BalanceBot

Simulink-Based Control Deployment on Hardware

Nithish Krishnabharathi GNANI & Müfide GÜLEKEN

07 April 2025

Contents

1 Project Description

The BalanceBot project can be divided into two phases. In the initial phase, we complemented the research conducted at i3S Laboratory with the objective of gaining an understanding of the Sigi robot. Sigi is an inverted pendulum on wheels developed at the Institute for Dynamic Systems and Control (IDSC) at ETH Zurich. It can be seen in Figure 1. The aim of the first phase was to model the robot digitally on MATLAB/Simulink, so that control algorithms can be implemented. In the second phase, we developed and implemented control algorithms on the two-wheeled robot to balance it.

For the development of the control algorithms we began with the standard inverted pendulum configuration, which was modeled to capture the system dynamics. Following the development of this model, a PID controller was formulated and evaluated. Two techniques were employed for tuning the PID controller: the model-following approach, which aims to match the system response to a desired reference model, and the Ziegler-Nichols method, which uses the critical gain and the oscillation period for parameter tuning.
In addition to the PID controller, a phase lead controller was designed to improve the system response and stability by adding a phase margin.
Initial testing of all controllers was conducted in simulation to verify their performance. While the lead phase controller demonstrated potential in simulation, its performance on the hardware was unsatisfactory. Consequently, we focused on optimizing the PID approach. The PID controller was deployed on the Sigi hardware and its parameters were calibrated to adapt to real-world dynamics.

In the later phase of the project, the scope was expanded to include an inverted pendulum with translational motion. To control this more complex configuration, a state feedback controller and an LQR controller were calculated and implemented in the simulation environment. Despite the simulation results indicating potential, attempts to deploy the extended system on hardware within the project timeline proved unsuccessful.

Figure 1: A photograph of the Sigi robot

2 System Description

Sigi is a wheeled inverted pendulum based on Pololu's Balboa 32U4 balancing robot kit [1]. As can be seen in figure 2, the control algorithms are designed in Matlab Simulink. These are then compiled and run on a Raspberry Pi 3B+ [3]. The Raspberry Pi communicates with the Pololu Balboa32U4 board through the I2C communication protocol. The Pololu Balboa32U4 includes an IMU sensor, among other hardware components, which are connected to an Atmel ATmega32U4 microprocessor. The system utilizes two Pololu Micro Metal 6V HPCB Gearmotors, which are controlled by Texas Instruments DRV8838 motor drivers. These motors drive Pololu Wheels (80 x 10 mm), with the following features:

- Diameter: 80 mm
- Thickness: 10 mm
- Material: ABS (Polylac) PA-7475 hub with a silicone rubber tire

To achieve the necessary torque and speed reduction, an external plastic gearbox with a 30:1 gear ratio is attached to the motors. For motion tracking, external wheel encoders are mounted outside the wheels to analyze backlash effects due to gearing.

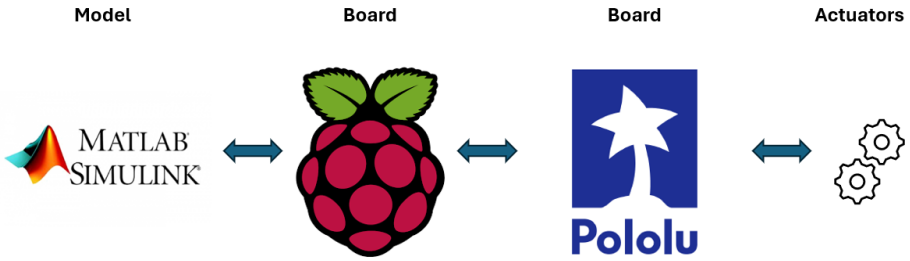

Figure 2: An overview of the system components

The three main components of the control loop architecture are presented in Figure 3. The System block consists of the Sigi robot and the I2C interface between the Pololu board and the Raspberry Pi. The observer block comprises reading the sensor outputs, the system model including the state estimator. Finally, the Controller block consists of the control algorithms we developed. After compiling, the Simulink model can be run in

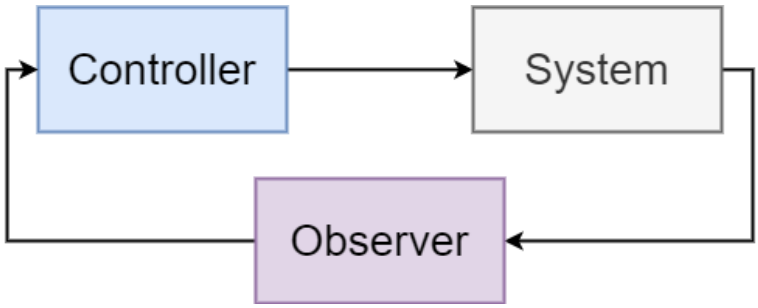

Figure 3: Control loop schematic

different modes on the Raspberry Pi [2]:

1. Normal mode simulation without Simulink IO: The model resides on the computer and the simulation is run independent from the Sigi hardware
2. Normal mode simulation with Simulink IO: The model is run in simulation mode on the computer, but the peripheral data is gathered by the actual hardware
3. External mode: The model is deployed on the hardware and the peripheral data is gathered by the hardware too. The data is exchanged with Simulink.
4. Deploy to hardware: The model is deployed on the hardware and runs independently from Simulink

3 Estimating the parameters

To model the robot, we need to estimate its parameters, such as the motor constant, resistance and gear ratio. To determine the gear ratio, the wheel was turned by a certain number of rotations, and the encoder values were read in MATLAB/Simulink. The encoder is 16-bit and hence overflows, leading to spikes in the measurements. Hence, code was written to handle the overflows by adding the reading to the last reading before overflow. The ratio between the number of wheel turns and the number of turns of the motor recorded was used to calculate the gear ratio. The plot of the rotations and velocity is shown in Figure 4. It can be seen that the two motors behave differently due to different resistances. An oscilloscope was used to measure and manually log voltage readings by gradually increasing the motor input duty cycle from 0 to 100%. The measurements unfortunately did not gave reliable information on neither the motor constant nor the motor resistance.
Therefore the values were approximated by using the least squares estimation method, using the `lsqr()` function in MATLAB [4]. This is a mathematical procedure for identifying the optimal curve that best fits a given set of points. This is achieved by minimizing the sum of the squares of the distances between the points and the curve. The results are shown in Figure 5. Hence, we average the estimated parameters for our model.

