UCLouvain

LINMA2671 - Advanced Control and Applications

Biomedical/Applied Mathematics Engineering

Lab#2 : Control of a Pendubot - UCLouvain [EN]

Authors:

Afonso Araújo afonso.soares@student.uclouvain.be
Colin Cloos colin.cloos@student.uclouvain.be

2023/2024 – 1st Semester

Contents

Introduction

In the following report the group discusses three different controllers for the purpose of controlling the nonlinear chaotic system that is a double inverted pendulum.

A structure for the report was proposed on the laboratory notice. The group took some liberties with this structure in order for the report to read as smoothly and less image-cluttered as possible, due to some data being missing.

The first chapter, 1, goes over the continuous and discrete time models, the second chapter, 2, goes over the mathematics behind controller's C_1 and C_2 as well as their software implementation, the third chapter, 3, goes over the results and trials for the closed loop performance of C_1 and the fourth chapter gives a detailed explanation of controller C_3 as well as its results.

The double inverted pendulum system is called Pendubot.

1 Linearized Model

The Pendubot's laboratory notice provides the nonlinear model 1 of the physical plant.

$$\dot{x} = f(x) + g(x)u = F(x,u) \tag{1}$$

where $x = [\alpha_1, \alpha_2, \dot{\alpha_1}, \dot{\alpha_2}]^T$, $u = \tau$ and f, g are defined in the laboratory notice.

In order to implement the controller C_1, the nonlinear system 1 is linearized around an equilibrium (x_e, u_e) to obtain 2.

$$\dot{x} \approx f(x_e) + g(x_e)u_e + A(x - x_e) + B(u - u_e) = A(x - x_e) + B(u - u_e) \tag{2}$$

where $A = \frac{\partial F(x,u)}{\partial x}|_{(x_e,u_e)}$ and $B = \frac{\partial F(x,u)}{u}|_{(x_e,u_e)}$.

Defining the new state $z := x - x_e$ and the new input $v := u - u_e$ gives the continuous time linear model 3.

$$\begin{cases} \dot{z} = Az + Bv \end{cases} \tag{3}$$

The matrices A and B are obtained from the partial derivatives of the functions f and g such that :

$$A = \frac{\partial f(x,u)}{\partial x}|_{(x_e,u_e)} + \frac{\partial f(x,u)}{\partial x}|_{(x_e,u_e)}u \tag{4}$$

$$B = g(x_e, u_e) \tag{5}$$

The controller C_1 needs to be designed to stabilize the system around the equilibrium $x_e = \left(\frac{4\pi}{5}, \pi, 0, 0\right)$. So we need to obtain the linear model around this equilibrium. Utilizing symbolic calculus in MATLAB allows us to find the torque at this equilibrium, $u_e = 1.1678\,\text{Nm}$, and the numerical expressions of 4 and 5 such that:

$$A = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 65.8721 & -42.5458 & 0 & 0 \\ -113.9390 & 131.8745 & 0 & 0 \end{bmatrix}, \quad B = \begin{bmatrix} 0 \\ 0 \\ 40.9835 \\ -70.8893 \end{bmatrix}$$

1.1 Continuous Time Linear Model Validation

In order to validate the linearized system 3, we test the system in open loop. However, since the system is only stable around the equilibrium $x_e = 0$ and $u_e = 0$ in open loop, the validation of the linearized system is conducted around this equilibrium. The corresponding matrices A and B are:

$$A = \begin{bmatrix} 0 & 0 & 1.0000 & 0 \\ 0 & 0 & 0 & 1.0000 \\ -92.3805 & 53.0791 & 0 & 0 \\ 175.7041 & -159.2373 & 0 & 0 \end{bmatrix}, \quad B = \begin{bmatrix} 0 \\ 0 \\ 46.4992 \\ -88.4397 \end{bmatrix} \tag{6}$$

Figure 1 illustrates the comparison between the real model and the simulated model in continuous time. Notably, the linearized model closely follows the behavior of the simulator, depicted by the red dashed line, under the absence of disturbances. The alignment of the red dashed line with the black solid line demonstrates the close agreement between the linearized model and the simulator.

On the other hand, the real system, represented by the blue dotted line, exhibits deviations from both the simulator and the linearized model. This disparity is evident in the Figure 1, highlighting the impact of real-world factors not accounted for in the simplified models.

