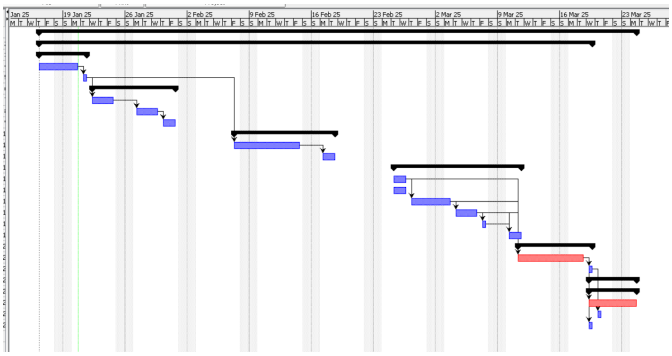

Figure 5: Gantt Chart Showing Critical Path (Red) and Completion Time

Gantt Chart Timeline:
The figure shows a Gantt chart for the Inverted Pendulum project. It visually presents scheduled tasks along a timeline from **January 16, 2025** to **March 24, 2025**, covering a total of **48 working days**.
Tasks on the critical path are highlighted in red. Their alignment indicates dependencies and the sequence of operations that directly affect project duration.

2.8 Gantt Chart

A figure of the Gantt Chart can be found in the appendix ((Fig. 48))
Key Notes:

- The project spans from **January 16, 2025** to **March 24, 2025**, totaling **48 working days**.
- Tasks are grouped under major milestones (Milestone A to F), each associated with key phases such as modelling, risk management, implementation, and reporting.
- Dependencies ensure that tasks follow a logical flow—for example, "Give Presentation" (Task 4) cannot begin until "Make Presentation" (Task 3) is completed.

- Later phases such as *Python Implementation*, *Prototyping*, and *Written Content* are on the critical path and must be tightly monitored.

2.9 RACI Matrix

	A	B	C	D	E	F
1	Activity	Task	Arjun (A)	Daniel (D)	Rehan (R)	Maaz (M)
2	M.A	1.a	I	R	I	I
3	M.A	1.b	A	R	C	C
4	M.B	2.a	R	I	A	R
5	M.B	2.b	I	I	C	R
6	M.B	2.c	I	R	C	C
7	M.C	3.a	R	C	I	I
8	M.C	3.b	R	R	C	C
9	M.D	4.a	A	I	A	R
10	M.D	4.b	I	C	C	R
11	M.D	4.c	I	I	C	R
12	M.D	4.d	I	I	R	A
13	M.D	4.e	I	I	R	R
14	M.D	4.f	R	R	C	C
15	M.E	5.a	I	I	R	C
16	M.E	5.b	I	I	I	I
17	M.F	6.a	R	R	C	C
18	M.F	6.b	C	R	C	C
19	M.F	6.c	I	I	A	R
20						

Figure 6: RACI Matrix for Project Team Responsibilities

Explanation:
Each row in the matrix corresponds to a project task (e.g., "1.a", "2.b", etc.) associated with one of the project's milestones (M.A–M.F). Each team member is assigned a role — **R** (Responsible), **A** (Accountable), **C** (Consulted), or **I** (Informed) — based on their involvement in that specific task.

For example, in Task 1.a (Make Presentation), Arjun is marked as "I", Daniel as "R", and Rehan as "I", indicating Daniel is actively working on the task while others are kept informed.

2.10 Probability of Completion

There are 48 days before the deadline. To determine the probability of completing the project within this time, we use the Z-score formula: $Z = \frac{X-\mu}{\sigma}$, where σ is the square root of the sum of variances of critical path activities. The variance for each activity is calculated as $\left(\frac{\text{Pessimistic - Optimistic}}{6}\right)^2$. Summing the variances along the critical path gives $\sigma^2 = 0.6563$, so $\sigma = \sqrt{0.6563} \approx 0.81$. Substituting into the Z-score formula: $Z = \frac{48-24.1}{0.81} = \frac{23.9}{0.81} \approx 29.51$. The probability $P(Z \leq 29.51)$ is approximately 1.00000, meaning the project is certain to be completed within 48 days.

3 Risk Management

3.1 Failure Modes and Effects Analysis (FMEA)

The FMEA Analysis form can be found in the appendix (Tab. 13)

3.2 Redundancies and Compensatory Measures

- **Sensor Redundancy:** The system was designed to support multiple encoders for measuring pendulum and motor positions. This redundancy ensures continued

feedback even if one sensor fails or generates noisy data.

- **Fail-safe Motor Control:** In the event of instability or a detected fault (e.g., excessive pendulum deviation), the Arduino sends a zero motor command, effectively acting as an emergency stop. This prevents runaway behavior and hardware damage.
- **Thermal Compensation:** Component layout and power regulation were tested to avoid overheating. Passive cooling, proper spacing, and sufficient current ratings help mitigate thermal risks.

3.3 Ishikawa (Fishbone) Diagram

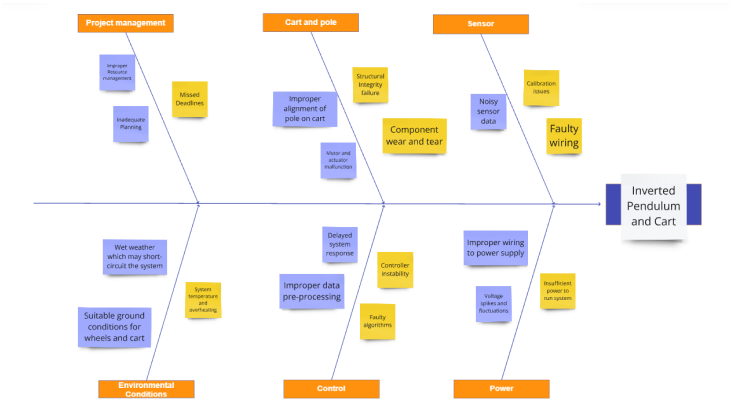

Figure 7: Fishbone Diagram for the Inverted Pendulum and Cart System

This diagram organizes potential causes into six major categories, each branching into specific sub-causes.

3.4 Power–Interest Grid

Power \ Interest	Low	High
High	**Keep Satisfied** • Innovation Lab Technicians • UCL Department / Faculty	**Closely Manage** • Module Leader (Dr. Saeedi) • Project Team
Low	**Monitor** • General Public	**Keep Informed** • Teaching Assistants