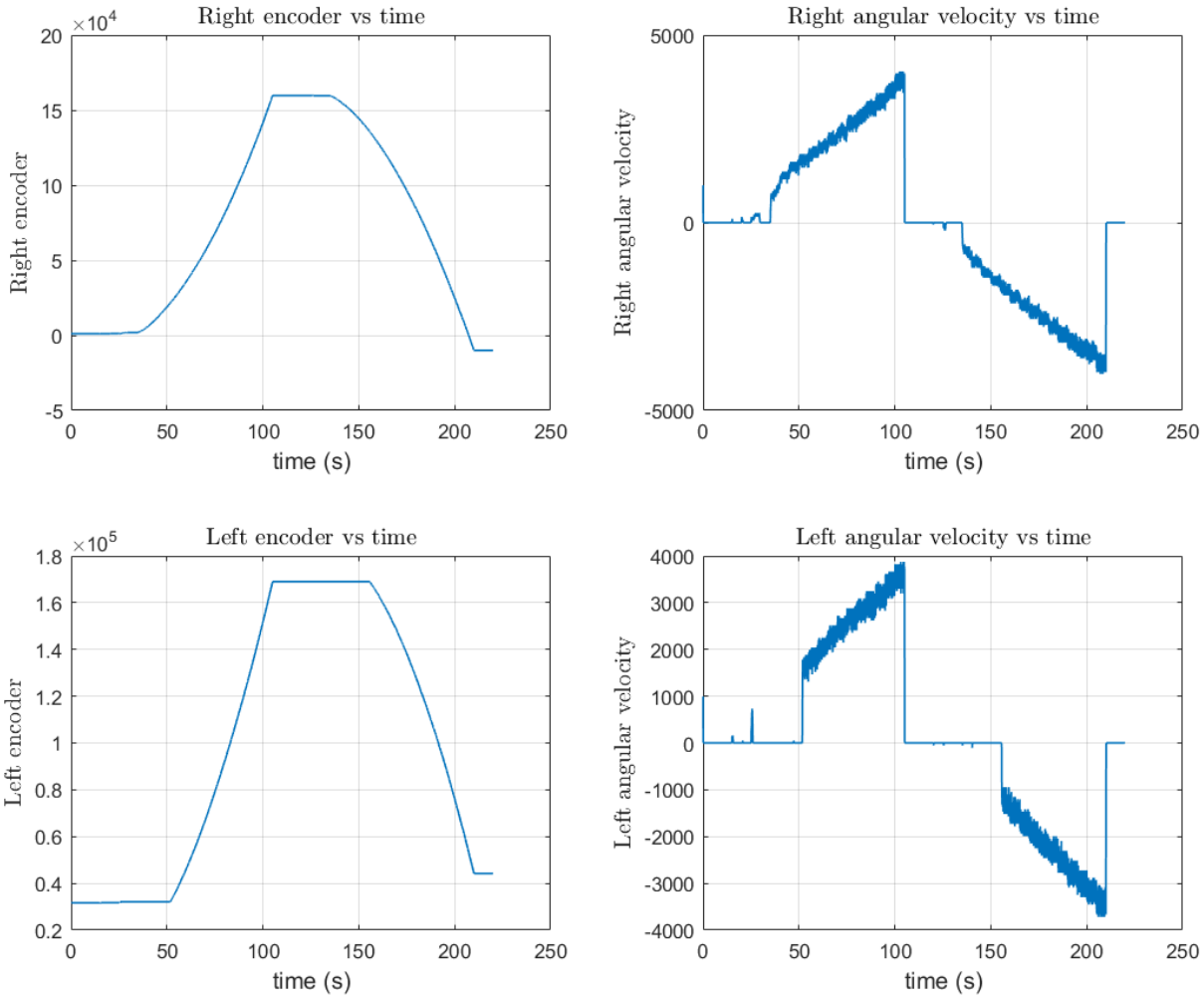

Figure 4: Encoder reading and motor angular velocity

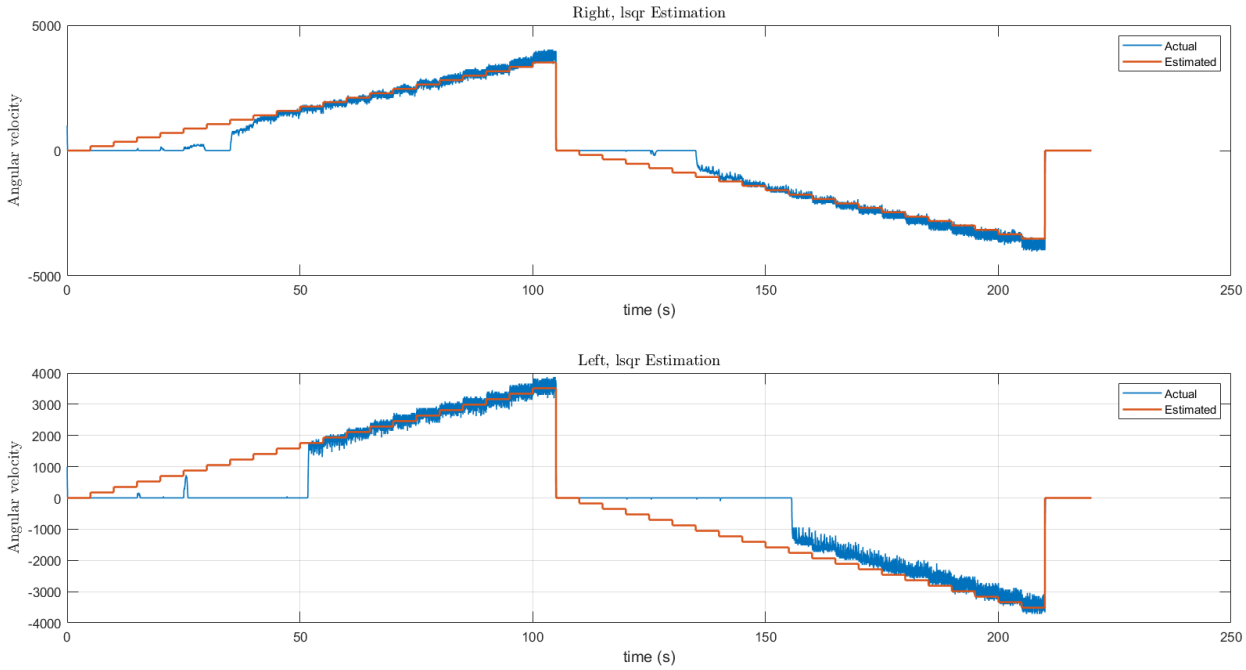

Figure 5: Least Square estimation on the measured data

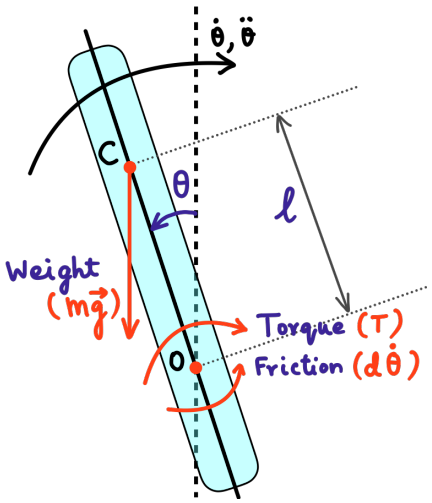

Figure 6: Forces and moments acting on the inverted pendulum

4 System Modeling - Inverted Pendulum

4.1 Derivation of Transfer Function of inverted pendulum

If the wheels of the Sigi are locked in place, the body behaves like an inverted pendulum. The forces and moments acting on it are shown in Figure 6. The pivot axis passes through point O and the mass of the body acts at the center of mass, point C. The length $\overline{OC} = \ell$. θ is the pitch angle of the Sigi body with respect to the vertical axis.

The equation of motion for the system is:

$$J\ddot{\theta}(t) = T - mg\ell \sin\theta(t) - d\dot{\theta}(t) \tag{1}$$

The parameters and their values are provided in table 1. The torque equation is given by:

$$T = \frac{K_m}{R_m}U - \frac{K_m^2}{R_m}i_{gb}\frac{\dot{x}_w}{r_w} + \frac{K_m^2}{R_m}i_{gb}\dot{\theta}$$

Where, T is the output Torque of a motor, U is the input voltage to the motor, K_m is the motor constant, i_{gb} the gearbox ratio, r_w is the radius of the wheel and R_m is the resistance of the motor coil.

For the inverted pendulum case, we hold the wheel in place. Hence the translation velocity, $\dot{x} = 0$. So the equation simplifies to:

$$T = \frac{K_m}{R_m}U + \frac{K_m^2}{R_m}i_{gb}\dot{\theta} \tag{2}$$

Substituting Equation 8 into Equation 2:

$$J\ddot{\theta}(t) = \frac{K_m}{R_m}u(t) + \frac{K_m^2}{R_m}i_{gb}\dot{\theta}(t) - d\dot{\theta}(t) - mg\ell\underbrace{\sin\theta(t)}_{\approx\theta} \tag{3}$$

For small angles, we approximate $\sin\theta \approx \theta$, linearizing the system at the operating point $\theta = 0$:

$$J\ddot{\theta}(t) = \frac{K_m}{R_m}u(t) + \frac{K_m^2}{R_m}i_{gb}\dot{\theta}(t) - d\dot{\theta}(t) - mg\ell\theta(t)$$

Taking the Laplace transform (assuming zero initial conditions):

$$s^2\Theta(s) = \frac{K_m}{JR_m}U(s) + \left(\frac{K_m^2 i_{gb}}{JR_m} - \frac{d}{J}\right)s\Theta(s) - \frac{mg\ell}{J}\Theta(s)$$

Rearranging for $\Theta(s)$:

$$(Js^2 - \left(\frac{K_m^2 i_{gb}}{R_m} - d\right)s + mg\ell)\Theta(s) = \frac{K_m}{R_m}U(s)$$

Thus, the transfer function is:

$$G(s) = \frac{\Theta(s)}{U(s)} = \frac{K_m}{R_m\left(Js^2 - \left(\frac{K_m^2 i_{gb}}{R_m} - d\right)s + mg\ell\right)} \tag{4}$$

4.2 Stability analysis of the plant model

Since the transfer function of the inverted pendulum is of second order, the stability of the system is determined. The poles of the transfer function are calculated as follows:

$$\begin{aligned} G(s) &= \frac{K}{R_m J s^2 - (K_m^2 i_{gb} - R_m d)\, s + R_m mg\ell} \\ &\rightarrow R_m J s^2 - \left(K_m^2 i_{gb} - R_m d\right) s + R_m mg\ell \stackrel{!}{=} 0 \\ s &= \frac{K_m^2 i_{gb} - R_m d \pm \sqrt{[K_m^2 i_{gb} - R_m d]^2 - 4 R_m^2 J mg\ell}}{2 R_m J} \\ \text{we get: } s_1 &= -2.285 + 12.68i \\ \text{and } s_2 &= -2.285 - 12.68i \end{aligned}$$

Due to the real part of the poles being smaller then zero, the poles are indeed stable.