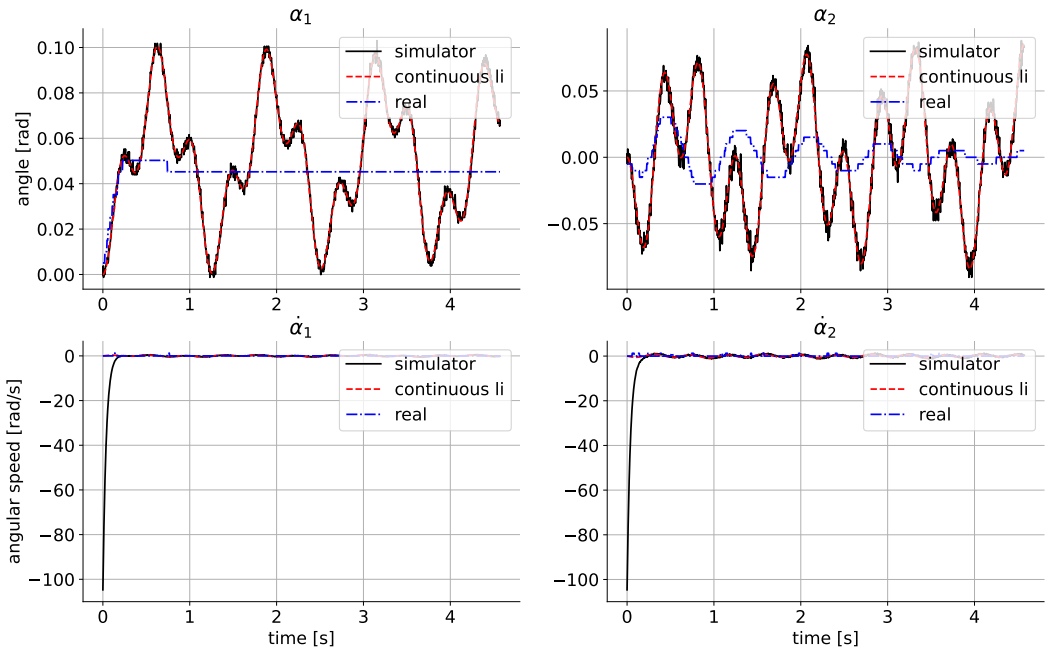


Figure 1: Linear Model Validation in Continuous Time

1.2 Discrete-Time Linear Model Validation

To adapt the continuous-time linear state-space representation of the system to discrete-time dynamics, we first construct the linear system using the state-space matrices A and B in MATLAB, as shown in Figure 6.

```
sys = ss(A_e, B_e, eye(4), 0);
```

Subsequently, we discretize it with a specified time step ($dt = 0.004$ s) using the zero-order hold (`'zoh'`) discretization method:

```
sys_d = c2d(sys, dt, 'zoh');
```

The resulting discretized system, denoted as `sys_d`, is characterized by the discretized matrices A_d and B_d. These matrices are given by:

$$A_d = \begin{bmatrix} 0.9993 & 0.0004 & 0.0040 & 0.0000 \\ 0.0014 & 0.9987 & 0.0000 & 0.0040 \\ -0.3693 & 0.2122 & 0.9993 & 0.0004 \\ 0.7023 & -0.6366 & 0.0014 & 0.9987 \end{bmatrix} \quad B_d = \begin{bmatrix} 0.0004 \\ -0.0007 \\ 0.1859 \\ -0.3535 \end{bmatrix}$$

These matrices capture the discretized dynamics of the system with a time step of 0.004 seconds. Notably, this discretization is conducted while maintaining the system's stability around the equilibrium point $x_e = 0$ and $u_e = 0$.

