AIN SHAMS UNIVERSITY
FACULTY OF ENGINEERING

Hybrid Control Systems MCT411s
Fall 2025 / Mechatronics Senior-2

Final Project Report
– Team 16 –

Name	ID
Amr Khaled Taha	2101427
Mostafa Samy Mohammed	2100349
Omar Mohsen Mohamed	2100881
Youssef Ahmed Mahmoud El-Gharably	2101548
Youssef Mostafa Kamel Mohamed	2101019

Submitted to:
Dr. Mohamed Ibrahim Awad
Dr. Mohamed Hamdy
Eng. Hesham Salah
Eng. Mohab Ahmed

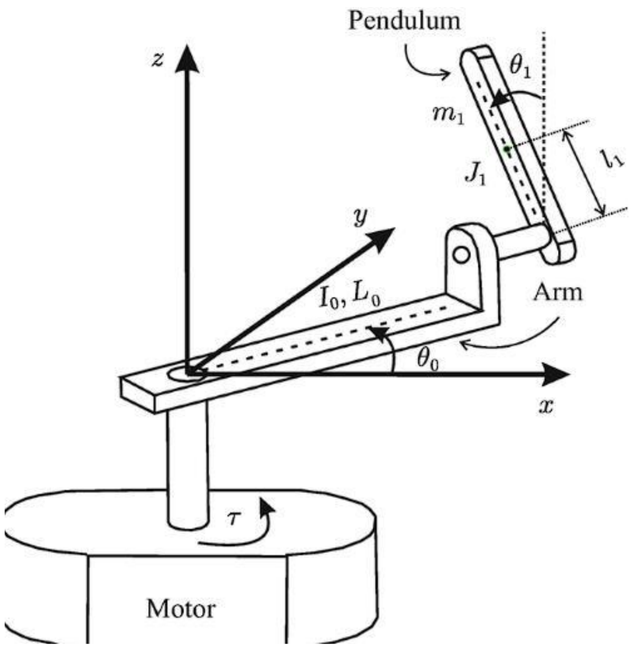
Table of Contents

List of figures

List of tables

Executive Summary

This report documents the design, modeling, and implementation of a Furuta Pendulum (Rotary Pendulum) control system. The primary objective is to stabilize the pendulum in the upright position ($\theta_1 = 0$) and reject external disturbances using a Linear Quadratic Regulator (LQR) combined with an energy-based Swing-Up controller.



- Project Pipeline & Methodology: The project followed a structured pipeline architecture:

1. **System Design:** Selection of mechanical and electrical components including a custom PCB shield for the ESP32 microcontroller and Cytron MDD3A driver.
2. **Parameter Estimation:** Identification of DC motor electrical and mechanical parameters using a Transfer Function (TF) block diagram approach.
3. **Advanced Modeling:** Import of the Solidworks CAD model into MATLAB Simscape Multibody. The plant model was linearized around the upright position ($\theta = 0$) using the identified motor torque as the joint input.

Implementation: Hardware-in-the-Loop (HIL) deployment using the ESP32 platform.

Key Achievements:

- Successfully derived a linearized state-space model from Simscape Multibody.
- Designed and tuned an LQR controller for stabilization and an energy-pumping controller for swing-up.
- Validated performance via HIL simulation showing stable regulation within required settling times.
- Demonstrated robustness against manual disturbances as per project requirements.

Objectives

1- Use pole placement / LQR techniques to control the pendulum.
2- Control the angle θ_1 to be zero. So, the pendulum stands in the upright position.
3- Overcome any disturbances that may affect the pendulum. So, the pendulum should maintain its upright position and not fall.

System Modeling

System Description

The Furuta Pendulum consists of a horizontal rotating arm (actuated by a DC motor) and a non-actuated vertical pendulum. The system is underactuated, having two degrees of freedom but only one control input (motor torque τ).

Parameter estimation

To ensure high-fidelity simulation, the DC motor parameters were estimated rather than relying solely on datasheet values by collecting data from the motor's encoder at different voltages with square signal generator. For validation we have also operated the motor with staircase signal voltage.