4.3 Controller design using Reference Model Following Approach

We consider a first order system as reference model:

$$M(s) = \frac{1}{1 + \tau_m s} \tag{5}$$

Using the model following approach, the controller is given by:

$$C(s) = \frac{M(s)}{(1 - M(s))G(s)} \tag{6}$$

Substituting Equations 4 and 5 in 6:

$$C(s) = \frac{1}{\tau_m s K_m} = \frac{J R_m s^2}{\tau_m K_m s} - \frac{R_m \left(\frac{K_m^2 i_{gb}}{R_m} - d\right) s}{\tau_m K_m s} + \frac{R_m mg\ell}{\tau_m K_m s} \cdot \frac{1}{s}$$

Identifying the PID components:

$$C(s) = \underbrace{-\frac{K_m^2 i_{gb} - d R_m}{\tau_m K_m}}_{P} + \underbrace{\frac{R_m mg\ell}{\tau_m K_m} \cdot \frac{1}{s}}_{I} + \underbrace{\frac{J R_m}{\tau_m K_m} \cdot s}_{D} \tag{7}$$

4.4 Lead-Phase Controller with Feedback Linearization

The lead-phase controller is defined as $\frac{1+a\tau s}{1+\tau s}$. To acquire the a and τ value, we first linearize the system by applying feedback linearization. Since the system has two non-linear terms, $\sin\theta(t)$ and $\dot{\theta}(t)$:

$$J\ddot{\theta}(t) = T - mg\ell \sin\theta(t) - d\dot{\theta}(t) \tag{8}$$

the output of the plant is fed back to the input of the controller to cancel out the non-linear terms.
Therefore we get:

$$\begin{aligned} J\ddot{\theta}(t) &= T \\ &= \frac{K_m}{R_m \left(J s^2 - \frac{K_m^2 i_{gb}}{R_m} s\right)} \\ &= \frac{\frac{K_m}{J R_m}}{s^2 - \frac{K_m^2 i_{gb}}{J R_m} s} \end{aligned}$$

We want the lead phase controller to introduce a phase shift of 45 °for a good balance between stability and responsiveness. So:

$$\tan^{-1}\left(\frac{a-1}{2\sqrt{a}}\right) \stackrel{!}{=} 45° \tag{9}$$

We obtain this by setting $a = 5.82$.
To obtain the desired phase shift of 45°, we calculate τ as follows. We define: $\alpha = \frac{K_m^2 i_{gb}}{JR_m}$, $\beta = \frac{K_m}{JR_m}$.

$$1 \stackrel{!}{=} \left|\frac{1+a\tau s}{1+\tau s}\right| \left|\frac{\beta}{s^2-\alpha s}\right|$$
$$1 = \frac{\sqrt{1^2+(a\tau\omega)^2}}{\sqrt{1^2+(\tau\omega)^2}} \frac{\beta}{\sqrt{(\omega^2)^2-(\alpha\omega)^2}}$$

By substitution of $\tau^2 = x$, setting $\omega = \frac{1}{\tau\sqrt{a}}$ and using the abc formula, we get:

$$\Leftrightarrow 0 = a^3\beta^2\tau^4 + a\alpha^2\tau^2 - 1$$
$$\tau = 0.0168$$

5 System Modeling with translation

The robot is modeled as a two-degree-of-freedom (2-DOF) non-linear dynamic system, where the key states include the horizontal position and velocity of the robot and the pitch angle and angular velocity of the body as shown in Figure 7. This dynamic model captures both the translational and rotational behavior of the robot under actuation by the motor and gearbox system.

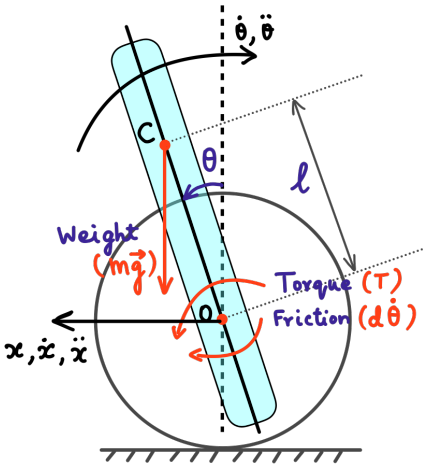

Figure 7: Forces and moments acting on the translating robot

- Input: Voltage U (drives the motor torque T_m).
- Outputs:
  - Position θ (pitch angle) and x (horizontal displacement).
  - Velocities $\dot{\theta}$ and $\dot{x}$.