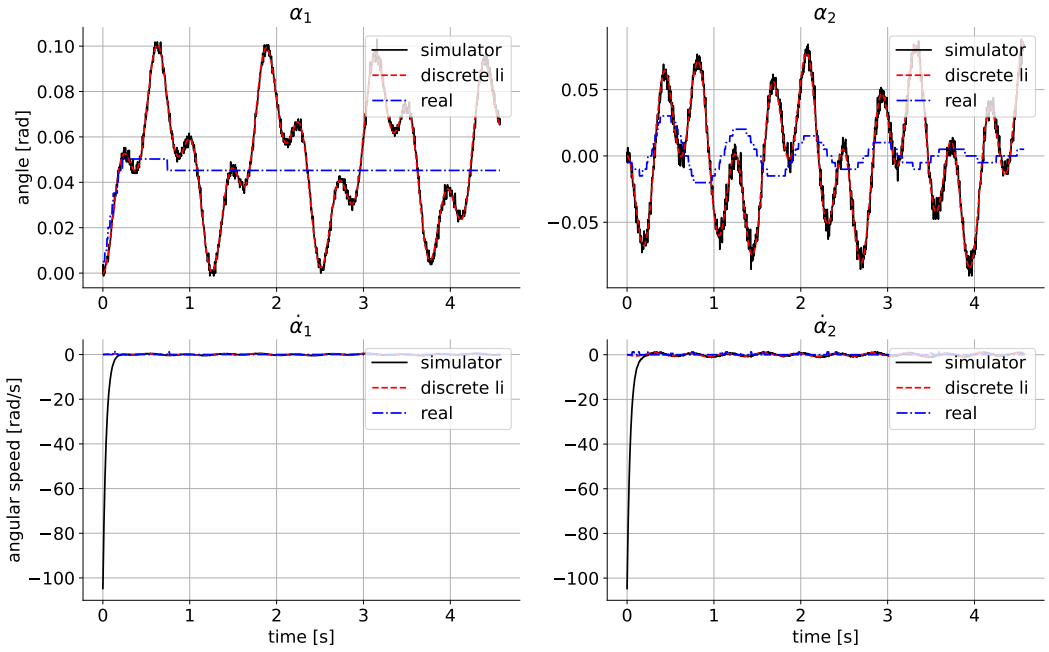


Figure 2: Linear Model Validation in Discrete Time

Figure 2 presents a comparison between the real model and the simulated model in discrete time. Remarkably, the obtained results closely resemble those depicted in Figure 1. The similarities between the two figures suggest consistent behavior in both continuous and discrete time simulations. Therefore, the conclusions drawn from the continuous-time analysis, as discussed in the context of Figure 1, remain applicable to the discrete-time scenario as well.

2 Controllers C_1 and C_2

2.1 Controller C_1

The controller C_1 requirements are for it to be a "high-performance and fast controller" which means that :

- Fast tracking, the group defined it as response time smaller than $3s$.
- Maximum overshoot of 20%.
- No static error.

In order to achieve precise control of the Pendubot, we have chosen to implement an LQR (Linear-Quadratic-Regulator) control law. The primary advantage of LQR lies in its capability to minimize a quadratic cost function. Additionally, LQR is designed offline, implying that the control law is computed beforehand and does not require real-time computation. This is particularly advantageous for our application, as controlling a Pendubot necessitates very short computation times to minimize delays. The control law is applied to the linearized system given by Equation (3). We opted to utilize the continuous-time linear model based on experimentation with both the simulator and the real plant. In our observations, both the continuous and discrete-time linear models demonstrated similar performances, with slightly superior results for the continuous-time model. Notably, the control law obtained with the continuous-time model exhibited less oscillation at the equilibrium point.

To design an LQR control law, the following steps are followed :

1. **Formulate the Cost Function:** Define the cost function as:
$$J = \int_0^\infty (z^T Q z + v^T R v)\, dt$$
where Q and R are positive definite weighting matrices that represent the cost of state and control input, respectively.

2. **Define the LQR Control Law:** The LQR control law is expressed as:
$$v = -Kz$$
where K is the gain matrix obtained by solving the algebraic Riccati equation:
$$A^T P + PA - PBR^{-1}B^T P + Q = 0$$
and P is the positive definite solution to the Riccati equation.

The LQR control law minimizes the cost function, providing an optimal control strategy for the given linearized system, and aligns with the objective of achieving precise control for the Pendubot.

The controller C_1 is implementerd using a Linear–Quadratic Regulator (LQR) control law obtained with the linearized model 3. The control law of an LQR is given by :

$$v = -kz \iff u = -kz + u_e \tag{7}$$

where $k \in \mathbb{R}^{1\times 4}$ and u_e is the torque applied at the steady state x_e. For the data of the design trials of C_1, reference chapter 3.1.

2.2 Simulator and Real Plant Implementations of C_1

Appendix A, B and C shows the MATLAB code to obtain C_1 as well as the Simulink environment implementation of C_1.

When it comes to the real implementation on the LabVIEW environment, the group copied 100% of the MATLAB code as indicated in B to the laboratory software.

2.3 Controller C_2

In order to eliminate the static error of the controller C_1 within the linear state-space system represented by matrices A and B, it is necessary to add an integrator to the LQR control law. The objective is to enhance the controller's capability to regulate the system at a desired reference point without static error. To achieve this, an additional state variable is introduced, representing the integral of the system's error over time. Consequently, the augmented state-space matrices are extended to accommodate this new state, resulting in augmented A and B matrices. This modification effectively transforms the LQR control law into an Integral LQR (ILQR) control law, providing the ability to drive the steady-state error to zero.

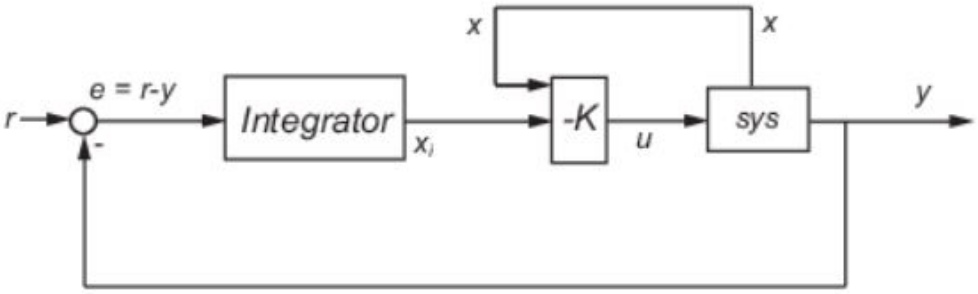


Figure 3: LQI controller block diagram from [1].