3.5 Risk Matrix

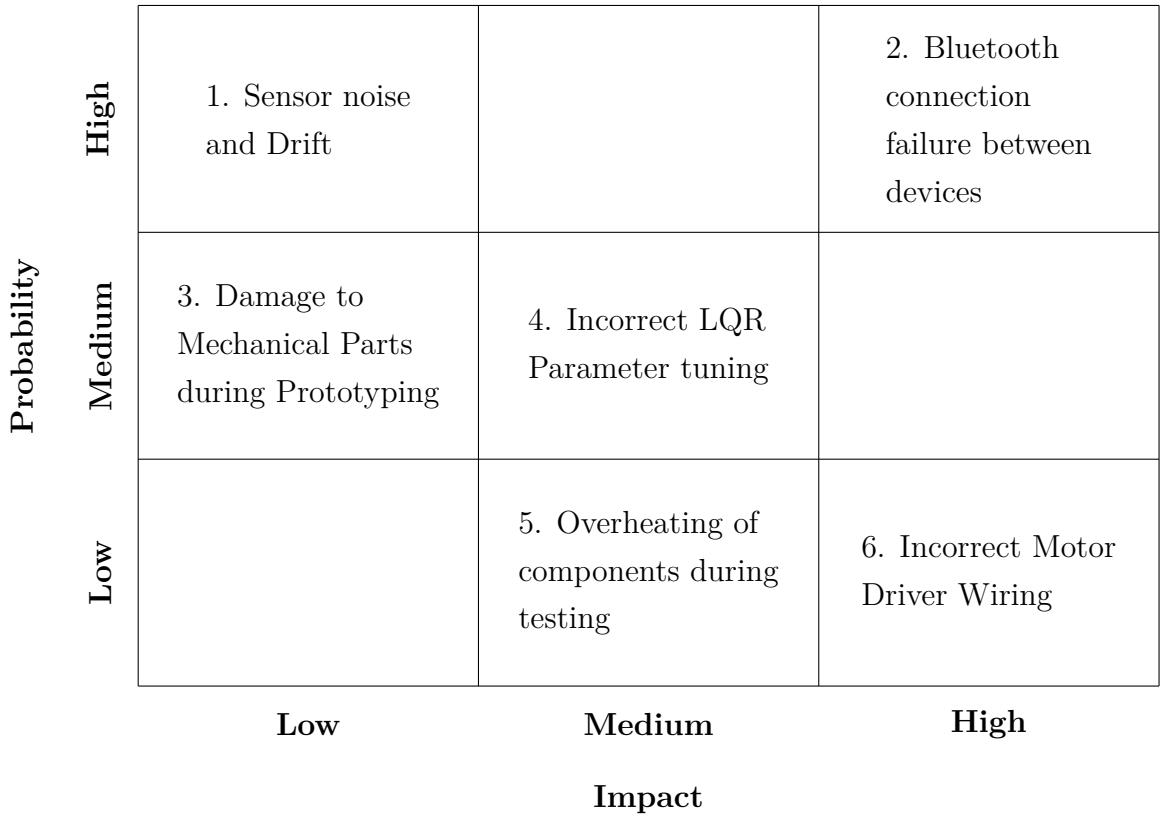

Figure 8: Risk Matrix: Probability vs. Impact

3.6 Security Threats, Vulnerabilities, and Mitigation

Threat	**Vulnerability**	**Mitigation Strategy**
Unauthorized Access to Control Signals	Unencrypted I^2C communication between Arduino and ESP32	Implement hardware-based authentication or use checksum verification to detect altered messages
Bluetooth Hijacking (if active)	Open pairing on ESP32 Bluetooth module	Disable Bluetooth if unused, or implement secure pairing with PIN authentication
Power Supply Tampering	Exposed power connections and unprotected regulator input	Add protective casing and fuse circuitry to prevent external damage or tampering

Table 1: Security Threats, Vulnerabilities, and Mitigation Strategies

3.7 Applied Lean Methods

1. Continuous Improvement (Kaizen):

- Weekly reflection meetings were held to assess what worked and what could be improved, iteratively refining both design and workflow.
- Feedback from each project milestone presentation was used to improve future deliverables.

2. Value Stream Mapping:

- All tasks were aligned with the goal of delivering a fully functional, stable inverted pendulum system.
- Activities such as unnecessary simulations or duplicate documentation were deprioritized to focus on high-value work (e.g., system testing, real-time control).

4 System Requirement Specification

4.1 Dynamics and Kinematics Requirements Consistency

- **Mathematical Modelling:** A state-space model was obtained by deriving the system dynamics from Newtonian mechanics and linearising around the upright equilibrium point. This model describes in both the cart and the pendulum the link among forces, accelerations, and angular displacements.
- **Control-Oriented Simulation:** Closed-loop responses under LQR control were simulated in Python using the same linear model. Small oscillations damped out as the system stabilised matched expected system behaviour in the simulated trajectory of the pendulum angle and cart location.

4.2 Logical Requirements Consistency

- **Functional Consistency:** From modelling and control design to physical integration on the PCB and software, every fundamental functional need—such as pendulum stabilisation, real-time sensor feedback, and motor actuation—is supported by a clear implementation route.
- **Control Loop Integrity:** The control loop first uses deterministic actuation then assumes timely and accurate sensor data. Verified for consistency and coherence is the logical flow from measurement (encoder readings) to decision-making (Arduino-based LQR control) to actuation (motor driver signals via ESP32).

4.3 Mealy Machine

- **States:** The system operates in several discrete states, such as:
  - *Idle:* Waiting for initialization signal.
  - *Calibrating:* Zeroing sensors, checking encoder values.
  - *Stabilizing:* Actively applying LQR control to balance the pendulum.
  - *Override:* Responding to manual intervention or predefined input triggers.
  - *Error:* Entered when sensor values are out of bounds or communication fails.
- **Inputs:**
  - Encoder readings (pendulum angle, cart position)
  - Manual override signals (e.g., UART input from Serial Monitor)
  - Initialization flags and sensor calibration results
- **Outputs:**
  - Motor command sent via PWM to the driver (based on LQR)
  - Status LEDs or serial messages for debugging and state identification
  - I^2C responses (sensor values) if requested by master
- **Transitions:** The system transitions between states based on real-time conditions, such as successful calibration or sensor failures. For example, the system transitions from *Calibrating* to *Stabilizing* once zero-position is verified, or from *Stabilizing* to *Error* if angle deviation exceeds threshold.

4.4 Model-Based System Engineering (MBSE)

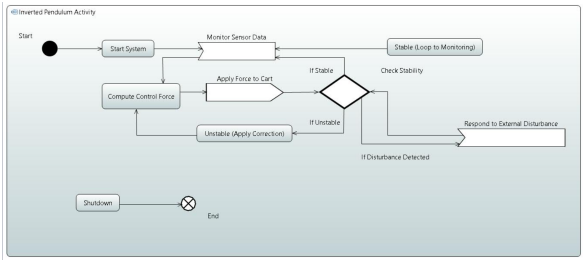

Figure 9: Eclipse Papyrus Activity Diagram

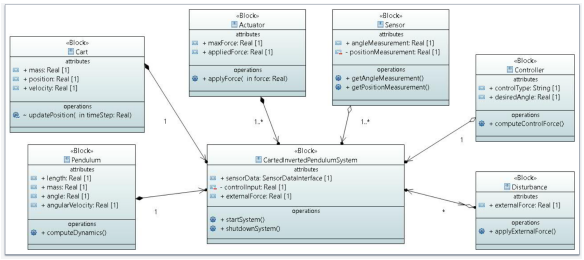

Figure 10: Eclipse Papyrus Block Definition Diagram

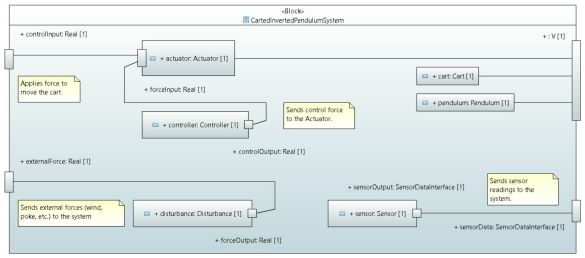

Figure 11: Eclipse Papyrus Internal Block Diagram

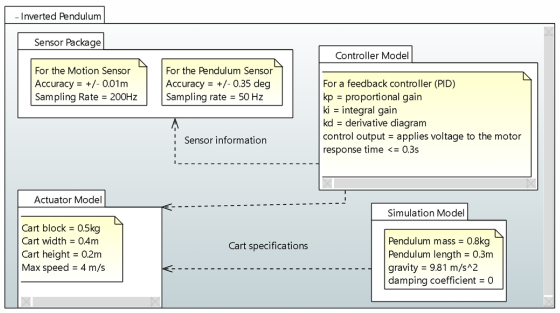

Figure 12: Eclipse Papyrus Package Diagram

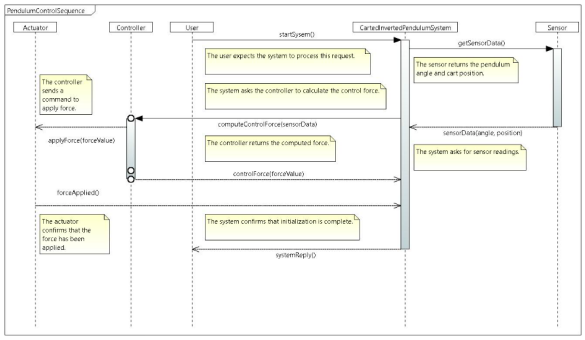

Figure 13: Eclipse Papyrus Sequence Diagram

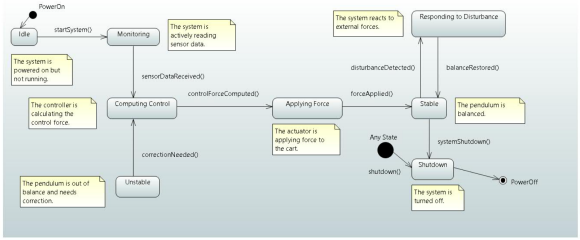

Figure 14: Eclipse Papyrus State Machine Diagram

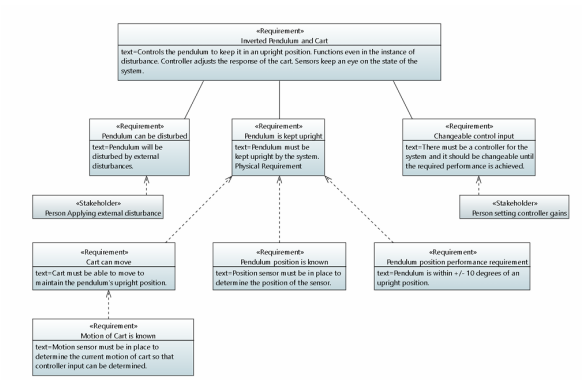

Figure 15: Eclipse Papyrus Requirements Diagram

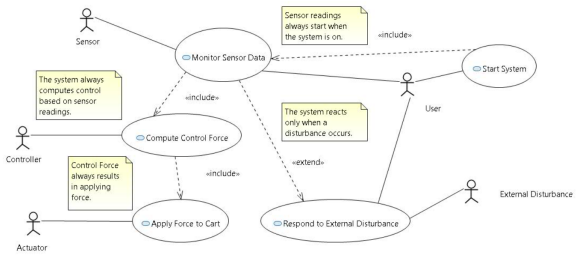

Figure 16: Eclipse Papyrus Use Case Diagram

4.5 Reliability and Failure Time Metrics

1. **Observed Reliability During Testing:**

- We conducted multiple 20-second test runs using each controller (LQR, Pole Placement, PID) to evaluate system stability.
- Both the ESP32 and Arduino consistently returned valid encoder data, and no significant drift or sensor dropout was observed.