5.1 Dynamics Equations

$$\text{Linear acceleration, } \ddot{x} = \frac{1}{d_1}\left(aI_O\dot{\theta}^2\sin\theta - a^2 g\sin\theta\cos\theta + T_m\left(\frac{I_O}{r} + a\cos\theta\right)\right) \tag{10}$$
$$\text{Angular acceleration, } \ddot{\theta} = \frac{1}{d_1}\left(-a^2\dot{\theta}^2\sin\theta\cos\theta + am_O g\sin\theta - T_m\left(m_O + \frac{a}{r}\cos\theta\right)\right) \tag{11}$$
$$\text{Motor torque, } T_m = \frac{K_m}{R_m}U - \frac{K_m^2}{R_m}i_{gb}\dot{x}_w + \frac{K_m^2 i_{gb}}{R_m r_w}\dot{\theta} \tag{12}$$

where:

$$
\begin{aligned}
a &= m_B l \\
I_O &= I_2 + m_B l^2 \\
m_{tot} &= m_B + 2m_W \\
m_O &= m_{tot} + J/r^2 \\
d_1 &= I_O m_O - a^2 \cos^2\theta \\
\dot{x}_w &= \dot{x}/r : \text{ Wheel angular velocity (assuming no slip)}
\end{aligned}
$$

Table 1: Parameters for Motor, Gearbox, and Mass Properties and Dimensions

Parameter	Value	Description
K_m	7.2032×10^{-4} Nm/A	Motor constant
R_m	6.4348 Ω	Motor resistance
i_{gb}	49.86	Gearbox ratio
r	0.04 m	Wheel radius
m_B	0.368 kg	Body mass
m_W	0.02 kg	Wheel mass
l	0.01 m	Axle to centre of mass distance
J	2.249×10^{-5} kg · m^2	Wheel moment of inertia
g	9.81 m/s^2	Gravitational acceleration
I_2	2.175×10^{-4} kg · m^2	Body moment of inertia

5.2 Designing State feedback controller and LQR

A **state feedback controller** is employed to stabilize the system. The full state vector—comprising position, velocity, pitch angle, and angular velocity—is fed back and multiplied by a gain vector K to generate the control input voltage. This effectively closes the loop around the nonlinear plant and allows the robot to balance itself.

The feedback gain K is calculated using the **Linear Quadratic Regulator (LQR)** approach. This technique involves linearizing the nonlinear plant model about the upright equilibrium point and then solving the algebraic Riccati equation with appropriately chosen weighting matrices Q and R. The resulting optimal gain K minimizes a cost function that balances state deviations against control effort.

This LQR-based state feedback design provides a systematic and optimal way to regulate the behavior of the balancing robot, and is directly integrated into the simulation framework for evaluating system response under various initial conditions.

The state vector is defined as:

$$\mathbf{x} = \begin{bmatrix} x & \dot{x} & \theta & \dot{\theta} \end{bmatrix}^\top$$

Linearization The nonlinear system is linearized around the upright equilibrium point:

$$\dot{x} = 0, \quad \theta = 0, \quad \dot{\theta} = 0$$

This results in state-space matrices A and B:

$$\dot{\mathbf{x}} = A\mathbf{x} + Bu$$

The A matrix and B vector were calculated to be:

$$A = \begin{bmatrix} 0 & 1.0000 & 0 & 0 \\ 0 & -0.0049 & -1.0647 & 0.0002 \\ 0 & 0 & 0 & 1.0000 \\ 0 & 0.3082 & 157.3685 & -0.0123 \end{bmatrix}; \qquad B = \begin{bmatrix} 0 \\ 0.0090 \\ 0 \\ -0.5705 \end{bmatrix}$$

The **LQR gain** K is computed using MATLAB's inbuilt function `lqr()` [5]:

$$K = \text{lqr}(A, B, Q, R)$$

where the weighting matrices Q and R are set as

$$Q = \text{diag}(1, 0.1, 10, 0.1), \quad R = 0.01$$

This gain is then used for the state feedback controller:

$$u = -K\mathbf{x}$$

6 Experimental Results - Simulation

6.1 1-DOF inverted pendulum system

We model the 1-DOF model as a plant in Simulink and test the performance of two controllers: the Lead Phase controller and the PID controller in the simulation as show in Figure 8. We provide a selector switch to choose between the controllers. There is an option to implement feedback linearization if the pendulum has to be balanced at angles other than 0°.

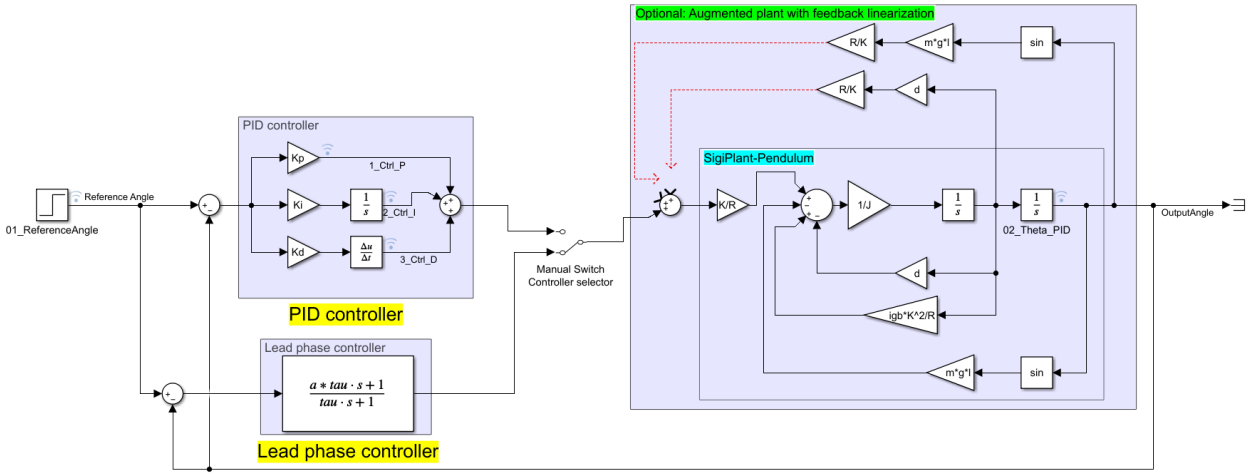

Figure 8: Simulink block diagram of the Simulated inverted pendulum and controllers

6.1.1 Lead Phase Controller

The values of a and τ calculated using equation 9 is implemented. The resulting pitch angle of the inverted pendulum is shown in Figure 9. The blue line signifies the reference angle, which is maintained at 0°, meaning the robot is standing straight. The orange line denotes the output angle of the system. The x-axis corresponds to the time variable in seconds and the y-axis indicates angular displacement in degrees. Initially, the system starts with an offset output angle of 10°. The lead phase controller thus balances the inverted pendulum to 0° within 2 seconds.