Figure 3 represents such a controller. To implement it, the choice of which state is considered as y is crucial. Since the static error is more important for x_1, we should take $y = x_1$. Another approach would be to integrate the errors of both states, x_1 and x_2, resulting in a more complex control law that includes two integrators. In the case of only one integrator, the augmented system and the control law would be:

$$\begin{bmatrix} \dot{x} \\ \dot{x}_i \end{bmatrix}^{5\times1} = \begin{bmatrix} (A - KI)^{4\times4} & 0 \\ v & -K \end{bmatrix}^{5\times5} + \begin{bmatrix} x \\ x_i \end{bmatrix}^{5\times1}$$

where $v = [-1 \quad 0 \quad 0 \quad 0]$ and I the 4×4 identity matrix.

The group however did not have the time to implement fully controller C_2, opting to maximize time spent on the real system for both controllers C_1 and C_3.

3 Closed-Loop Performances of C_1

In this section, the group compares the performance over two different design trials utilizing the Simulink environment C to design the Q and R matrices.Both of these design trials were computed for the closed-loop nonlinear model. In the end, the group shows the real plant validation for C_1.

The group decided to present the data in this way so that the reader could see the mathematical proofs of the (LQR) trials and the subsequent plots all at once.

3.1 Controller C_1 : Simulator Trial#1

In the design of the Linear-Quadratic-Regulator (LQR) control law for our linearized system, an alternative set of weighting matrices Q and R was considered:

$$Q = \begin{bmatrix} 2 & 0 & 0 & 0 \\ 0 & 10 & 0 & 0 \\ 0 & 0 & 0.1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad R = 0.3$$

The strategy for tuning Q is to primarily penalize the errors in states 1 and 2. This is achieved by setting $Q_{1,1}$ to 2 and $Q_{2,2}$ to 10, ensuring a more significant penalty for these states. This choice is made to prevent static errors. To facilitate fast tracking without overly penalizing energetic yet reactive commands, the value of R is set to a low value.

The resulting K matrix is given by:

$$K = \begin{bmatrix} -1.4341 & -21.7625 & -2.0257 & -3.1800 \end{bmatrix}$$

The simulated results are illustrated in Figure 4. The system achieves fast tracking in less than 3 seconds with no overshoot. However, a static error is present, and it does not converge to zero. As a result, the obtained controller falls short of meeting the performance requirements for C_1 validation.

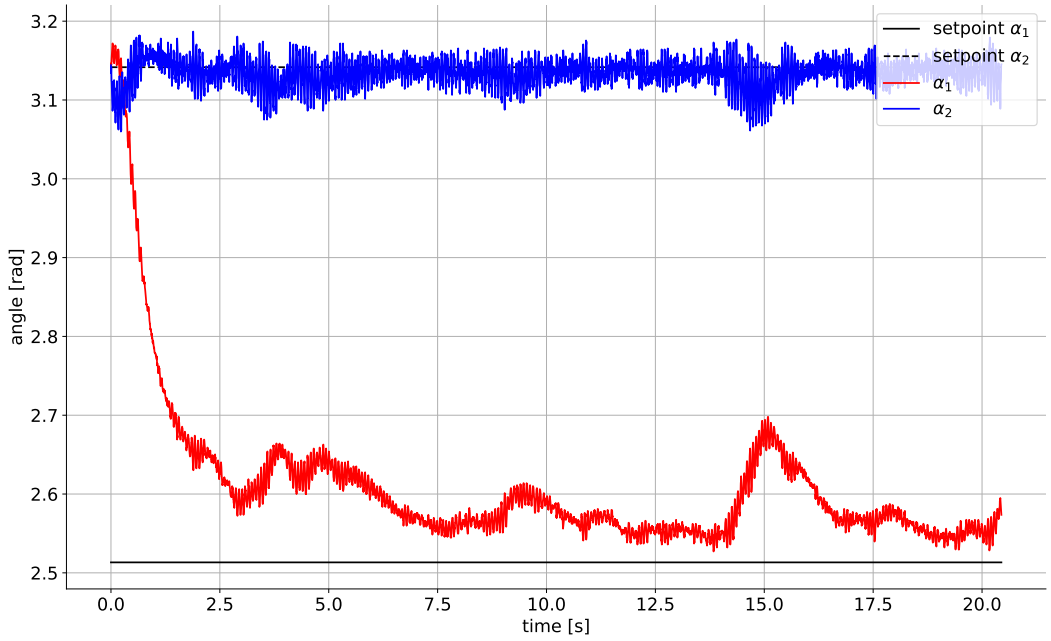


Figure 4: Controller C1 Validation with the simulated Pendubot, first attempt.