Analysis

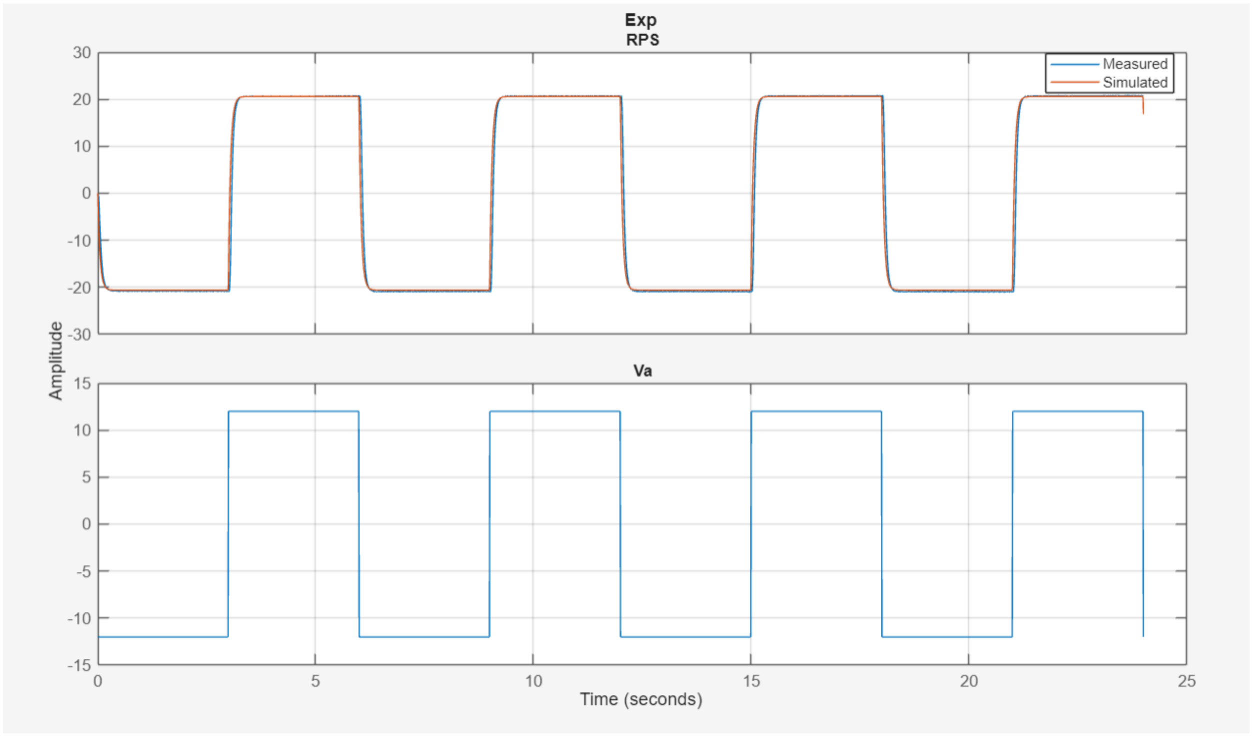


Figure 1. Parameter estimation experiment [1] plot

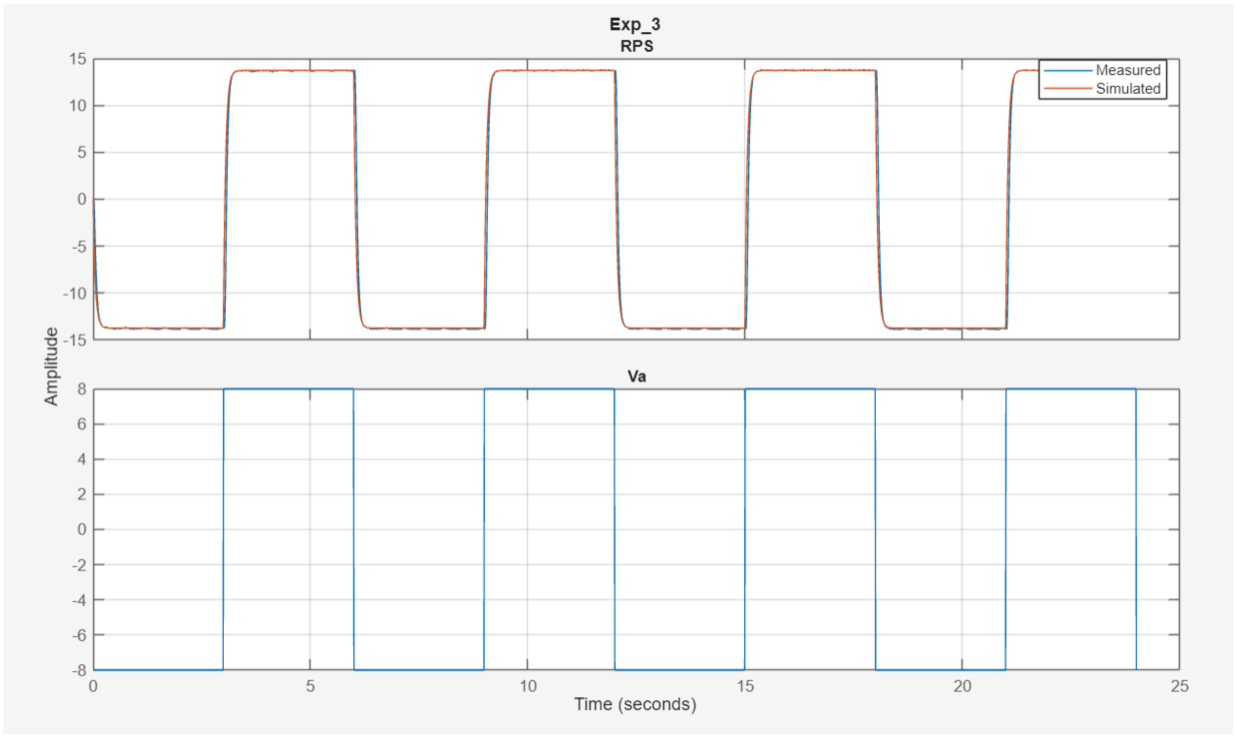


Figure 2. Parameter estimation experiment [2] plot

Validation

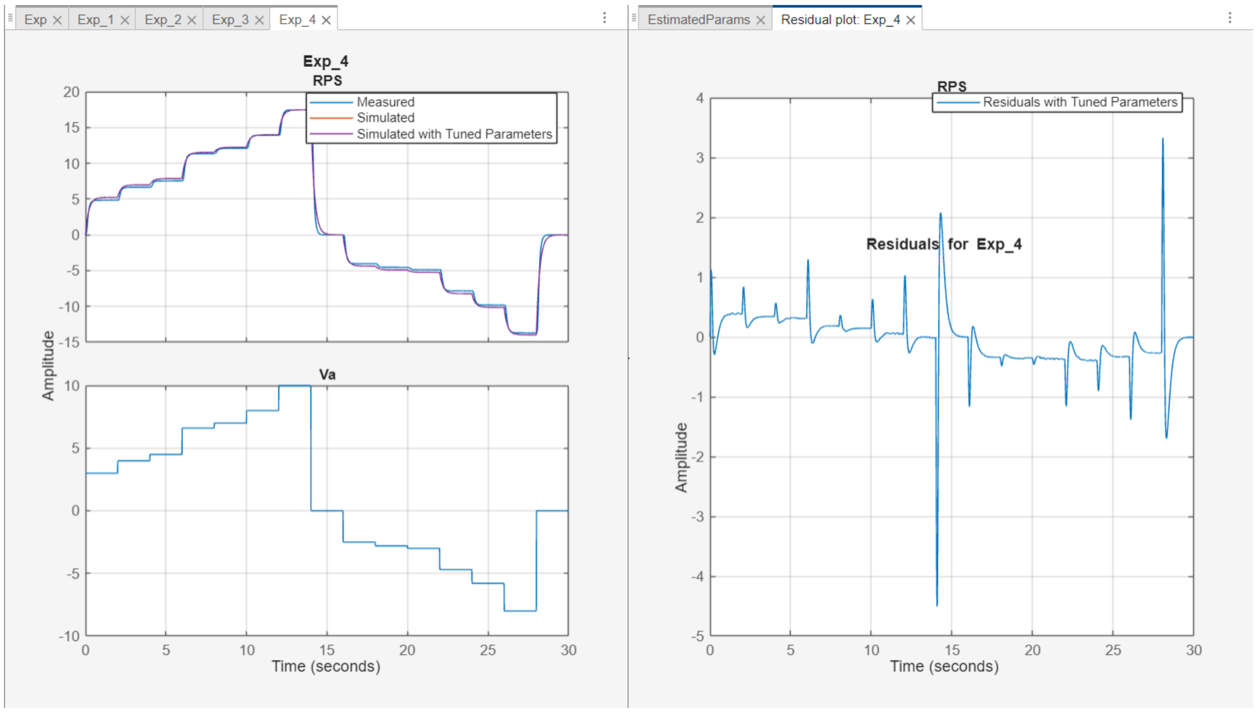


Figure 3. Parameter estimation validation plot

Results

Using a Transfer Function block diagram approach, we identified the following parameters for the motor equivalent circuit and mechanical dynamics:

Table 1. DC Motor parameters

Parameter	Symbol	Estimated Value
Armature Resistance	R_a	$9.86\ \Omega$
Inductance	L_a	$0.00045\ H = 0.45\ mH$
Back-EMF Constant	K_b	$0.5686\ V/(rad/s)$
Torque Constant	K_t	$1.2739\ N \cdot m/A$
Rotor Inertia	J_m	$0.0068\ kg \cdot m^2$
Viscous Friction	b_m	$0.0018\ N \cdot m \cdot s$

Simscape Multibody Model

Instead of manual Euler-Lagrange derivation alone, the system dynamics were modeled by importing the Solidworks CAD assembly into MATLAB Simscape Multibody.

1. **CAD Import**: The Solidworks model was exported to an XML definition and imported into Simulink.

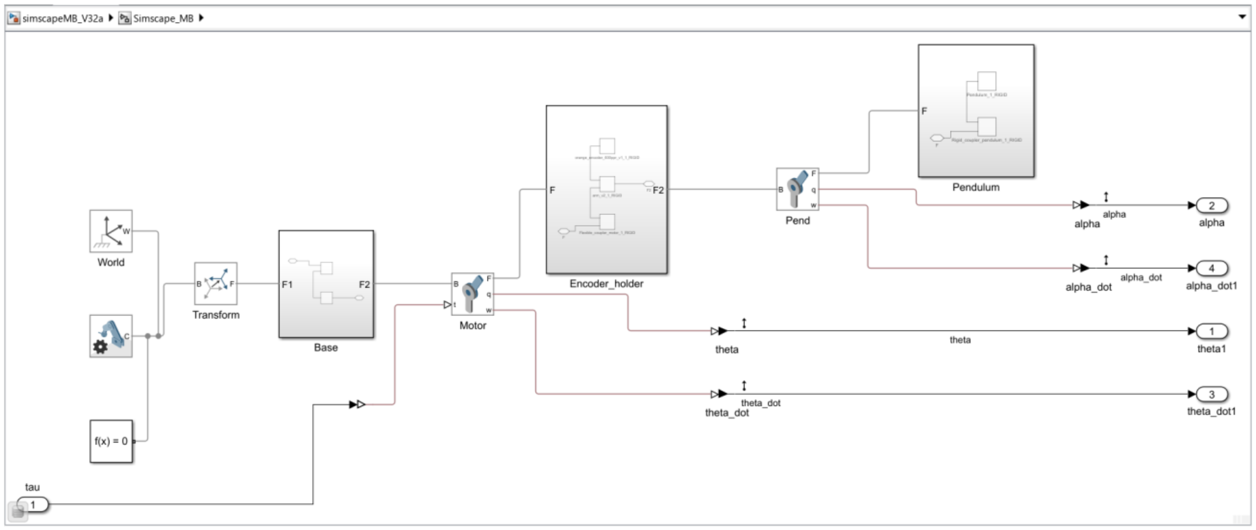


Figure 4. Simscape multibody block diagram of the system

2. **Actuation:** The identified DC motor model (from Section 1.2) was coupled to the arm joint, providing torque τ based on voltage input V.