2. Failure Conditions and Response:

- We defined a failure condition as any instance where the pendulum angle exceeded $\pm 15^\circ$ or the cart position went beyond $\pm 0.5\,\text{m}$.
- The failsafe mode halted motor commands to protect hardware and prompted the system to log diagnostic data for post-analysis.

5 Modelling and Simulation

In our simulation, the following measured hardware parameters were used:

- Cart mass (M): 1.081 kg
- Pendulum mass (m): 0.093 kg
- Pendulum length (l): 0.75 m
- Gravitational acceleration (g): 9.81 m/s^2

5.1 Equations of Motion

5.1.1 Nonlinear Dynamics

The full nonlinear model, which includes contributions from inertial coupling, gravitational forces, and damping effects, is given by:

$$(M+m)\ddot{x} - ml\ddot{\theta}\cos\theta + ml\dot{\theta}^2\sin\theta = F - b\dot{x}, \tag{1}$$

$$-ml\ddot{x}\cos\theta + ml^2\ddot{\theta} - mgl\sin\theta = -b_d\dot{\theta}, \tag{2}$$

where:

- F is the external force applied to the cart,
- b represents the damping in cart motion,
- b_d is the damping at the pendulum pivot,
- $\dot{x}$ and $\ddot{x}$ are the cart velocity and acceleration, and
- $\dot{\theta}$ and $\ddot{\theta}$ are the angular velocity and acceleration of the pendulum.

5.1.2 State-Space Representation

By defining the state vector as:

$$\mathbf{s} = \begin{bmatrix} x & \dot{x} & \theta & \dot{\theta} \end{bmatrix}^T,$$

the accelerations can be expressed as functions of the state and the control input. The resulting nonlinear state-space model is:

$$\ddot{x} = \frac{F + ml\left(\dot{\theta}^2 \sin\theta - \ddot{\theta}\cos\theta\right) - b\dot{x}}{M+m}, \tag{3}$$

$$\ddot{\theta} = \frac{(M+m)g\sin\theta + \cos\theta\left(F - b\dot{x} + ml\dot{\theta}^2\sin\theta\right) - b_d\dot{\theta}}{ml\left(\frac{M+m}{m} - \cos^2\theta\right)}. \tag{4}$$

5.1.3 Linearized Model

For controller design, the system is linearized about the unstable equilibrium ($\theta = 0$, $\dot{\theta} = 0$, $\dot{x} = 0$) using the small-angle approximations $\sin\theta \approx \theta$ and $\cos\theta \approx 1$. The resulting linearized state-space model is:

$$\dot{\mathbf{x}} = A\mathbf{x} + Bu, \tag{5}$$

with

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & -\frac{b}{M} & \frac{mg}{M} & 0 \\ 0 & 0 & 0 & 1 \\ 0 & -\frac{b}{Ml} & \frac{(M+m)g}{Ml} & -\frac{b_d}{ml^2} \end{bmatrix}, \quad B = \begin{bmatrix} 0 \\ \frac{1}{M} \\ 0 \\ \frac{1}{Ml} \end{bmatrix}. \tag{6}$$

For simplified controller design on the pendulum's angular dynamics, a reduced second-order model is also derived:

$$A_2 = \begin{bmatrix} 0 & 1 \\ \frac{g}{l} & 0 \end{bmatrix}, \quad B_2 = \begin{bmatrix} 0 \\ \frac{1}{ml^2} \end{bmatrix}. \tag{7}$$

5.2 Controller Design and Implementation

5.2.1 PID Controller

The PID controller regulates the pendulum angle by combining proportional, integral, and derivative actions:

$$u(t) = -\left(K_p\theta + K_i\int_0^t \theta(\tau)d\tau + K_d\frac{d\theta}{dt}\right). \tag{8}$$

Enhancements such as a deadzone for small errors, integrator anti-windup, position bias compensation, and control scaling near equilibrium are added. Through iterative testing,

the controller gains were tuned to $K_p = 1.2$, $K_i = 1.63$, and $K_d = 0.22$.

5.2.2 Pole Placement Controller

Using the linearized model, the pole placement controller applies full-state feedback on the reduced state vector $\mathbf{x}_\theta = [\theta\ \dot{\theta}]^\mathsf{T}$:

$$u = -K\mathbf{x}_\theta = -k_1\theta - k_2\dot{\theta}. \tag{9}$$

The gain vector $K = [k_1, k_2]$ is computed so that the closed-loop poles are placed at $p_1 = -3$ and $p_2 = -4$, ensuring a rapid yet stable response.

5.2.3 Nonlinear Controller

A nonlinear controller is implemented to handle large-angle deviations. It is defined by:

$$u = k_2\Big(E_{\text{desired}} - E_{\text{current}}\Big)\,\text{sign}(\dot{\theta}\cos\theta), \tag{10}$$

where

- $E_{\text{desired}} = mgl$ is the target potential energy at the upright position,
- $E_{\text{current}} = \frac{1}{2}ml^2\dot{\theta}^2 + mgl(1 - \cos\theta)$ represents the current mechanical energy, and
- k_2 is a tuning parameter set to 1.0.

5.3 Python Implementation Details

5.3.1 Pendulum Physics Module

The inverted pendulum's physical behaviour is captured by integrating the nonlinear differential equations using a 4th-order Runge-Kutta method. (Fig. 50)

5.3.2 Sensor Noise and Filtering

Gaussian noise is added to the simulated sensor readings to reflect various noise sources. To minimize the impact of this noise on the control performance, we implemented a weighted moving average filter. (Fig. 51) (Fig. 52)

5.4 Results and Benchmarking

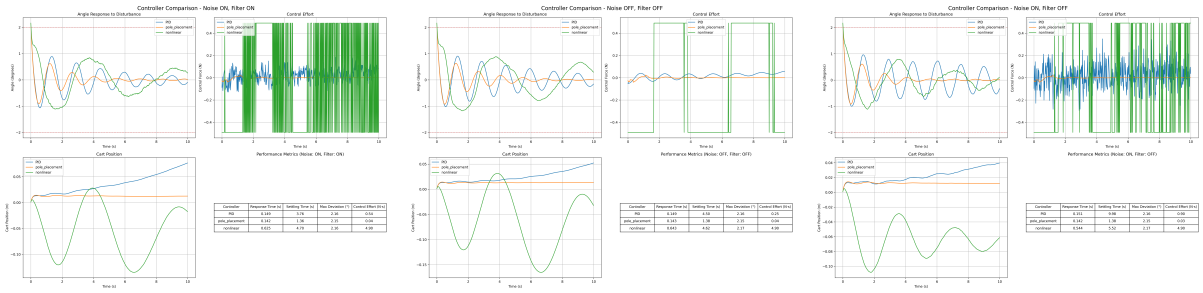

Figure 17: *
Scenario 1
Noise On, Filter On

Figure 18: *
Scenario 2
Noise Off, Filter Off

Figure 19: *
Scenario 3
Noise On, Filter Off

Figure 20: Comparison of Controller Performance Across Three Simulation Scenarios

Table 2: Comparison of Controller Performance Across Scenarios

Metric	**Scenario 1: Noise + Filtering**	**Scenario 2: Ideal (No Noise/Filter)**	**Scenario 3: Noise Only (No Filter)**
PID	Fast rise (0.149s), long-lasting oscillations, improved with filtering	Clean response, longer settling time (4.50s), reduced effort (0.25 $N \cdot s$)	High noise sensitivity, increased effort (1.27 $N \cdot s$), settling time 9.98s
Pole Placement	Smooth and efficient, minimal overshoot, 1.38s settling time, 0.06 $N \cdot s$ effort	Nearly identical to noisy case, shows high robustness	Robust to noise, 0.141s rise time, stable 1.38s settling time
Nonlinear	Larger cart swings, higher energy use (+4.90 $N \cdot s$), stabilised eventually	Unexpectedly large excursions despite no noise, energy-saving logic	Unpredictable control effort, high but stable, pendulum remained upright