3.2 Controller C_1 : Simulator Trial #2

In order to improve the control law performances, we need to change the matrix Q and R. The second attempt was to take the following :

$$Q = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 10 & 0 & 0 \\ 0 & 0 & 0.1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad R = 0.5 \tag{8}$$

The main adjustments made include a slight decrease in $Q_{1,1}$ to reduce the penalty on the angle of the first joint. Additionally, we increased R to make the system less reactive to disturbances, penalizing variations in the input. These modifications were introduced to enhance the controller's performance and stability in the face of external disturbances.

The resulting K matrix is given by:

$$K = \begin{bmatrix} -0.5336 & -17.2353 & -1.5268 & -2.4778 \end{bmatrix}$$

The simulated results are illustrated in Figure 5. The achieved performance now satisfies the required criteria: a response time of less than 3 seconds, no overshoot for the first state, an overshoot of less than 20% for the second state (specifically, only 2%), and the elimination of any static error.

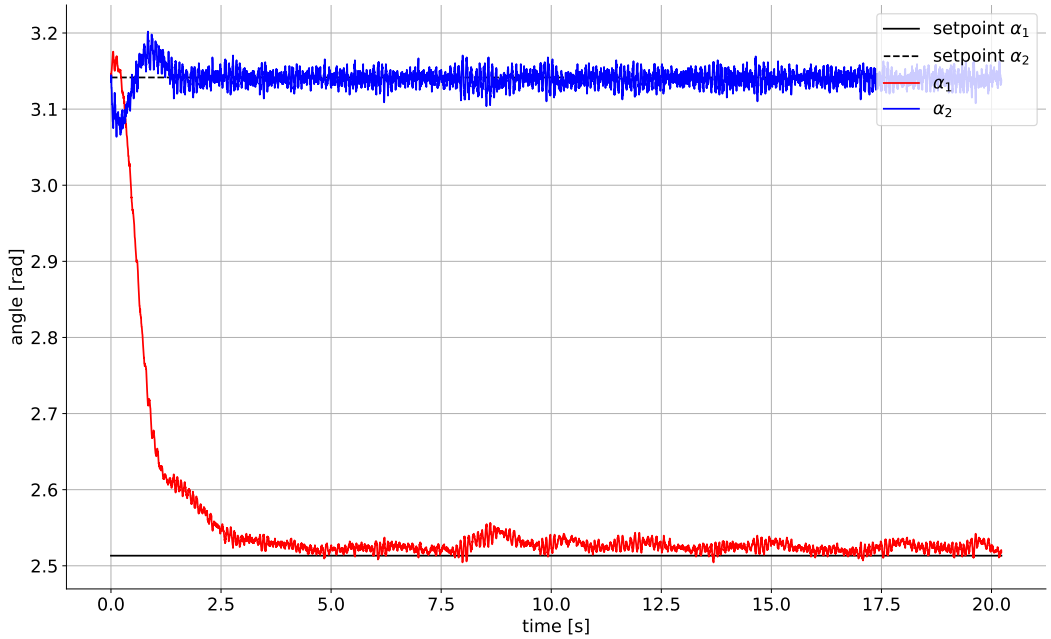


Figure 5: Controller C1 Validation with the simulated Pendubot, final attempt.

3.3 Controller C_1 : Real Plant Validation

The last step for the controller C_1 is to be validated with the real Pendubot. the gains of the LQR are :

$$K = \begin{bmatrix} -0.5336 & -17.2353 & -1.5268 & -2.4778 \end{bmatrix}$$

Figure 6 illustrates the obtained results. Our LQR controller was applied at 5 seconds. The real system with the LQR control law exhibits a minimum phase behavior. This means that initially, the error increases before decreasing. This phenomenon was not captured by the simulator. Furthermore, the response time is even shorter, approximately 1 second compared to 3 seconds in the simulator. There is no overshoot for the first state, and the second state still exhibits a very small overshoot (less than 20%). The static error remains almost null with small oscilations.