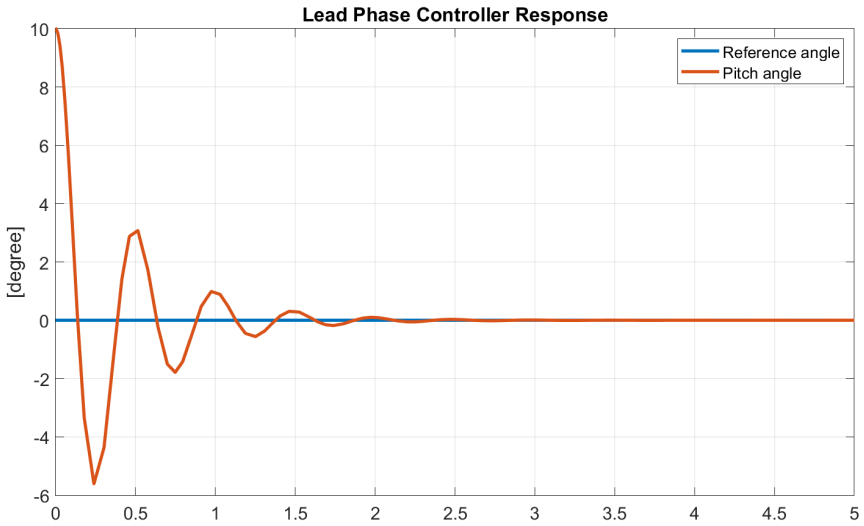

Figure 9: 1-DOF inverted pendulum system - lead phase controller simulation results

6.1.2 PID Controller

By substituting the values of the parameters in the equation for the PID gains using the reference model following approach as written in equation 7, we obtain $K_p = -45.025$, $K_i = 1.6125e + 03$, $K_d = 1.0045$.

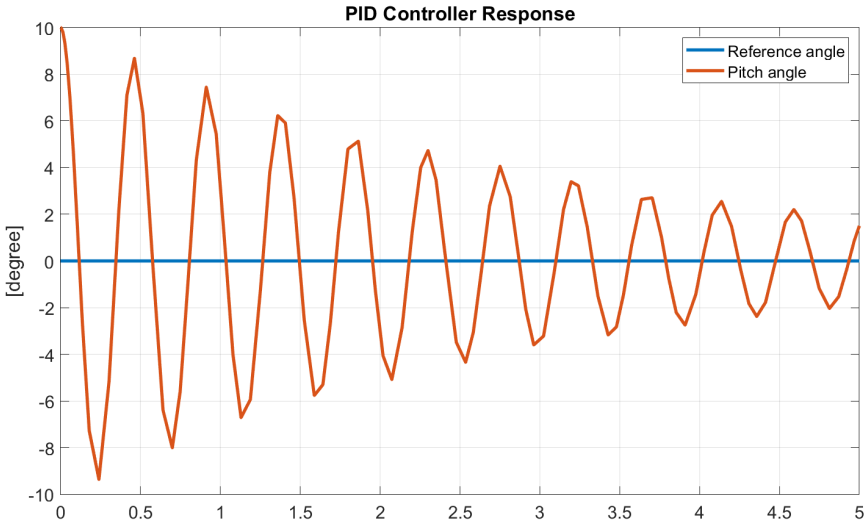

Figure 10: 1-DOF inverted pendulum system - model reference following based PID simulation results

The plot in figure 10 illustrates the response of the PID-controlled system. Initially, the system starts with an offset output angle of 10°. But the PID controller using the gains calculated using the reference model following approach as written in equation is unable to balance the system even after $5s$. This could be due to the fact that the reference model is chosen to be a first order system while our inverted pendulum is a second order system of a form that is not suitable to be controlled using this method. Thus we get large values of K_p and K_i components trying to correct the initial error too aggressively.

We manually tune the PID controller gains using the Ziegler-Nichols method as explained in Section 7.1.1. The simulation results are shown in Figure 11. The performance is improved drastically and the controller can balance the inverted pendulum from an initial angle of 10° to 0° within $2s$.

Unlike the lead phase controller, the PID controller can be easily transformed into a state feedback controller for a higher degree of freedom system. We thus choose the PID for hardware implementation.

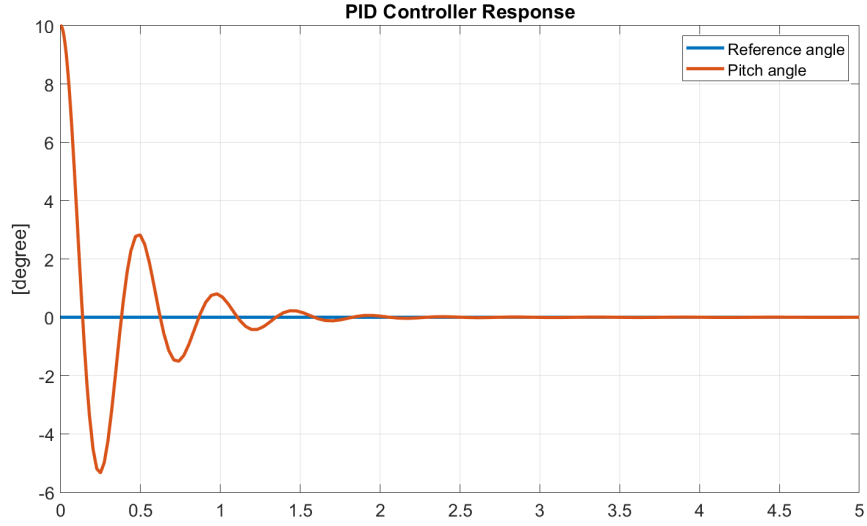

Figure 11: 1-DOF inverted pendulum system - manually tuned PID simulation results

6.2 2-DOF Translating System

In the simulation, the system model is treated as a plant that evolves according to its nonlinear differential equations written in section 5.1. This model is implemented in MATLAB using the `ode45` solver [6], where the plant's parameters, such as motor constant, gear ratio, inertia, and mass distribution, are defined based on real hardware specifications.