6 Prototyping

6.1 Electrical Diagram (Schematics)

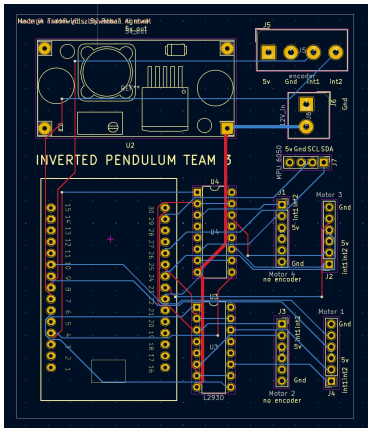

Figure 21: Custom PCB Layout for Inverted Pendulum System

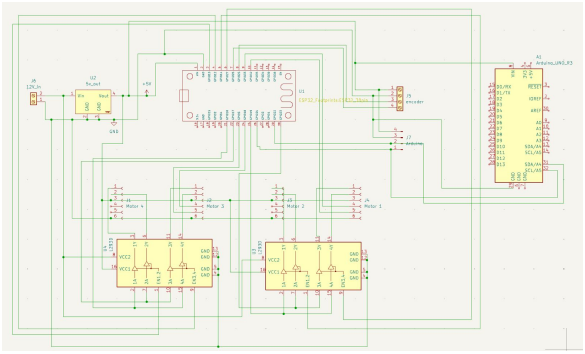

Figure 22: Full Electrical Schematic of the Inverted Pendulum System

- **Microcontroller Interface (U2):** This is where the ESP32 (or other controller) is mounted. It connects directly to peripheral components and motor drivers.
- **Motor Driver (U3 - L293D):** Positioned at the center, the L293D dual H-Bridge driver controls two DC motors. Additional space is provided for a second L293D for controlling all four motors.
- **Encoder and Motor Headers (J1 to J4):** These connectors provide organized outputs for motor pins (intH1, intH2, 5V, GND), enabling easy plug-and-play with motor encoders and wiring.
- **Power Management:**
  - Power is supplied via a 12V input header (J6), then regulated down to 5V via an onboard regulator (visible in the upper left).
  - A dedicated power trace runs to each submodule, separating logic power and motor power where needed.
- **Peripheral Integration:**
  - The MPU6050 header (U7) supports I^2C connection for the gyroscope/accelerometer.
  - I^2C lines (SCL, SDA) and power are routed directly to this module.

- **Routing:**
  - Red traces represent the top copper layer.
  - Blue traces represent the bottom copper layer.
  - Signal traces are kept short and clear to minimize interference and cross-talk.
- **Microcontrollers:**
  - **ESP32 (U1)** is used for motor control and sensor data handling. It receives commands via I^2C from the Arduino UNO (A1).
  - **Arduino UNO (A1)** is responsible for high-level control, executing the LQR algorithm, and transmitting motor commands to the ESP32.
- **Power Supply (J6, U2):**
  - A 12V input (J6) is fed into a 5V voltage regulator (U2), which powers the logic circuitry and sensor modules.
  - Power is distributed to both the motor drivers and the microcontrollers.
- **Motor Drivers (U3, U4 – L293D):**
  - Two L293D chips are used to control up to four DC motors (J1 to J4).
  - Each driver has inputs (IN1, IN2) from the ESP32, and outputs connected to motor terminals.
  - Enable pins (EN1–EN4) are connected to PWM outputs on the ESP32 for variable speed control.
- **Encoders and Sensors:**
  - The rotary encoders are connected to the ESP32 via header J5, sending signals on interrupt-capable pins for precise angular and linear motion tracking.
  - I^2C communication is used for devices such as the MPU6050, sharing SDA and SCL lines.
- **Signal Interfaces:**
  - The ESP32 and Arduino communicate via I^2C using the header labeled `J7`, with defined SDA, SCL, and GND lines.
  - Clear labeling ensures minimal error in assembly and helps during debugging.

6.2 Mechanical (CAD) Design

The mechanical structure of our inverted pendulum system was developed through iterative design and refinement. Our final model was changed to meet the specific project requirements but some aspects were inspired by existing designs found in online resources [2].

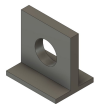

Figure 23: Bearing Holder CAD Model

Figure 24: Mass Holder CAD Model

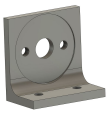

Figure 25: Motor Holder CAD Model

Figure 26: Rod Holder CAD Model

Table 3: Descriptions of Custom Mechanical Holders

Component	**Description**
Bearing Holder	Supports spinning shafts or axles using a precisely circular cutout to accommodate the bearing and reduce friction. The base plate connects to the chassis, while the vertical plate adds structural rigidity.
Mass Holder	Ensures a snug fit inside a shallow chamber and allows bottom-up insertion of the mass for easy attachment to the rod. Designed for tool-free locking using insertions rather than screws.
Motor Holder	Aligns and secures a motor to a base frame. Features include mounting holes for the frame and screw holes for attaching the motor faceplate. A vertical flange supports the motor's flat mounting face and adds rigidity.
Rod Holder	Holds the pendulum rod securely on the cart base. Includes a vertical bore for the rod and a horizontal hole for a locking pin or screw. Its oblong shape prevents rod slippage during movement.

6.3 Controller Design Approaches

PID controllers are commonly applied due to their simplicity and effectiveness in real-time applications [3]. Pole Placement techniques have been utilized for precision control in various robotic systems requiring rapid stabilization [4]. Meanwhile, LQR-based approaches have demonstrated superior robustness and energy efficiency in balancing tasks, particularly in robotics and aerospace applications [5].

6.3.1 System Modeling and Linearization

All three controllers rely on a model of the inverted pendulum mounted on a cart. We recall the physical parameters:

- M: mass of the cart (in kg),
- m: mass of the pendulum (in kg),
- l: distance from the pivot to the pendulum's center of mass (in m),
- g: gravitational acceleration (approximately $9.81\,\mathrm{m/s^2}$).

Let the states be

$$x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} \text{cart position} \\ \text{cart velocity} \\ \text{pendulum angle (relative to upright)} \\ \text{pendulum angular velocity} \end{bmatrix}.$$

Linearizing the system around $\theta = 0$, the linearized state-space representation is:

$$\dot{x} = Ax + Bu,$$

where u is the force applied to the cart, and

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & -\frac{mg}{M} & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & \frac{g(M+m)}{Ml} & 0 \end{bmatrix}, \quad B = \begin{bmatrix} 0 \\ \frac{1}{M} \\ 0 \\ -\frac{1}{Ml} \end{bmatrix}.$$

6.3.2 PID Controller

1. Measure the angle θ (and optionally x) using an encoder.
2. Compute the PID terms (proportional, integral, derivative) each loop cycle.

3. Combine the commands to form the total control input $u = u_\theta \pm u_x$ (depending on your chosen control architecture).

4. Send u to the motor driver as a PWM and direction signal.

6.3.3 Pole Placement Controller

1. After linearizing, we define the state vector $\left[x, \dot{x}, \theta, \dot{\theta}\right]^\mathsf{T}$.

2. We choose the desired poles that give us a stable and sufficiently damped response.

3. Solve for K_{pp} offline (in Python, MATLAB, etc.).

4. In the microcontroller loop, compute $u = -K_{\text{pp}}\, x$ and convert to PWM signals for the motor driver.

6.3.4 LQR Controller

1. Compute K_{lqr} offline using numerical routines such as `solve_continuous_are` in Python or the MATLAB `lqr` function.

2. In the microcontroller loop, form the state vector $x = [x, \dot{x}, \theta, \dot{\theta}]^\mathsf{T}$ from sensors/encoders.

3. Calculate the control input $u = -K_{\text{lqr}}\, x$.

4. Scale and send the command u to the motor driver for actuation.

7 Overview

7.1 Benchmarking and Results

7.1.1 LQR Controller

In this subsection, we present the performance of the system under an LQR controller. Figures 27 and 28 show the detailed angular and linear responses (with annotated metrics), while Figures 29 and 30 show the underlying position and velocity data over 20 s. A disturbance was introduced around $t = 10$ s. Key metrics for each of the signals (linear position x and angular position θ) are also given below.

- **Linear Position Response:** The LQR controller drives x from an initial value of -0.011 m to a steady state of about -0.020 m. The rise time (10 % to 90 %) is roughly 12.5 s, and the system settles by about 18 s. The overshoot is modest (about 3.1 %), and the peak excursion occurs near 20 s.

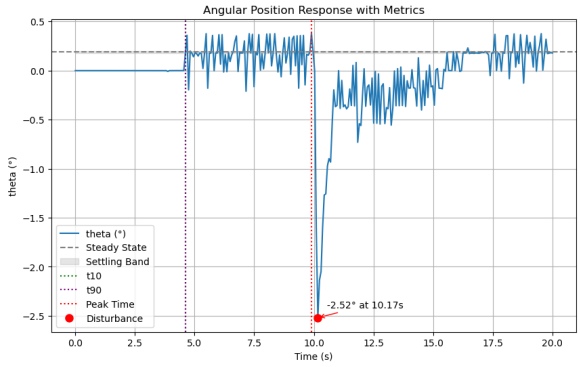

Figure 27: Angular Position Response with Metrics under LQR control. A disturbance of approximately $-2.52°$ occurs at $t = 10.17\,\text{s}$. The final steady-state angle is about $0.19°$.