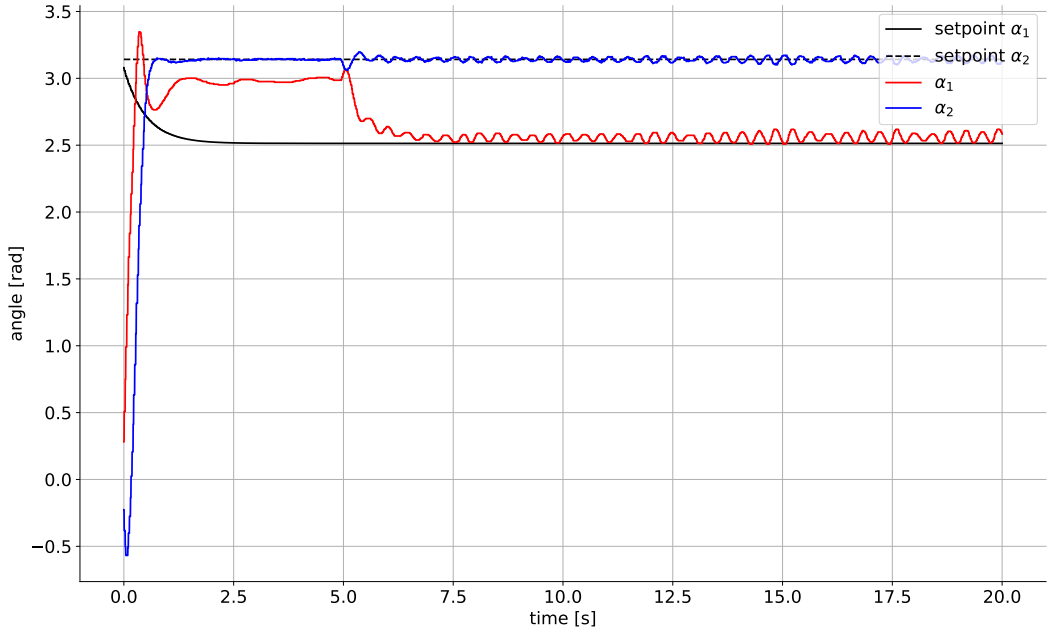


Figure 6: Controller C1 Validation with the real Pendubot

4 Controller C_3

Controller C_3 is split into two different controllers - **Swing-Up** and **Catch**. These controllers serve the purpose of swinging up the Pendubot from $(0, 0, 0, 0)$ to $(\pi, \pi, 0, 0)$. At the end of the chapter is the simulator and real plant validation/discussion.

4.1 Swing-Up Controller

The **Swing-Up** controller is a nonlinear energy-based controller rooted on a Lyapunov function inspired by [2].

$$V(\alpha, \dot{\alpha}) = k_k K + \frac{1}{2}\Delta\alpha^T k_d \Delta\alpha \tag{9}$$

where, $K = \frac{1}{2}\dot{\alpha}^T M(\alpha)\dot{\alpha}$ is the kinetic energy, $k_d = 30$ and $k_k = 0.1$ are fine-tuned constants.

From this Lyapunov function the derivative is such that,

$$\dot{V} = k_k \dot{K} + \Delta\dot{\alpha}^T k_d \Delta\dot{\alpha} \tag{10}$$

Since,

$$\dot{K} = \dot{\alpha}^T M(\alpha)\ddot{\alpha} + \frac{1}{2}\dot{\alpha}^T \dot{M}(\alpha)\dot{\alpha} \tag{11}$$

and,

$$M(\alpha)\ddot{\alpha} = -C(\alpha, \dot{\alpha})\dot{\alpha} - G(\alpha) + \tau \tag{12}$$

from algebraic manipulation it is possible to show that,

$$\dot{V} = k_k \dot{\alpha}^T(\tau - G(\alpha)) + \Delta\alpha^T k_d \dot{\alpha} \tag{13}$$

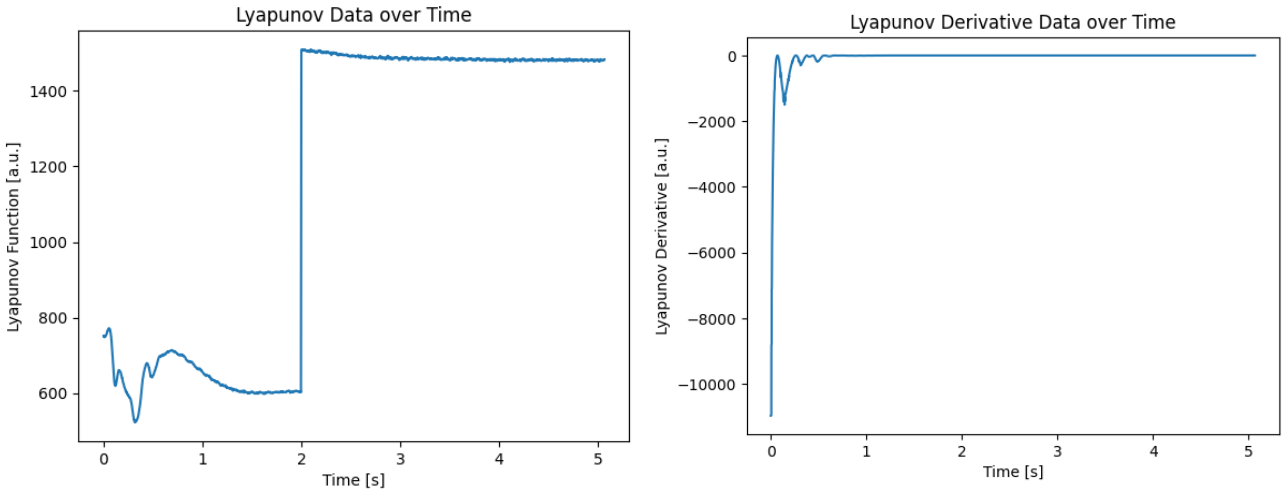
To fulfill the Lyapunov condition for asymptotic stability, $\dot{V} < 0$, it can be proposed that

$$\dot{V} = -||\dot{\alpha}||^2 \iff \dot{V} = -\dot{\alpha}^T\dot{\alpha} \tag{14}$$

hence the control law would be,

$$\tau = -\frac{1}{k_k}(\dot{\alpha}_1 + k_d\Delta\alpha_1) + G_1(\alpha) \tag{15}$$