The result of the simulation is shown in Figure 12. The initial pitch angle is set as $\theta = 10°$. It can be seen that the controller balances the BalanceBot to near $0°$ within $0.5s$. The linear position goes up to $8cm$ and then the BalanceBot returns towards $0cm$. It is observed that the peak angular velocity is around $-50°/s$. While this works in simulation, it would be too high for practical implementation due to physical constraints such as saturation.

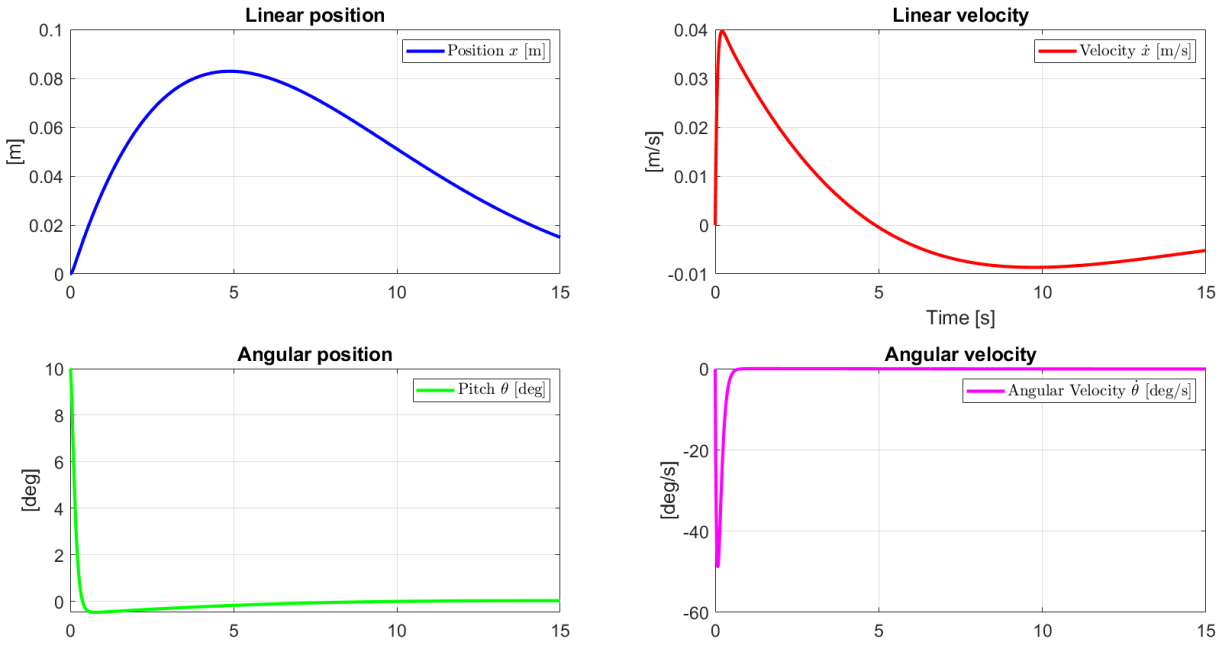

Figure 12: 2-DOF translating system - simulation results

7 Experimental Results - Hardware implementation

To run the model designed in Simulink, the Raspberry Pi has been configured to enable connection through WiFi with the target computer. In Simulink, the Raspberry Pi has been selected as the target hardware, and external mode has been enabled to ensure real-time monitoring, as described in section 2. This allowed the parameters to be adjusted and the live data to be observed without redeploying the model. The Simulink model was then built and compiled into an executable and transferred to the Raspberry Pi. Once deployed, the model was run on the Raspberry Pi's processor through the Monitor and Tune option, which provided real-time data of the model execution. To test the stability of the inverted pendulum, the robot was pinned by its wheels to ensure only one degree of freedom. The system's stability was then tested by physically changing the robot's position to test its ability to return to an upright position.

7.1 1-DOF inverted pendulum system

7.1.1 PID controller: Manual PID tuning

Drawing inspiration from the Ziegler-Nichols method, in which the controller gains K_p, K_i and K_d are progressively adjusted to stabilize the system, the PID controller was tuned incrementally.
K_p was increased until the system started to show regular oscillations. Then, K_i was increased to eliminate the steady-state error, and finally, K_d was introduced to reduce the overshoot and stabilize the system completely. The values obtained with this method are: $K_p = 12, K_i = 0.5, K_d = 1.06$.

The result of the simulation is shown in Figure 13, a screenshot of the Data Inspector in Simulink. It can be seen that when the angle is disturbed, the controller balances the BalanceBot to near $0°$. Due to the gains chosen, the controller output to correct pitch angle θ is larger in magnitude than the controller output to correct the pitch angular velocity $\dot{\theta}$.