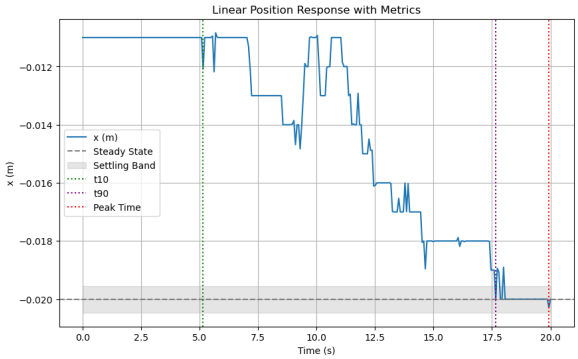

Figure 28: Linear Position Response with Metrics under LQR control. The final steady-state position is approximately $-0.0200\,\text{m}$.

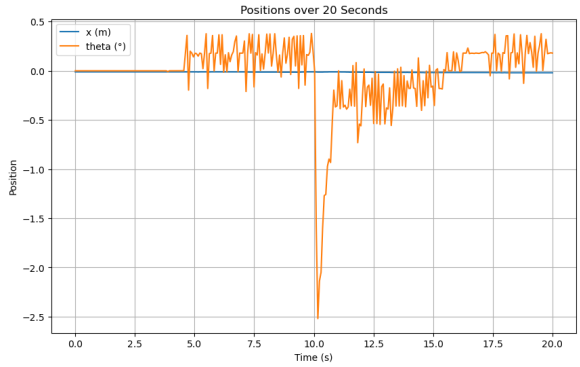

Figure 29: Overall Position (x) and Angular Position (θ) over $20\,\text{s}$ under LQR control.

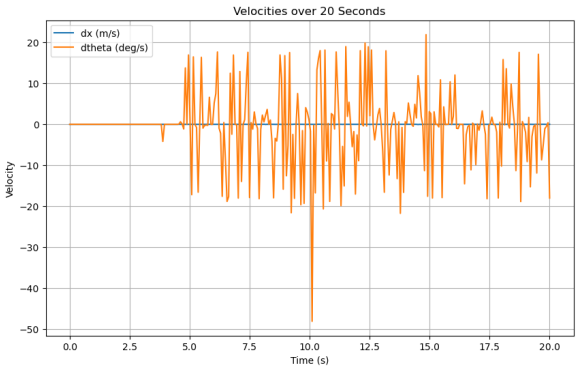

Figure 30: Velocities ($\dot{x}$ and $d\theta/dt$) over $20\,\text{s}$ under LQR control.

- **Angular Position Response:** The disturbance around $t = 10\,\text{s}$ causes a large drop to about $-2.52°$ at $t = 10.17\,\text{s}$. The final steady state remains near $0.19°$, and the controller recovers within about $10\,\text{s}$ after the disturbance. The overshoot metric appears high (about 101%) because the system goes from $0°$ to nearly $-2.52°$ upon disturbance.
- **No Failures Observed:** The maximum angular position magnitude is well below $\pm 15°$, and the linear position stays within $\pm 0.5\,\text{m}$, so no failure conditions are reached during this $20\,\text{s}$ test.

7.1.2 Pole Placement

Experimental Setup and Approach The system was commanded from an initial condition of $\theta(0) = 0°$ and $x(0) = 0$ m. The controller was designed to move the sys-

Table 4: Linear Position (x) Metrics Under LQR Control

Metric	Value
Initial Position	-0.011 m
Steady-State Position	-0.0200 m
Step Change	-0.0090 m
t_{10}	5.15 s
t_{90}	17.66 s
Rise Time ($t_{90} - t_{10}$)	12.51 s
Settling Time	11.06 s
Overshoot (%)	3.11%
Peak Time	19.93 s
RMS Error	0.00665 m
IAE	0.11488
ISE	0.000885

Table 5: Angular Position (θ) Metrics Under LQR Control

Metric	Value
Initial Angle	0.0°
Steady-State Angle	0.189°
Step Change	0.189°
t_{10}	4.62 s
t_{90}	4.62 s
Rise Time ($t_{90} - t_{10}$)	0.0 s
Settling Time	12.93 s
Overshoot (%)	100.87%
Peak Time	9.90 s
RMS Error	0.4203°
IAE	5.0200
ISE	3.5435

tem to a desired steady-state offset (i.e., its "step change") while respecting prescribed constraints:

- Angular position θ must remain below $\pm 15°$.
- Linear position x must remain within ± 0.5 m.

Results and Discussion Figure 31 shows the angular position response over time along with key performance indicators (e.g. 10% and 90% times, peak time, etc.). Figure 32 shows the analogous linear position response. Figures 33 and 34 provide additional views of the positions and velocities over the entire experiment.

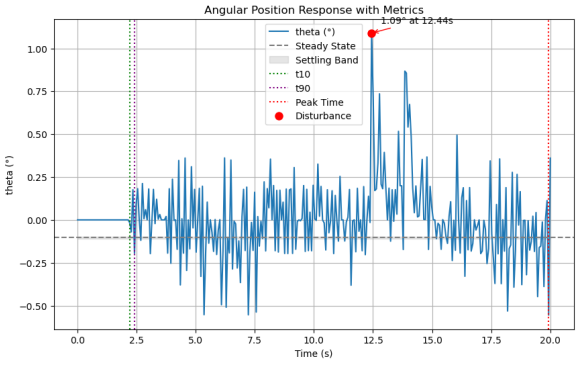

Figure 31: Angular Position Response with Metrics

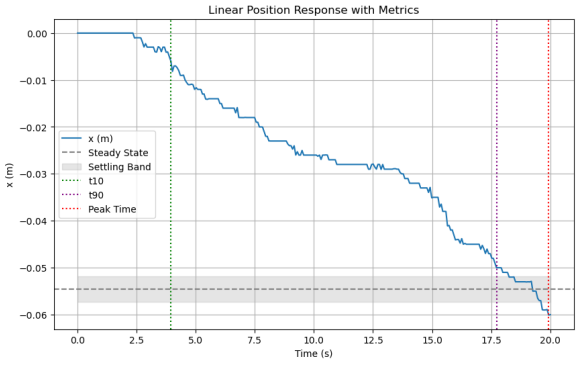

Figure 32: Linear Position Response with Metrics

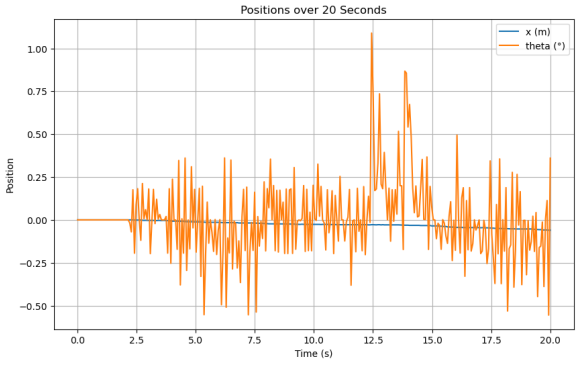

Figure 33: Positions (linear x and angular θ) over 20 s

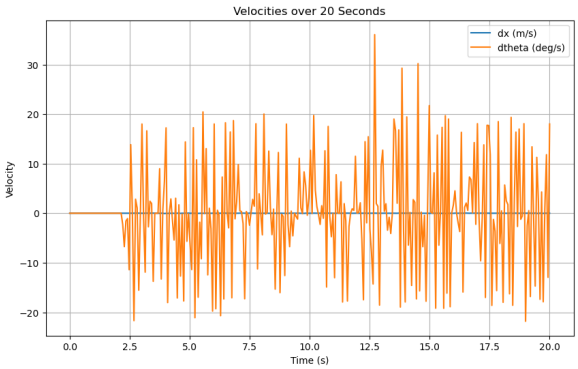

Figure 34: Velocities ($\dot{x}$ and $d\theta/dt$) over 20 s

Linear Position Metrics Table 6 summarizes the key performance metrics for the linear position x. Notably, the system has a steady-state value of about -0.0545 m, a 10% rise time near 3.95 s, and a 90% rise time near 17.73 s. This results in an overall rise time (90%-10%) of roughly 13.78 s. The overshoot is about 10.25%, and the final peak occurs near 19.93 s.

Angular Position Metrics Table 7 lists the corresponding metrics for the angular position θ. A key observation is the large overshoot percentage of about 438%. However, the absolute magnitude of θ remained below $\pm 15°$, indicating that the physical constraint was not violated. The 10% and 90% times are approximately 2.21 s and 2.41 s, giving a short overall rise time of roughly 0.20 s. The system's final steady state is near $-0.1032°$.