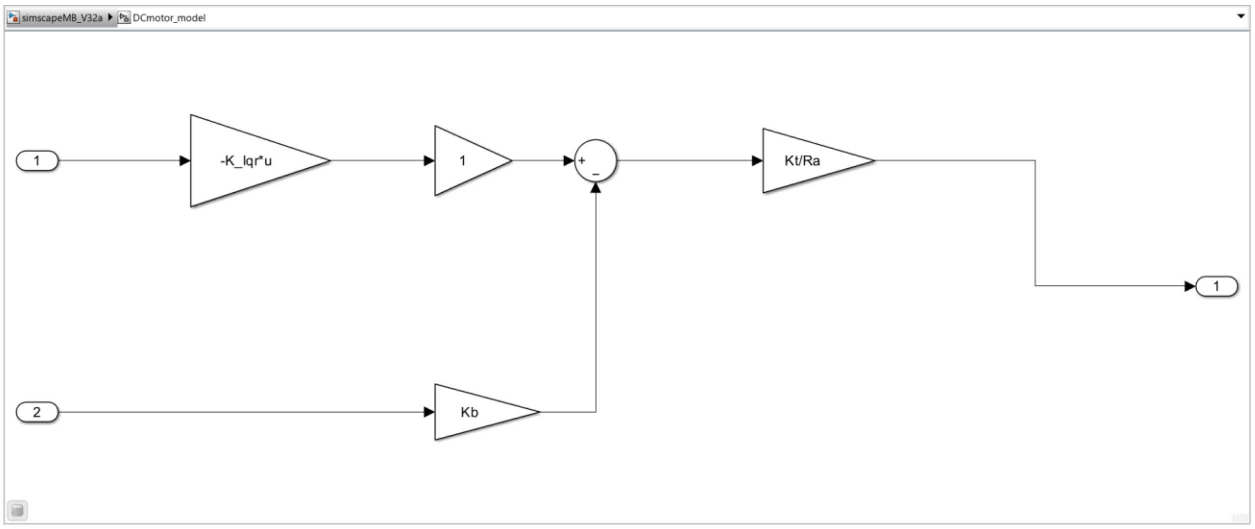


Figure 5. DC motor model

3. **Linearization:** The non-linear Simscape model was linearized around the unstable equilibrium point (Upright: $\theta_{arm} = 0, \theta_{pend} = 0$) to obtain the state-space matrices.

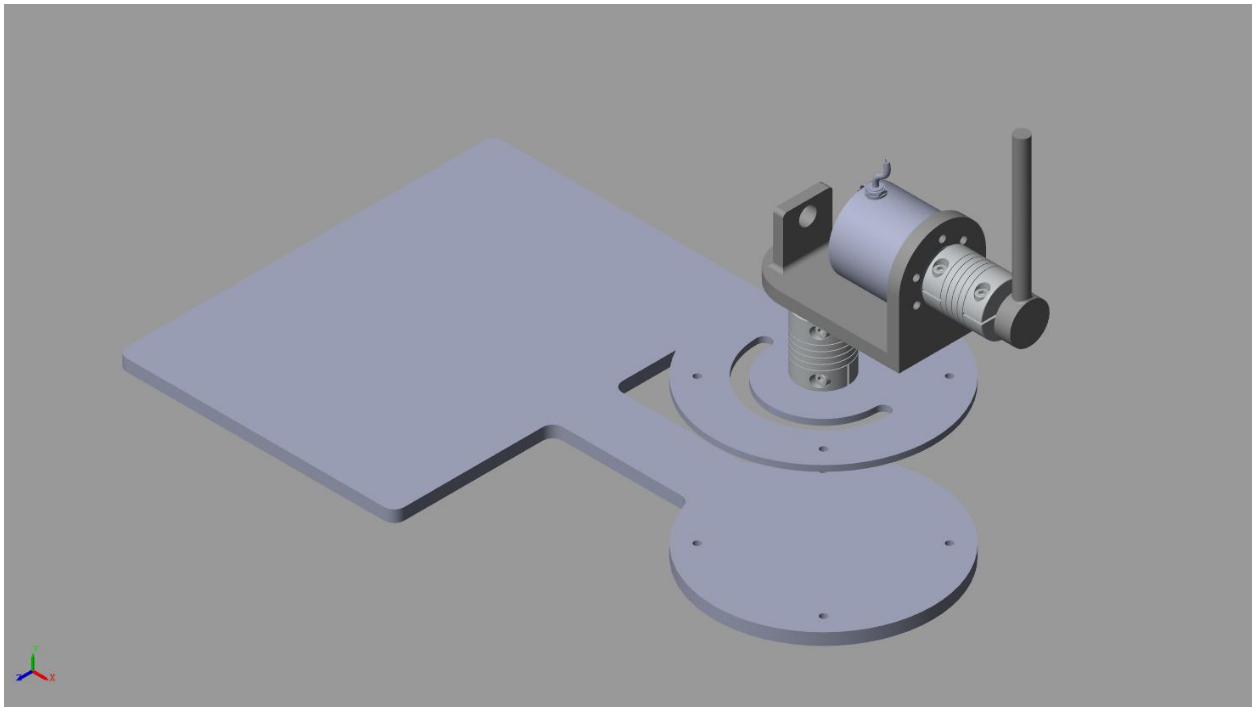


Figure 6. The system model in mechanics explorer

Linearized State-Space Model

The resulting linearized model takes the form

$$\dot{x} = Ax + Bu \tag{1.1}$$

$$y = Cx + Du \tag{1.2}$$

, where:

- State vector

$$x = [\theta_0, \alpha_1, \dot{\theta}_0, \dot{\alpha}_1]^T \tag{1.3}$$

- Output vector

$$u = [\theta_0, \alpha_1] \tag{1.4}$$

- System parameters matrix

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ \sim 0 & a_{32} & a_{33} & \sim 0 \\ \sim 0 & a_{42} & a_{43} & a_{44} \end{bmatrix} \tag{1.5}$$

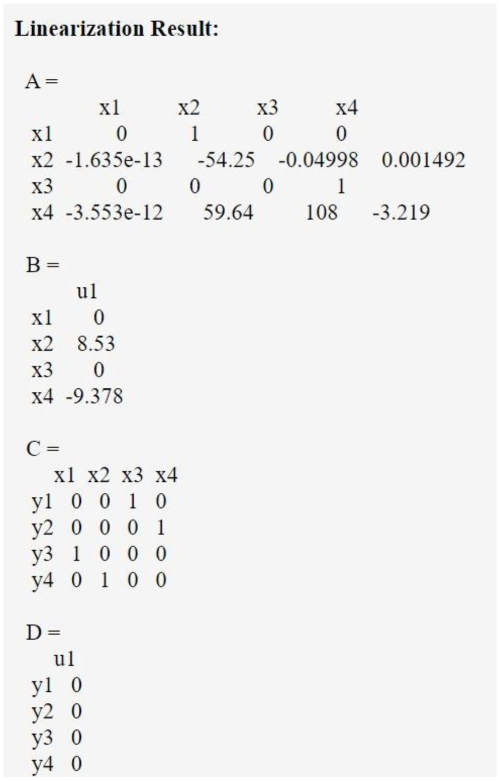
$$B = \begin{bmatrix} 0 \\ b_2 \\ 0 \\ b_4 \end{bmatrix} \tag{1.6}$$

$$C = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \tag{1.7}$$

$$D = 0 \tag{1.8}$$

Results

After linearization of the system at $\theta = 180°$, the result state space model is given below:



```
Linearization Result:

 A =
           x1         x2         x3         x4
  x1        0          1          0          0
  x2  -1.635e-13     -54.25    -0.04998    0.001492
  x3        0          0          0          1
  x4  -3.553e-12      59.64       108      -3.219

 B =
        u1
  x1     0
  x2   8.53
  x3     0
  x4  -9.378

 C =
      x1  x2  x3  x4
  y1   0   0   1   0
  y2   0   0   0   1
  y3   1   0   0   0
  y4   0   1   0   0

 D =
      u1
  y1   0
  y2   0
  y3   0
  y4   0
```