7.2 2-DOF translating system

The state feedback controller with the gain matrix obtained using the LQR method worked in simulation. But when it is implemented in hardware, the robot has a jittery motion and balances for about 3-5 seconds before

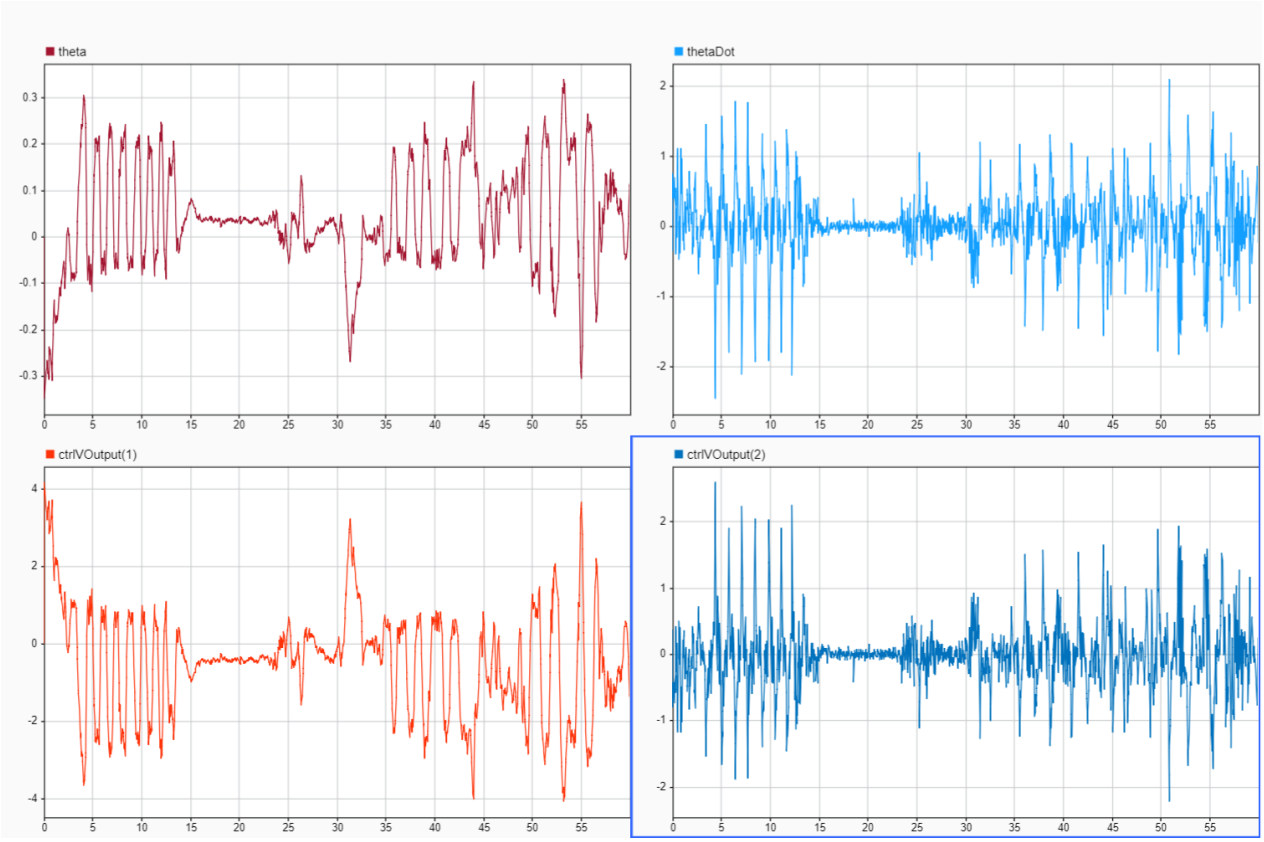

Figure 13: 1-DOF inverted pendulum - hardware implementation results

tipping. Some of the reasons include motor saturation and backlash effect in the gears.

8 Conclusion

The project successfully demonstrated the modeling and control of an inverted pendulum system using classical control techniques. A PID controller was designed and tuned using both the model following approach and the Ziegler-Nichols method. Effective results were observed in both simulation and hardware deployment. While a phase lead controller was also developed to enhance system performance, it proved ineffective in the real-world setup.

To expand the project's scope, a state feedback and an LQR controller were developed for an inverted pendulum with translational motion. These control strategies were implemented and tested in simulation, yielding results in terms of stability and performance. However, attempts to deploy this system on hardware were unsuccessful due to system constraints and time limitations.

9 Future work

The next step is making the BalanceBot to balance smoothly. This could be solved by accounting for the saturation in the controller design. The backlash effect can be solved by adding a time step delay between the observed state and controller output. Once that is done, we can include more degrees of freedom, such as the yaw angle control, which would make the BalanceBot a 4-DOF system.

Code Files

The code written for this project can be found in this GitHub repository: `https://github.com/nithishkgnani/BalanceBot`

References

[1] Pololu. *Balboa 32U4 Balancing Robot Kit (No Motors or Wheels)*. Retrieved April 11, 2025, from `https://www.pololu.com/product/3575`

[2] MathWorks. *Communicate with Hardware Using Connected IO*. Retrieved April 07, 2025, from `https://www.mathworks.com/help/simulink/supportpkg/raspberrypi_ug/connected-io.html`

[3] Raspberry Pi Foundation. *Raspberry Pi 3 Model B+*. Retrieved April 07, 2025, from `https://www.raspberrypi.com/products/raspberry-pi-3-model-b-plus/`

[4] MathWorks. *lsqr*. Retrieved April 07, 2025, from `https://www.mathworks.com/help/matlab/ref/lsqr.html`

[5] MathWorks. *lqr (Control System Toolbox)*. Retrieved April 07, 2025, from `https://www.mathworks.com/help/control/ref/lti.lqr.html`

[6] MathWorks. *ode45 (Solve nonstiff differential equations)*. Retrieved April 07, 2025, from `https://www.mathworks.com/help/matlab/ref/ode45.html`