Table 6: Linear Position (x) Metrics

Metric	**Value**
Initial Position $x(0)$	0.0 m
Steady State	-0.0545 m
Step Change	-0.0545 m
t_{10} (s)	3.9465
t_{90} (s)	17.7258
Rise Time (s)	13.7793
Settling Time (s)	16.43
Overshoot (%)	10.25%
Peak Time (s)	19.9331
RMS Error	0.0342
IAE	0.5982
ISE	0.0233

Table 7: Angular Position (θ) Metrics

Metric	**Value**
Initial Angle $\theta(0)$	0.0°
Steady State	$-0.1032°$
Step Change	$-0.1032°$
t_{10} (s)	2.2074
t_{90} (s)	2.4080
Rise Time (s)	0.2006
Settling Time (s)	15.23
Overshoot (%)	438.03%
Peak Time (s)	19.9331
RMS Error	0.2502
IAE	3.7366
ISE	1.2491

Conclusion From the data, the pole placement controller successfully kept the system within acceptable bounds. The linear position steadily moved to approximately -0.0545 m

with about 10% overshoot, while the angular position remained near its target and did not exceed $\pm 15^\circ$ in magnitude, despite a large overshoot figure in percentage terms.

7.1.3 PID Analysis

Figures 35–38 show the time responses and the computed performance metrics are summarized below.

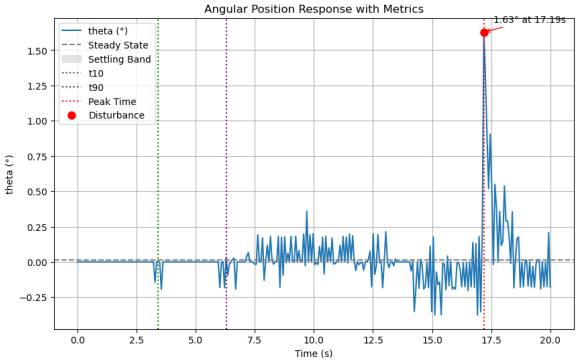

Figure 35: Angular Position Response with Metrics

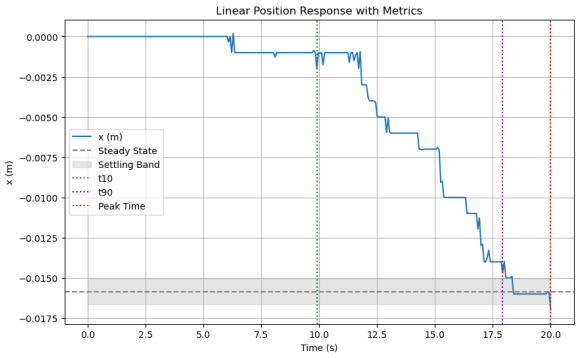

Figure 36: Linear Position Response with Metrics

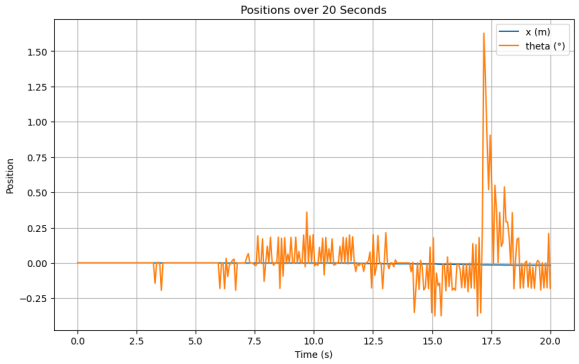

Figure 37: Positions (x and θ) over 20 s

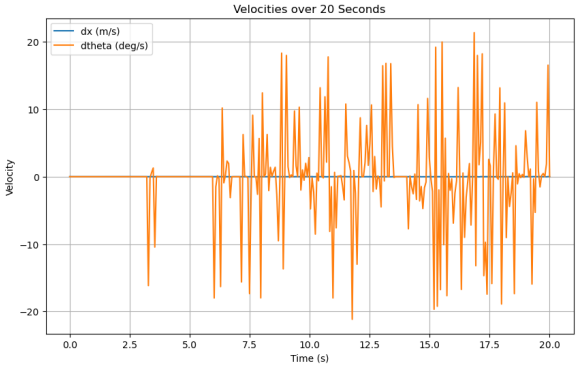

Figure 38: Velocities ($\dot{x}$ and $\dot{\theta}$) over 20 s

Table 8: Linear Position (x) Metrics

Metric	Value
Initial Position $x(0)$	0.0 m
Steady State	−0.0158554 m
Step Change	−0.0158554 m
t_{10} (s)	9.90
t_{90} (s)	17.93
Rise Time (s)	8.03
Settling Time (s)	4.04
Overshoot (%)	7.22%
Peak Time (s)	20.0
RMS Error	0.01266 m
IAE	0.22845
ISE	0.003208

Table 9: Angular Position (θ) Metrics

Metric	Value
Initial Angle $\theta(0)$	0.0°
Steady State	0.0137440°
Step Change	0.0137440°
t_{10} (s)	3.41
t_{90} (s)	6.29
Rise Time (s)	2.88
Settling Time (s)	3.98
Overshoot (%)	11739.73%
Peak Time (s)	17.19
RMS Error	0.19003°
IAE	1.84791
ISE	0.72342

7.1.4 PID_with_failure

As shown in the plots below, the angular position exhibits a significant overshoot around $t \approx 12.4\,\text{s}$. That large deviation arises because of the deliberate human action, rather than controller instability alone. The linear position, though perturbed, remains within acceptable bounds (well below 0.5 m).

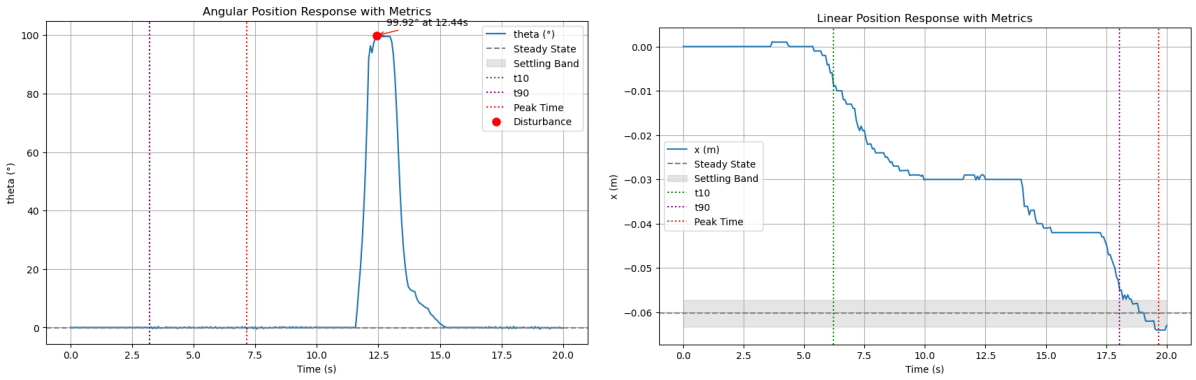

Figure 39: (Left) Angular Position Response with Metrics (Right) Linear Position Response with Metrics. The overshoot in angular position is a result of a large human-induced disturbance.

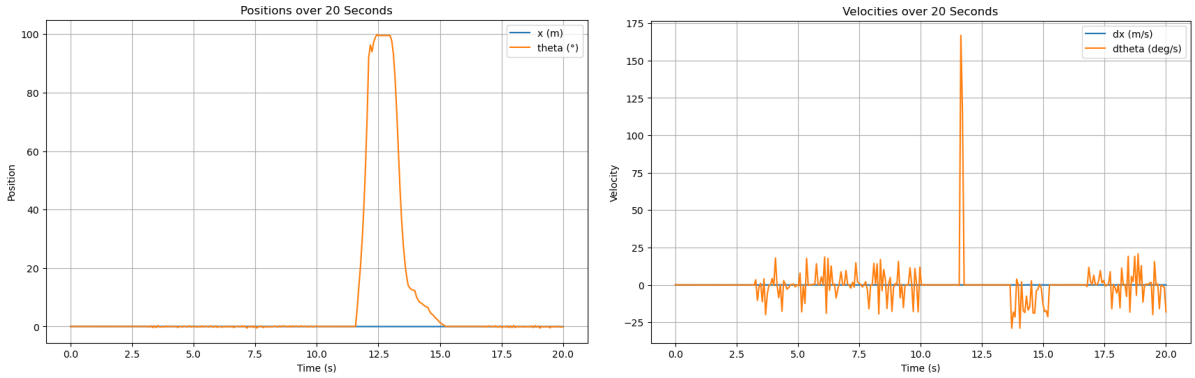

Figure 40: (Left) Positions over 20 Seconds (Right) Velocities over 20 Seconds.