Figure 7. Linearization result state space model

Controller Design

Control Strategy

The control architecture consists of two distinct controllers switched based on the pendulum's state:

1. **Swing-Up Controller:** Brings the pendulum from the downward (stable) position to the upright region $[-\frac{\pi}{8} : \frac{\pi}{8}]$.
2. **Stabilization Controller (LQR):** Balances the pendulum once it is within a small angular threshold of the upright position $[-\frac{\pi}{8} : \frac{\pi}{8}]$.

Stabilization: LQR Design

We utilized the Linear Quadratic Regulator (LQR) method to calculate the optimal gain matrix K that minimizes the cost function:

$$J = \int_0^\infty (x^T Q x + u^T R u)\, dt \tag{2}$$

Weight Selection

The weighting matrices were tuned to prioritize pendulum angle error (θ_1) minimization while respecting motor voltage limits.

$$Q = \begin{bmatrix} Q_\theta & 0 & 0 & 0 \\ 0 & Q_\alpha & 0 & 0 \\ 0 & 0 & Q_{\dot\theta} & 0 \\ 0 & 0 & 0 & Q_{\dot\alpha} \end{bmatrix} = \begin{bmatrix} 10 & 0 & 0 & 0 \\ 0 & 250 & 0 & 0 \\ 0 & 0 & 1.5 & 0 \\ 0 & 0 & 0 & 80 \end{bmatrix} \tag{3}$$

$$R = R_u = 3 \tag{4}$$

Resulting Gain

The calculated state-feedback gain K:

$$K = [k_1\ k_2\ k_3\ k_4] = [-2.2361\ \ 219.4128\ \ -2.0954\ \ 14.9704] \tag{5}$$

Swing-Up Controller

The fundamental idea is based on the **Total Mechanical Energy** (E) of the system. The controller treats the pendulum as an energy management problem rather than a trajectory tracking problem.

1. **Target Energy [E_{ref}]:** You calculate the potential energy the pendulum possesses when it is perfectly upright and stationary. This is your reference energy.
2. **Current Energy [E]:** At every time step, the system calculates the pendulum's current kinetic and potential energy.
3. **Energy Error:** The difference between the two ($E - E_{ref}$) dictates the control action.

Mathematically, the control law u (torque applied to the rotary base) often looks like this:

$$u = k \cdot \left(E - E_{ref}\right) \cdot \dot{\alpha} \cdot \cos\alpha \tag{6}$$

, where:

- k is a tunable gain.
- E is the current total energy.
- $\dot{\alpha}$ is the pendulum angular velocity.
- $\cos\alpha$ ensures the force is applied effectively relative to the arm's rotation (geometric coupling).

Software-in-the-loop (SIL)

To ensure the reliability and safety of the control algorithms prior to physical deployment, a high-fidelity Software-in-the-Loop (SIL) simulation environment was developed. This architecture decouples the control logic from the physical hardware, allowing the exact control code to be tested against a virtual "digital twin" of the rotary inverted pendulum.

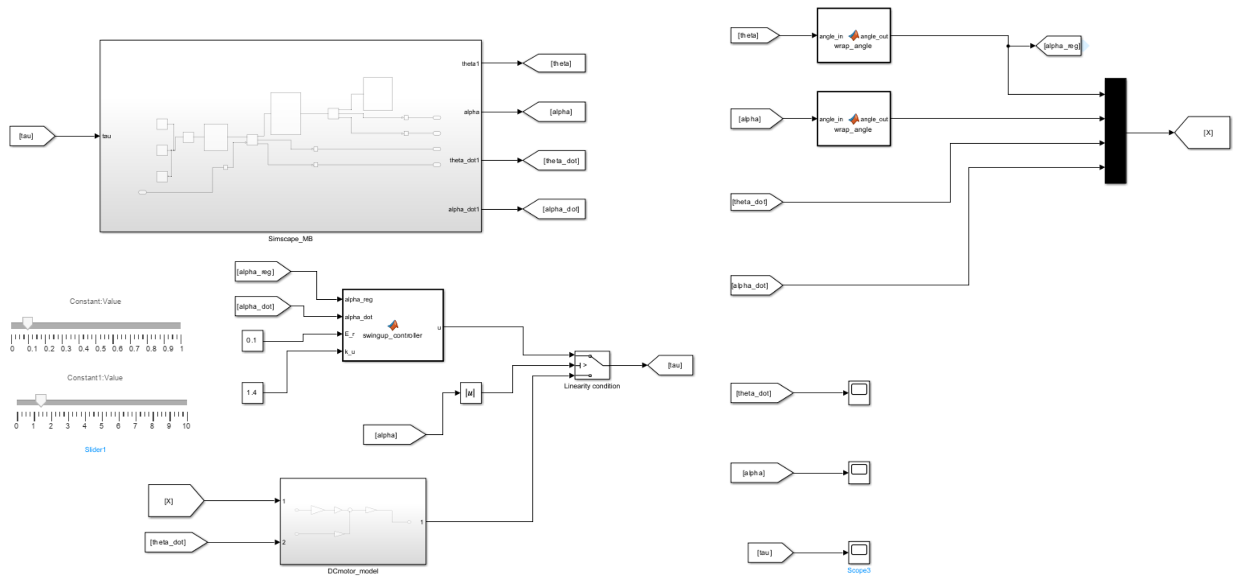


Figure 8. SIL [Software in the loop]

Results & analysis

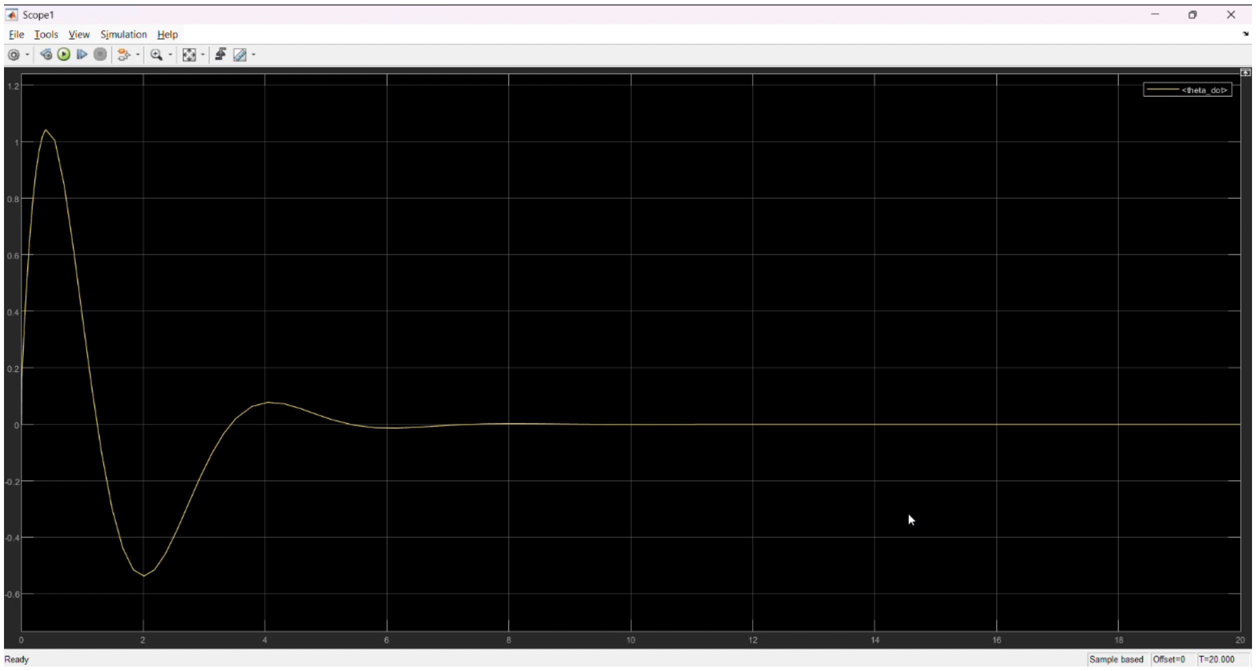


Figure 9. Motor's angular velocity graph

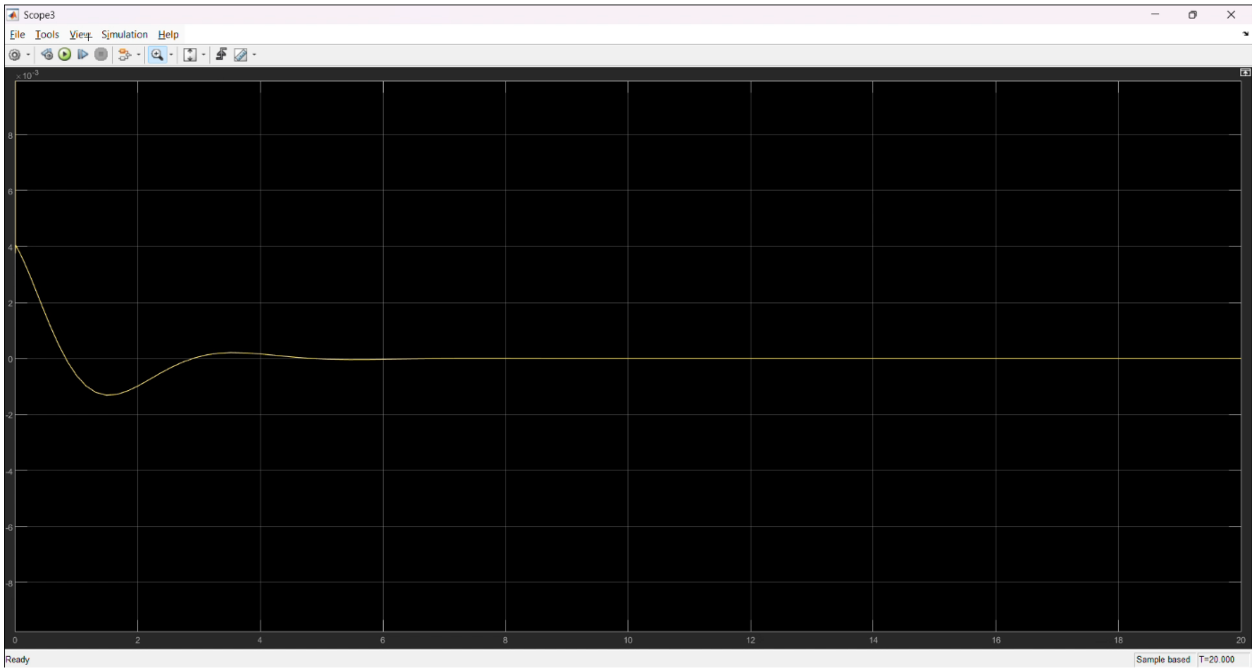


Figure 10. Motor's torque graph

Implementation and HIL Testing

Hardware Architecture

The hardware implementation follows strict wiring and component fixation requirements.

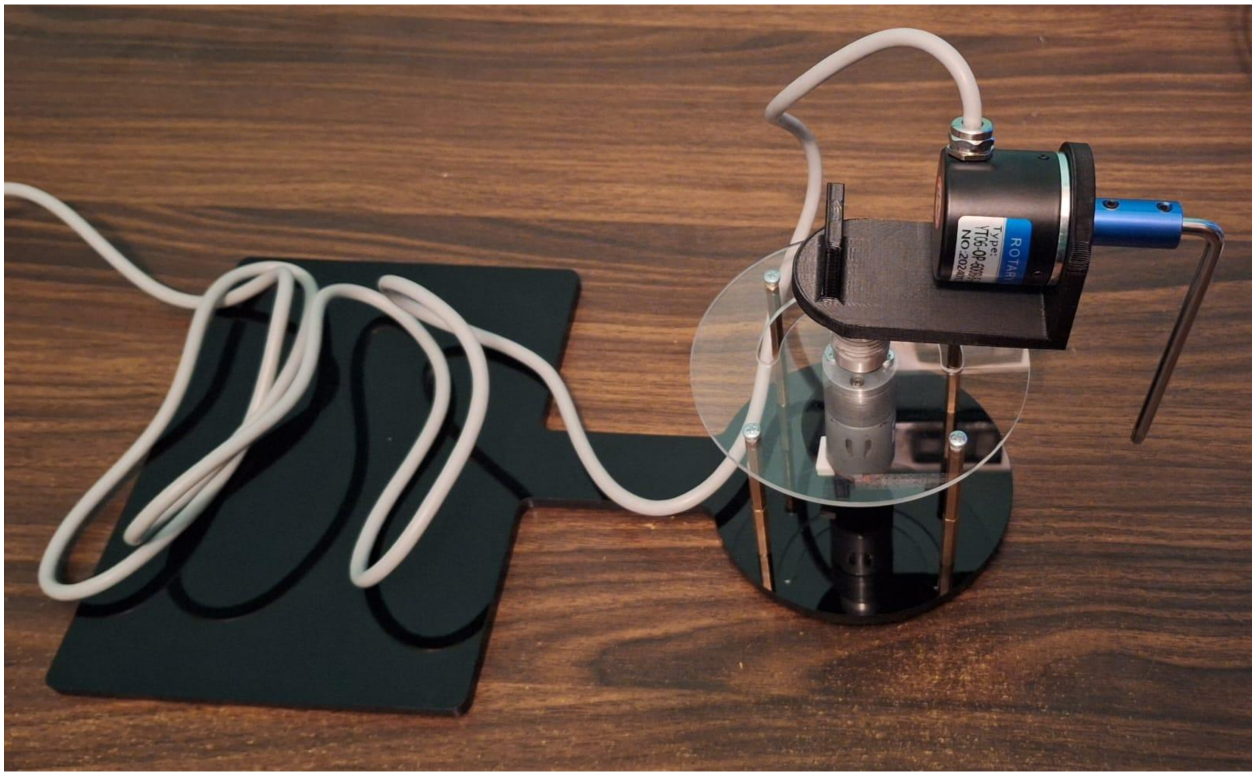


Figure 11. Manufactured rotary inverted pendulum

Electronics and PCB

To ensure reliability and reduce wiring clutter, a custom PCB shield was designed for the microcontroller.

Table 2. Bill of materials [Electrical components]

Component	Model/Specs	Role
Microcontroller	ESP32-DevkitC-38pin	Main controller (HIL Interface)
Motor driver	Cytron MDD3A	High-current DC motor drive
Power supply	12V DC – 5A Supply	System power supply
Rotary incremental encoder	11 PPR [2090 PPR after gearbox in MATLAB Simulink]	
Optical rotary incremental encoder	600 PPR [2400 PPR in MATLAB Simulink]	State feedback (θ_0, θ_1)
Shield	Custom PCB	Interconnects ESP32, Driver, and Encoders

Hardware-in-the-Loop (HIL)

Hardware-in-the-Loop (HIL) implementation for the autonomous Furuta inverted pendulum project. The system is designed to interface an ESP32 microcontroller with

MATLAB/Simulink to achieve real-time balancing control using an LQR (Linear Quadratic Regulator) strategy.