Below are the key metrics captured during this run:

Table 10: Linear and Angular Position Metrics

Linear Position (x) Metrics

Metric	Value
Initial	0.0
Steady State	-0.0602
Step Change	-0.0602
t_{10} (s)	6.2207
t_{90} (s)	18.0602
Rise Time (s)	11.8395
Settling Time (s)	None
Overshoot (%)	6.3409
Peak Time (s)	19.6656
RMS Error	0.0407
IAE	0.7177
ISE	0.0331

Angular Position (θ) Metrics

Metric	Value
Initial	0.0
Steady State	-0.1052
Step Change	-0.1052
t_{10} (s)	3.2107
t_{90} (s)	3.2107
Rise Time (s)	0.0
Settling Time (s)	None
Overshoot (%)	427.7388
Peak Time (s)	7.1572
RMS Error	25.0158
IAE	157.9738
ISE	12557.6483

7.1.5 Failure Metrics

The system was subjected to five disturbances with the following settling times:

Disturbance Performance Summary:

- 1st disturbance: 4.6 s
- 2nd disturbance: 3.7 s
- 3rd disturbance: 6.78 s
- 4th disturbance: 4.9 s
- 5th disturbance: **Failed** (no settling time; the robot fell over)

Excluding the failed 5th disturbance, the average settling time is:

$$\bar{t}_{\text{settle}} = \frac{4.6 + 3.7 + 6.78 + 4.9}{4} \approx 5.0 \text{ s}$$

The best performance was observed at a 6° pendulum angle, where the system maintained stability.

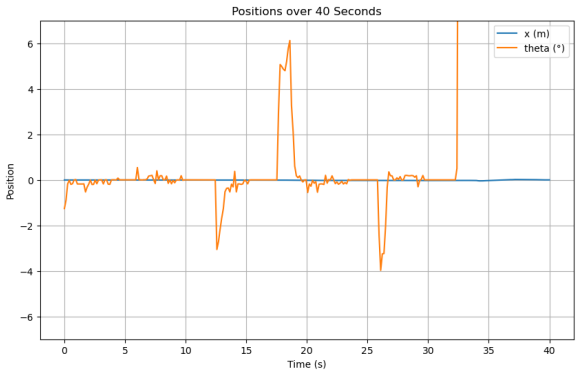

Figure 41: System response over 40 seconds. Stability maintained at 6°; failure occurred on 5th disturbance.

Additional Failure Times:

In further tests, four separate failure points were recorded (where the system could no longer balance):

- Failure 1: 32.51 s
- Failure 2: 32.37 s
- Failure 3: 32.52 s
- Failure 4: 10.30 s

Mean Time to First Failure (MTTF):

$$\text{MTTF} = 32.51 \text{ s}$$

(This is simply the time of the first observed failure.)

Mean Time Between Failures (MTBF):

$$\text{MTBF} = \frac{32.51 + 32.37 + 32.52 + 10.30}{4} \approx 26.93 \text{ s}$$

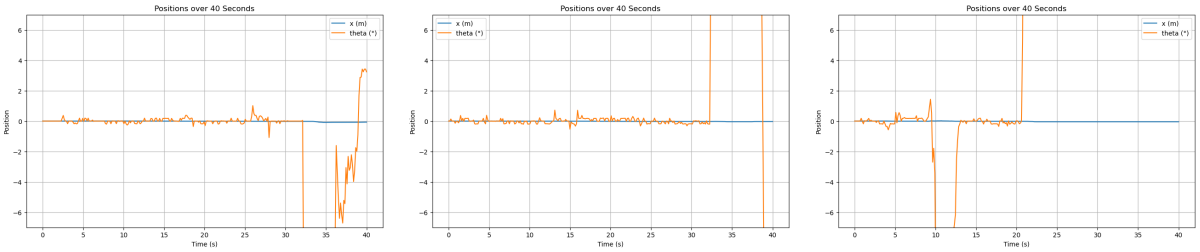

Figure 42: Failure at 32.37 s Figure 43: Failure at 32.52 s Figure 44: Failure at 10.30 s

Figure 45: System response plots showing failures during different disturbances.

7.1.6 Overall Controller Performance Comparison

Table 11 gives a concise overview of how all three controllers (PID, LQR, and Pole Placement) performed. While each controller remained within the specified safety limits ($\pm 15°$ for angle and $\pm 0.5\,\text{m}$ for position), the following observations were made (from best to worst overall behavior):

Table 11: High-Level Comparison of Controller Outcomes

Aspect	**PID**	**LQR**	**Pole Placement**
Max $\lvert\theta\rvert$ (deg)	$\sim 1.6°$	$< 3°$	$< 4°$
Max $\lvert x\rvert$ (m)	~ 0.02	< 0.03	~ 0.06
Angle Overshoot (%)	$\sim 11740\%$ (small setpoint)	$\sim 100\%$	$\sim 438\%$
Position Overshoot (%)	$\sim 7\%$	$\sim 3\%$	$\sim 10\%$
Settling Time (s)	3.98	12.93	15.23
Overall Smoothness	Most smooth	Moderate	Very aggressive
Safety Limits Violated?	No	No	No

8 Conclusion

This project compared PID, Pole Placement, and Nonlinear Control for inverted pendulum stabilisation. Each had performance, complexity, and robustness tradeoffs.

Although simple and effective, the PID controller is sensitive to tuning and disturbances [3]. Pole placement enhanced stability and response time, but required a precise system model [4]. While nonlinear control was more robust and handled more circumstances, it required a better understanding of the system [5].

The optimum strategy is the PID controller. With low computing overhead, it is simple and effective. PID's ease of implementation and ability to work well in many situations make it the best option for stabilising an inverted pendulum, notwithstanding its tuning and disturbance sensitivity.

9 References

References

[1] U. of Michigan, "Inverted pendulum: System modeling," n.d. [Online]. Available: https://ctms.engin.umich.edu/CTMS/index.php?example=InvertedPendulum&section=SystemModeling

[2] A. Benitez, "Inverted pendulum cad model," 2024. [Online]. Available: https://grabcad.com/library/inverted-pendulum-4

[3] A. Unknown, "Optimal controller design for inverted pendulum system: A comparative study," *International Journal of Scientific and Engineering Research*, vol. 10, no. 5, pp. 1–6, 2019. [Online]. Available: https://www.ijser.org/researchpaper/OPTIMAL-CONTROLLER-DESIGN-FOR-INVERTED-PENDULUM-SYSTEM-A-COMPARATI pdf

[4] S. Irfan, L. Zhao, S. Ullah, A. Mehmood, and M. F. U. Butt, "Control strategies for inverted pendulum: A comparative analysis of linear, nonlinear, and artificial intelligence approaches," *PLOS ONE*, vol. 19, no. 3, p. e0298093, 2024. [Online]. Available: https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0298093

[5] A. Unknown, "Optimal control of nonlinear inverted pendulum system using pid controller and lqr," *Journal of Control, Automation and Electrical Systems*, vol. 25, no. 2, pp. 187–194, 2014. [Online]. Available: https://link.springer.com/article/10.1007/s11633-014-0818-1

10 Appendix

10.1 Ethical Considerations

A private business wants to exploit the idea to make a government-sold armament system. Kant's formalism or obligation ethics says we must defend everyone's life. Since the design would be utilised to construct a weapon system that would damage people, we must not weaponise it. Since the creator cannot guarantee ethical use, we must not hand over the design. The private corporation and government will have their own responsibility to ensure the technology is not mishandled. Mill's utilitarianism maximises everyone's net good. The ethical conclusion for utilitarianism depends on whether the weapon saves or kills people. To help more people, the government will employ the weapon, which is unethical because it would harm lives. Weaponising the design is severe and immoral, according to Aristotle's virtue ethics. The private corporation should not develop since aiding war is immoral unless it is morally excellent. Contractionism says everyone has rights by being alive. Thus, anybody harmed by this design's weaponization should have their right to life protected. In this case, handing over the design would violate the individual's right to life, even though it would express their freedom.