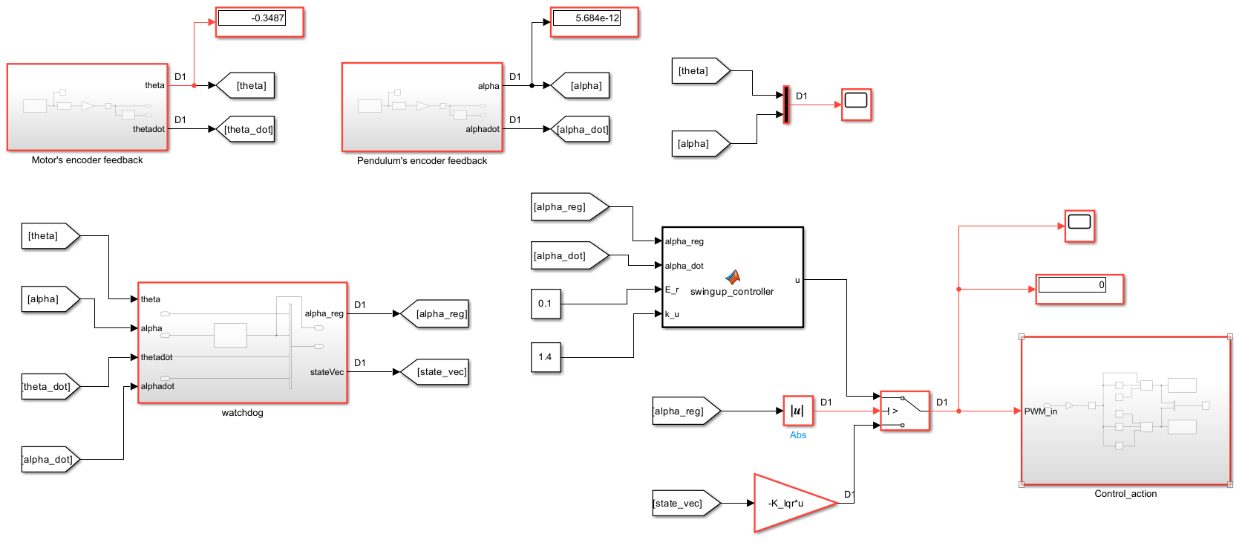


Figure 12. HIL block diagram

1. Encoder Feedback and State Estimation

The first stage of the HIL system focuses on acquiring the physical states of the pendulum. The system utilizes two distinct encoder inputs: one for the DC motor (arm position) and one for the rotary encoder (pendulum angle). Within the Simulink environment, raw encoder counts are received via hardware interruptions and converted into a double precision format. These counts are then scaled by a gain block (deg) and converted to radians (D2R) to represent the angular position, theta.

To derive the angular velocity, theta_dot, the signal is passed through a **Filtered Derivative** block. This block implements a first-order lead-lag transfer function, s/(0.03s+1), which calculates the derivative while simultaneously applying a low-pass filter with a 0.03s time constant. This filtering is critical to mitigate the high-frequency quantization noise inherent in digital encoder signals, ensuring a smooth velocity estimate for the LQR controller.

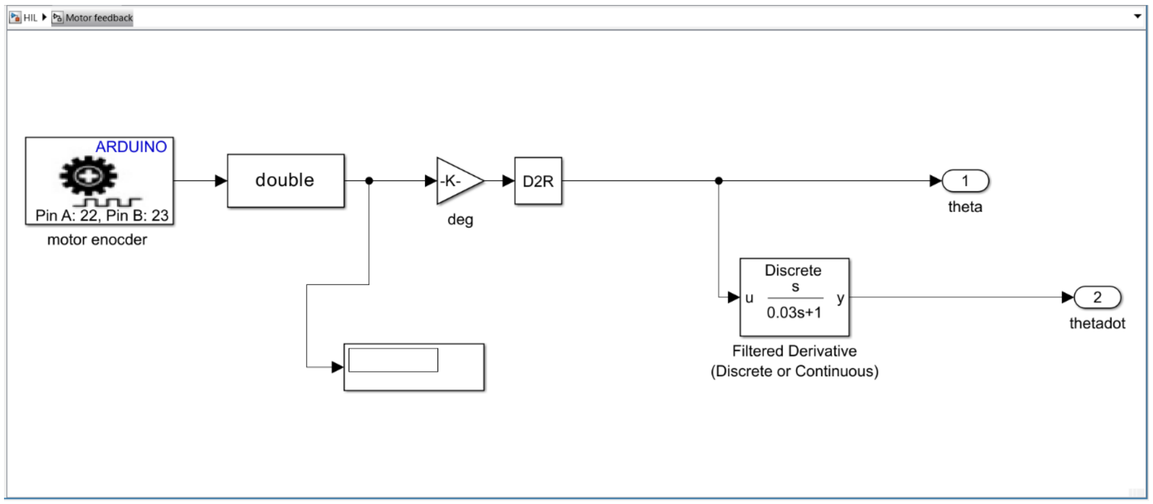


Figure 13. Encoder's feedback

2. Angle Wrapping Logic

To ensure the controller operates within a consistent coordinate system, the angular data is processed through an **Angle Wrapper** section. This is implemented via a MATLAB Function block that constrains the input angle (in radians) to the range [-pi, pi]. The wrapping is achieved using the trigonometric identity atan2(sin(angle_in), cos(angle_in)). This operation prevents the controller from attempting to "unwind" the pendulum if it completes multiple full rotations, maintaining a continuous and bounded state representation for both the arm and the pendulum angles.

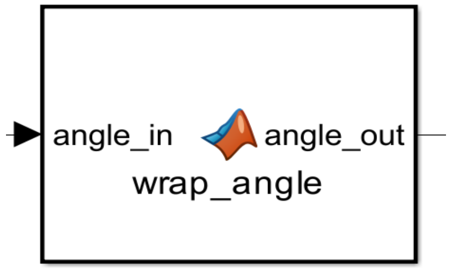


Figure 14. Angle wrapping function block

3. Linearity Condition and LQR Control Switching

The control logic employs a **Linearity Condition** to manage the transition between the inactive state and active balancing. The system monitors the absolute value of the wrapped pendulum angle (|u|). A switch block, labeled "linearity condition," remains at a default output of 0 until the pendulum is manually swung into the upright vertical position.

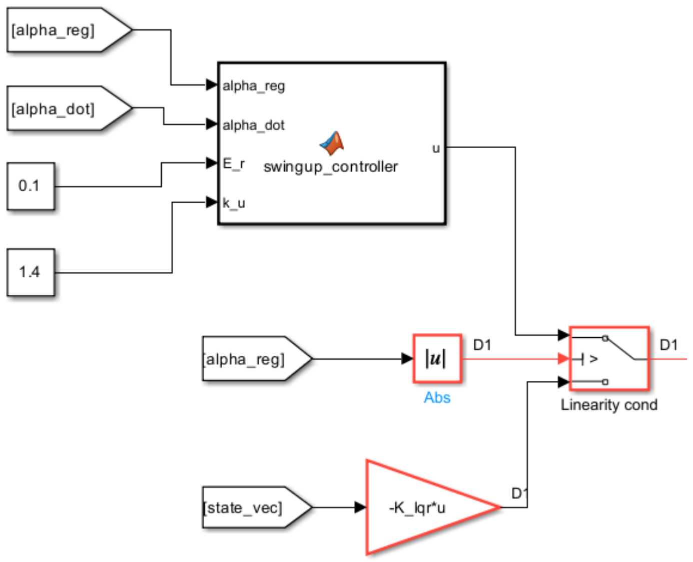
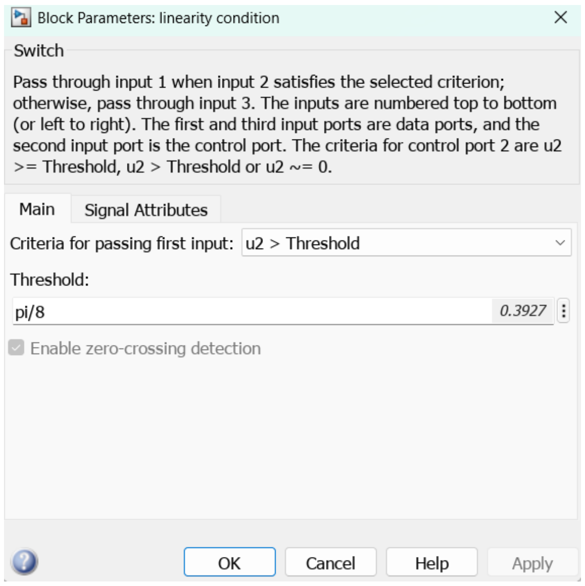