10.2 Figures and Code Snippets

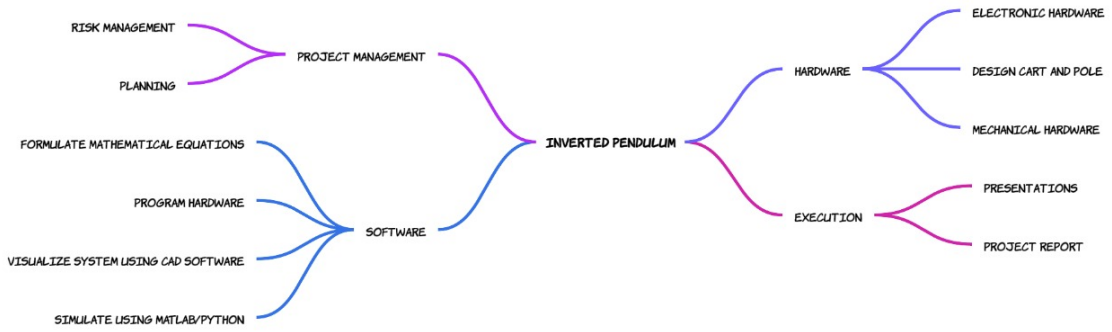

Figure 46: Mind Map for Inverted Pendulum Project

	A	B	C	D	E	F
1	Milestone	Steps	Responsibility	Time	Pre	Resources
2	1. Milestone A	a. Make Pres.	D	1		Projectfile
3		b. Give Pres.	A, P	2	1.a	Projectfile
4	2. Milestone B	a. Modelling	M, R	3	1.b	Eclipse Program
5		b. BDD, IBP, Diagrams	M	3	2.a	Eclipse Program
6		c. Present	A, D	2	2.b	Powerpoint
7	3. Milestone C	a. Apply RM methods	A	6	1.b	Powerpoint
8		b. Present	A, D	2	3.a	Powerpoint
9	4. Milestone D	a. Eq. of motion	R, M	2		Matlab
10		b. Sensor Sim.	M	2		python/simulink
11		c. Controller Sim.	M	3	4.a	python/simulink
12		d. Perform Implementation	R, M	3	4.c	innovation lab
13		e. Visualisation	R, M	1	4.d	Fusion 360
14		f. Present	A, D	2	4.d-e	powerpoint
15	5. Milestone E	a. Prototype	R, M	6 days	4.a, 4.c, 4.d	Eclipse Program, 3D printer
16		b. Testing	R, M	1	5.a	Eclipse Program, 3D printer
17	6. Milestone F	a. Written content	A, D	4	5.a	overleaf
18		b. Video	R, M	1	6.a	adobe/editing software
19		c. System code	R, M	1	5.a	github
20						

Figure 47: Work Breakdown Structure (WBS) Table for the Inverted Pendulum Project

Name	Duration	Start	Finish	Predecessors
⊟Project Inverted Pendul	**48 days**	**1/16/25 8:00 AM**	**3/24/25 5:00 PM**	
⊟Presentations	**45 days**	**1/16/25 8:00 AM**	**3/19/25 5:00 PM**	
⊟Milestone A	**4 days**	**1/16/25 8:00 AM**	**1/21/25 5:00 PM**	
Make Presentation	3 days	1/16/25 8:00 AM	1/20/25 5:00 PM	
Give Presentation	1 day	1/21/25 8:00 AM	1/21/25 5:00 PM	4
⊟Milestone B	**8 days**	**1/22/25 8:00 AM**	**1/31/25 5:00 PM**	
Modelling	3 days	1/22/25 8:00 AM	1/24/25 5:00 PM	5
BBD,IBD and software	3 days	1/27/25 8:00 AM	1/29/25 5:00 PM	7
Presentation B	2 days	1/30/25 8:00 AM	1/31/25 5:00 PM	8
⊟Milestone C	**8 days**	**2/7/25 8:00 AM**	**2/18/25 5:00 PM**	
Apply Risk Managemen	6 days	2/7/25 8:00 AM	2/14/25 5:00 PM	5
Presentation C	2 days	2/17/25 8:00 AM	2/18/25 5:00 PM	11
⊟Milestone D	**11 days**	**2/25/25 8:00 AM**	**3/11/25 5:00 PM**	
Equations of Motion	2 days	2/25/25 8:00 AM	2/26/25 5:00 PM	
Sensor Noise Simulatior	2 days	2/25/25 8:00 AM	2/26/25 5:00 PM	
Controller Simulator	3 days	2/27/25 8:00 AM	3/3/25 5:00 PM	14
Python implementation	3 days	3/4/25 8:00 AM	3/6/25 5:00 PM	16
Visualisation	1 day	3/7/25 8:00 AM	3/7/25 5:00 PM	17
Presentation D	2 days	3/10/25 8:00 AM	3/11/25 5:00 PM	17;18
⊟Milestone E	**7 days**	**3/11/25 8:00 AM**	**3/19/25 5:00 PM**	
Prototype	6 days	3/11/25 8:00 AM	3/18/25 5:00 PM	14;16;17
Test in Lab	1 day	3/19/25 8:00 AM	3/19/25 5:00 PM	21
⊟Final Report	**4 days**	**3/19/25 8:00 AM**	**3/24/25 5:00 PM**	
⊟Milestone F	**4 days**	**3/19/25 8:00 AM**	**3/24/25 5:00 PM**	
Written Content	4 days	3/19/25 8:00 AM	3/24/25 5:00 PM	21
Videos	1 day	3/20/25 8:00 AM	3/20/25 5:00 PM	22
System Code	1 day	3/19/25 8:00 AM	3/19/25 5:00 PM	21

Figure 48: Gantt Chart Table: Project Task Breakdown and Dependencies

Category	Description
Technical Risk: Sensor Inaccuracy	Drift, noise, or delay in sensor data can lead to incorrect control decisions.
Technical Risk: Controller Instability	Poorly tuned controllers can result in system oscillations.
Technical Risk: Mechanical Failure	Includes weak mounting of the pendulum and excessive friction at the pivot point.
Legal Risk: IP Infringement	The project may face legal consequences for copyright or patent violations without proper attribution.
Legal Risk: Safety and Liability Concerns	If the system malfunctions and causes harm, there could be legal consequences for the team.
Legal Risk: Data Privacy Compliance	If the system logs user interactions or sensor data, it must comply with data protection regulations.

Table 12: Risk Breakdown Structure

Failure Mode	**Effect of Failure**	**S**	**O**	**D**	**RPN**
Incorrect sensor reading	Unstable system response	8	6	4	**192**
Motor driver failure	No motion / system collapse	9	4	3	**108**
Loose wire connection	Intermittent power or signal	6	5	5	**150**
Overheating components	Unexpected shutdown	7	3	6	**126**
Control loop timing issues	Delayed corrections / oscillations	6	6	5	**180**
Incorrect LQR tuning	System cannot stabilize	7	5	4	**140**
Bluetooth interference (if active)	Data transmission failure	5	4	6	**120**

Table 13: FMEA Table for Inverted Pendulum Project

```
def angle_only_pole_placement(self):
    A2 = np.array([
        [0, 1],
        [self.g / self.L, 0]
    ])
    B2 = np.array([
        [0],
        [1 / (self.m * self.L**2)]
    ])
    poles2 = [-3, -4]
    K2 = signal.place_poles(A2, B2, poles2).gain_matrix.flatten()
    return K2
```

Figure 49: Place poles function

```
def step_dynamics(self, dt, controller_type="PID"):
    s0 = self.state
    k1 = self.dynamics(s0, controller_type)
    k2 = self.dynamics(s0 + 0.5 * dt * k1, controller_type)
    k3 = self.dynamics(s0 + 0.5 * dt * k2, controller_type)
    k4 = self.dynamics(s0 + dt * k3, controller_type)
    new_state = s0 + (dt / 6.0) * (k1 + 2 * k2 + 2 * k3 + k4)
    # ...additional constraints and state updates...
```

Figure 50: Step Dynamics function

```
if self.noise_enabled:
    x_meas += np.random.normal(0, 0.01)
    x_dot_meas += np.random.normal(0, 0.02)
    theta_meas += np.random.normal(0, 0.01)
    theta_dot_meas += np.random.normal(0, 0.02)
```

Figure 51: Noise enabled

```
if self.filter_enabled:
    self.filter_history.append([
        x_meas, x_dot_meas, theta_meas, theta_dot_meas
    ])
    if len(self.filter_history) > self.filter_len:
        self.filter_history.pop(0)

    # Apply weighted moving average
    if len(self.filter_history) >= 3:
        weights = np.linspace(0.5, 1.0, len(self.filter_history))
        weights = weights / np.sum(weights)

        # Reset measurements
        x_meas = x_dot_meas = theta_meas = theta_dot_meas = 0

        # Apply weighted average
        for i, hist in enumerate(self.filter_history):
            x_meas += hist[0] * weights[i]
            x_dot_meas += hist[1] * weights[i]
            theta_meas += hist[2] * weights[i]
            theta_dot_meas += hist[3] * weights[i]
```

Figure 52: Filter enabled

10.3 Planning and Results Video Link or Screenshot

The following video links provide further insight into our planning, results, and testing using alternative control algorithms. Click on the links below to view the respective videos:

- Evaluation 1 and 2
- Testing with PID, LQR and Pole Placement

10.4 Code Submissions

The project code has been submitted via Moodle. For a complete view of the source code, please visit the GitHub repository below:

Inverted Pendulum GitHub Repository