Figure 15. Linearity condition block diagram

When the pendulum angle (alpha) falls within the linear region—defined as being less than pi/8 radians from the vertical—the system switches to the active control state. In this mode, the full state vector of the inverted pendulum is multiplied by the pre-calculated **LQR Gains** (K). This LQR strategy minimizes a cost function to provide the optimal torque required to stabilize the pendulum in the upright position.



Block Parameters: linearity condition

Switch

Pass through input 1 when input 2 satisfies the selected criterion; otherwise, pass through input 3. The inputs are numbered top to bottom (or left to right). The first and third input ports are data ports, and the second input port is the control port. The criteria for control port 2 are u2 >= Threshold, u2 > Threshold or u2 ~= 0.

Main | Signal Attributes

Criteria for passing first input: u2 > Threshold

Threshold: pi/8 0.3927

Enable zero-crossing detection

Figure 16. Linearity condition block

4. Motor Hardware Interface and PWM Generation

The final section translates the calculated control action into physical movement via the motor driver. The control signal is first processed through a gain and a saturation block to ensure the PWM action remains within the hardware limits of the Cytron motor driver. This saturated signal is then split into two paths to control the motor direction and speed.

Using logic gates and switch blocks, the system determines the sign of the control action. If the action is positive, a PWM signal proportional to the absolute value (|u|) is sent to Pin 18, while Pin 19 is held at 0. Conversely, if the action is negative, the PWM signal is routed to Pin 19 while Pin 18 is grounded. These digital outputs are sent directly to the ESP32 pins to drive the motor, completing the real-time HIL control loop.

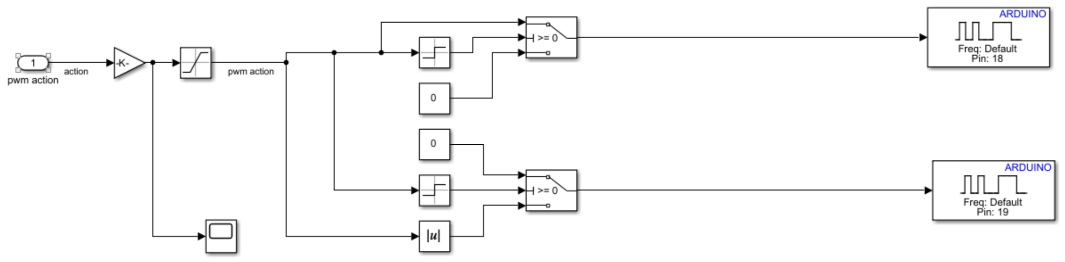
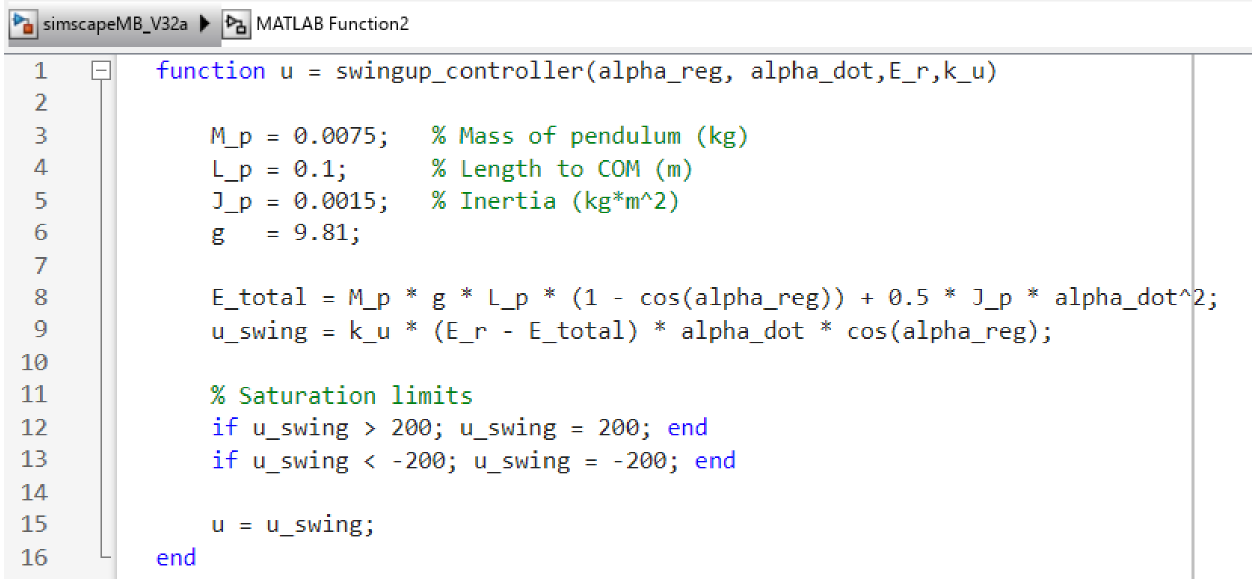


Figure 17. Hardware interface and PWM generation

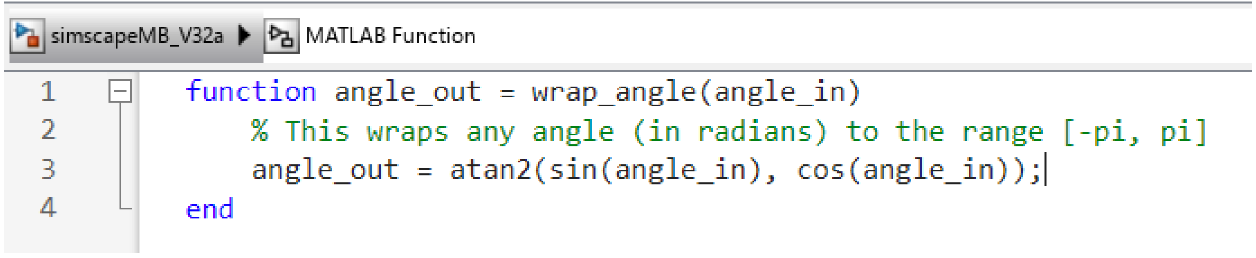
Appendix

Swing-up controller function



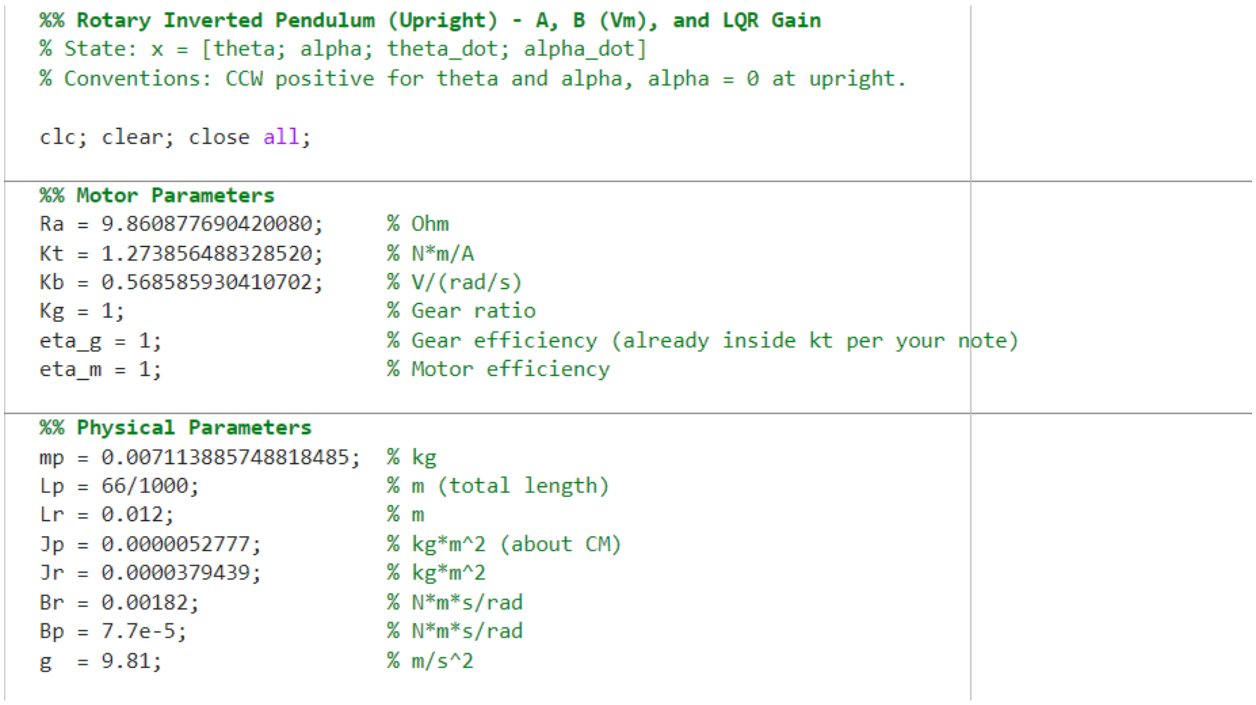
```
function u = swingup_controller(alpha_reg, alpha_dot,E_r,k_u)

    M_p = 0.0075;   % Mass of pendulum (kg)
    L_p = 0.1;      % Length to COM (m)
    J_p = 0.0015;   % Inertia (kg*m^2)
    g   = 9.81;

    E_total = M_p * g * L_p * (1 - cos(alpha_reg)) + 0.5 * J_p * alpha_dot^2;
    u_swing = k_u * (E_r - E_total) * alpha_dot * cos(alpha_reg);

    % Saturation limits
    if u_swing > 200; u_swing = 200; end
    if u_swing < -200; u_swing = -200; end

    u = u_swing;
end
```

Angle wrapping function



```matlab
function angle_out = wrap_angle(angle_in)
    % This wraps any angle (in radians) to the range [-pi, pi]
    angle_out = atan2(sin(angle_in), cos(angle_in));
end
```

LQR code



```matlab
%% Rotary Inverted Pendulum (Upright) - A, B (Vm), and LQR Gain
% State: x = [theta; alpha; theta_dot; alpha_dot]
% Conventions: CCW positive for theta and alpha, alpha = 0 at upright.

clc; clear; close all;

%% Motor Parameters
Ra = 9.860877690420080;      % Ohm
Kt = 1.273856488328520;      % N*m/A
Kb = 0.568585930410702;      % V/(rad/s)
Kg = 1;                      % Gear ratio
eta_g = 1;                   % Gear efficiency (already inside kt per your note)
eta_m = 1;                   % Motor efficiency

%% Physical Parameters
mp = 0.007113885748818485;   % kg
Lp = 66/1000;                % m (total length)
Lr = 0.012;                  % m
Jp = 0.0000052777;           % kg*m^2 (about CM)
Jr = 0.0000379439;           % kg*m^2
Br = 0.00182;                % N*m*s/rad
Bp = 7.7e-5;                 % N*m*s/rad
g  = 9.81;                   % m/s^2
```

```
%% ===== Linearized model about upright (alpha ~ 0) with TORQUE input =====
% From workbook EOM (linearize sin(alpha)~alpha, cos(alpha)~1, drop products)
% M * [theta_ddot; alpha_ddot] = [ tau - Br*theta_dot ;
%                                  +0.5*mp*Lp*g*alpha - Bp*alpha_dot ]
%
% where:
Jr_t = Jr + mp*Lr^2;                % effective arm inertia term at alpha=0
Jp_t = Jp + 0.25*mp*Lp^2;           % effective pendulum inertia term
Jx   = 0.5*mp*Lp*Lr;                % coupling term

detM = Jr_t*Jp_t - (Jx^2);

% A,B for torque input u = tau
A_tau = zeros(4,4);
B_tau = zeros(4,1);

A_tau(1,3) = 1;
A_tau(2,4) = 1;

% theta_ddot = (1/detM)*( Jp_t*(tau - Br*theta_dot) + Jx*(0.5*mp*Lp*g*alpha - Bp*alpha_dot) )
A_tau(3,2) = (Jx * (0.5*mp*Lp*g)) / detM;   % alpha term
A_tau(3,3) = -(Jp_t * Br) / detM;           % theta_dot term
A_tau(3,4) = -(Jx   * Bp) / detM;           % alpha_dot term
B_tau(3)   = (Jp_t) / detM;                 % tau term

% alpha_ddot = (1/detM)*( Jx*(tau - Br*theta_dot) + Jr_t*(0.5*mp*Lp*g*alpha - Bp*alpha_dot) )
A_tau(4,2) = (Jr_t * (0.5*mp*Lp*g)) / detM; % alpha term
A_tau(4,3) = -(Jx   * Br) / detM;           % theta_dot term
A_tau(4,4) = -(Jr_t * Bp) / detM;           % alpha_dot term
B_tau(4)   = (Jx) / detM;                   % tau term

%% ===== Convert TORQUE input to VOLTAGE input Vm (workbook Eq. 2.4 style) =====
% tau = (eta_g*Kg*eta_m*kt/Rm)*Vm - (eta_g*Kg^2*eta_m*kt*km/Rm)*theta_dot
KtVm = (eta_g*Kg*eta_m*Kt)/Ra;           % tau per Vm
Kbemf = (eta_g*(Kg^2)*eta_m*Kt*Kb)/Ra;   % tau per theta_dot (back-emf term)

A = A_tau;
B = B_tau;

% Add back-emf term into A via the same pattern used in workbook snippet:
A(3,3) = A(3,3) - Kbemf * B(3);
A(4,3) = A(4,3) - Kbemf * B(4);

% Scale B to convert input from tau to Vm:
B = KtVm * B;

% Output matrices (if you need them)
C = [1 0 0 0;
     0 1 0 0];
D = [0;0];
```

```
%% Display A, B
disp('A (Vm input) ='); disp(A);
disp('B (Vm input) ='); disp(B);
```

```
%% ===== LQR design =====
% Tune Q and R as you like (start here, then adjust).
Q = diag([ 1e-5,  2000,   0.3,  0.05 ]);   % [theta, alpha, theta_dot, alpha_dot]
R = 0.3;                                    % voltage effort penalty

% MATLAB returns K_lqr for u = -K_lqr*x
K_lqr = lqr(A,B,Q,R);

% If you implement u = K*(xd - x), use:
K_use = -K_lqr;

disp('K_lqr (for u = -K_lqr*x) ='); disp(K_lqr);
disp('K_use (for u =  K_use*(xd - x)) ='); disp(K_use);

% Quick sign check (you wanted [+ - + -] on K_use):
disp('sign(K_use) ='); disp(sign(K_use));
```