Below is the evolution of the Lyapunov function as well as its derivative:



(a) Lyapunov Function

(b) Lyapunov Derivative Function

Figure 7: Lyapunov Control Law

As we can see $V > 0$ and $\dot{V} < 0$ which fulfills the Laypunov condition for asymptotic stability. Moreover, at around $t = 0.3s$ is when the switch condition, explained below, happens and the controller goes from **Swing-Up** to **Catch Controller**.

4.2 Catch Controller

When it comes to the **Catch Controller** this controller is split into a Switch Condition and a LQR Controller.

4.2.1 Switch Condition

When it comes to the Switch Condition this is the treshold that if activated allows the general C_3 controller to "switch" between **Swing-Up** control and **Catch** control.

It is set as,

$$|(\alpha_1 - \frac{\pi}{2})| < 0.01 \text{ and } |(\frac{\pi}{2} - (\alpha_1 + \alpha_2))| < 0.2 \tag{16}$$

Once switched, the controller is designed to stay as the **Catch Controller** for the rest of the simulation. This is obtained through the implementation of dynamic thresholds, as seen in F,

through the "Link 1 Threshold" and "Link 2 Threshold" MATLAB functions. This means that after the switch is triggered, the thresholds increase to effectively infinity,

$$|(\alpha_1 - \frac{\pi}{2})| < 100 \text{ and } |(\frac{\pi}{2} - (\alpha_1 + \alpha_2))| < 200 \tag{17}$$

If there were no dynamic thresholds there is the possibility that once the switch condition is verified, due to the chaotic behaviour of the pendulum, a small disturbance around both α_1 and α_2 could lead the controller to go back from the **Catch** state to the **Swing-Up** state. This would be undesirable, therefore the group built the controller with this constraint in mind.

4.2.2 LQR Controller

In order to maintain the pendulum at a UU position, $x_e = (\pi, \pi, 0, 0)$ and $u_e = 0\,\text{Nm}$, the same LQR control law obtained for C_1 is applied.

First, the system needs to be linearized around the new equilibrium x_e, u_e. The resulting A and B matrices are:

$$A = \begin{bmatrix} 0 & 0 & 1.0000 & 0 \\ 0 & 0 & 0 & 1.0000 \\ 92.3805 & -53.0791 & 0 & 0 \\ -175.7041 & 159.2373 & 0 & 0 \end{bmatrix}, \quad B = \begin{bmatrix} 0 \\ 0 \\ 46.4992 \\ -88.4397 \end{bmatrix}$$

By retaining the Q and R values that yield good performances for C_1 (see Equation 8), the resulting gains are:

$$K = \begin{bmatrix} -0.4519 & -17.1142 & -1.6944 & -2.4519 \end{bmatrix}$$

4.3 Controller C3 : Simulator Validation

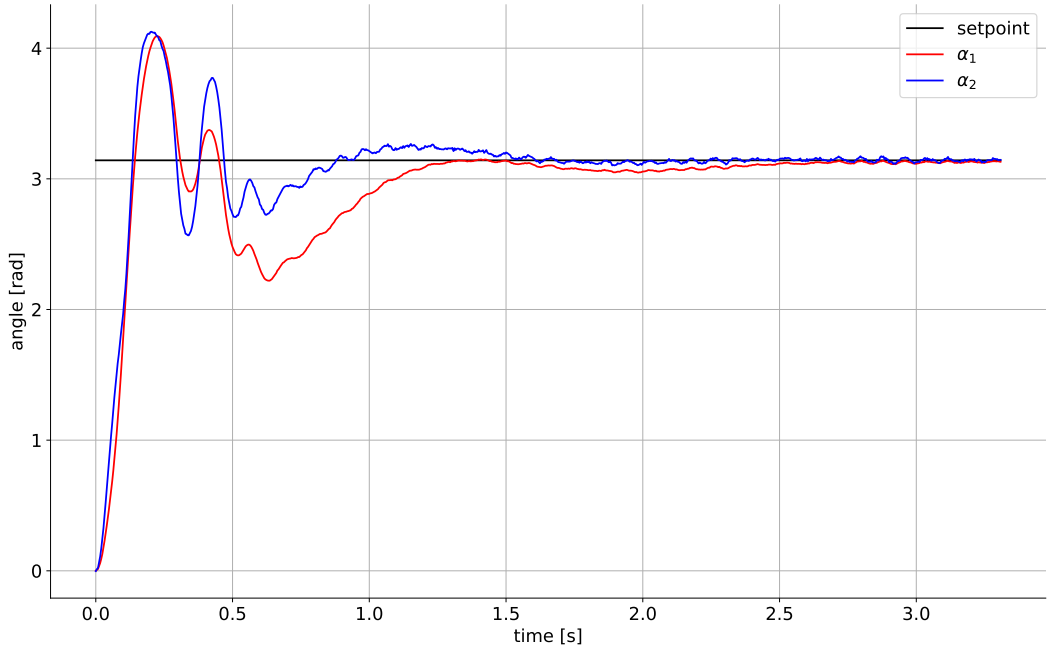


Figure 8: Controller C3 Validation with the simulator

When it comes to the simulated double inverted pendulum through controller C_3, F, the group was sucessful. The provided energy based control law was able to shoot up the pendulum from a DD configuration, $(0, 0, 0, 0)$, and the switch condition was fine-tuned to provide a good time interval for the C_1 controller to maintain the pendulum at UU configuration, $(\pi, \pi, 0, 0)$.

4.4 Controller C_3 : Real Plant Validation

While the group did test for the real system, there were no fruitful results. The real Pendubot system required different k_k and k_d parameters when it comes to the control law in order for it to swing up properly. Nevertheless D and E show the code for the simulated C_3 controller.

5 Conclusion

In conclusion there were major takeaways that improved the knowledge and capabilities of the group members when it comes to designing advanced controllers for such an unorthodox system.

Firstly, the group learned the importance of having a correct linearized model for a nonlinear chaotic system. Without it, it is exponentially harder to proceed with any controller design.

Secondly, the drastic differences between a simulated and real environments. While the group was sucessful when controlling the system in a simulated environment through C_1 and C_3, the real system was a completely different challenge. Had the group more time, the control

laws could have been fine tuned better for controller C_3 and more practical results could have been obtained.

Thirdly, whilst not fully implemented, the group still engaged in the design and some simulations of C_2. However the group felt as though these simulations were incomplete and did not bring anything relevant in terms of data comparison to controller C_1. Nevertheless, studying high-performance versus robust controllers gives a good insight on how performance and energy consumption trade-off works for different controllers. This intuition and knowledge is of high importance for any control engineer to acquire and consolidate.

Fourthly, when it comes to the design of C_1 the group went through several attempts to reach the proper design methods. For this the group used fine-tuning methods of the Q and R matrices that ate away more time than they should have. Whilst good results were obtained, more careful *a priori* theoretical consideration of the system and the design constraints of C_1 could have saved time. So while there is a purpose to fine-tuning by hand LQR parameters, these should not be done blindly, but instead with a clear goal in mind, i.e penalizing more/less certain states that we know have more significance when it comes to controlling the Pendubot.

Fifthly, when it comes to the design of C_3 the group understood the importance of taking into consideration the totality of the dynamics of the pendulum. The pendulum is a two joint system, hence there exist kinetic and potential energy components at play that should not be neglected. Formulating a good Lyapunov function that takes into account these parameters is of the highest importance to generate a clean and good swing up of the pendulum. Once these parameters are well deconstructed the energy-based controller becomes easier to construct. However, the group feels as though the real system behaved poorly because the chosen Lyapunov function was not robust enough to outside disturbances, something barely addressed in the proposed control law. In the future, when designing a real world energy-based controller, not only should the entirity of the dynamics be taken into consideration, but real system disturbances as well.

In the end, we are proud of the results and insights we have achieved in this project, despite not being able to fulfill every aspect as initially envisioned. It's important to recognize that the constraints of time and the complex nature of the material, posed significant challenges. We know that with more time and more engaged team-work our outcomes could have been further enhanced. This project and the one before were two valuable learning experiences gifting us with ways of thinking we had yet to develop when it comes to designing specific advanced controllers.

A Linearization and LQR Control Law Gains

The following MATLAB code provides the linearization of the nonlinear system and computes the LQR, DLQR control gains.

```
1
2  syms a1 a2 da1 da2 t;
3
4  % Define the constants
5  p1 = 0.0148;
6  p2 = 0.0051;
7  p3 = 0.0046;
8  p4 = 0.1003;
9  p5 = 0.0303;
10 grav  = 9.81;
11 k = 3.9621/8;
12
13 % Define matrices M, Vm, G
14 M  = [p1+p2+2*p3*cos(a2-a1), p2+p3*cos(a2-a1); p2+p3*cos(a2-a1),
      p2];
15 Vmda = p3*sin(a2-a1) * [da1^2-da1*da2-da2^2; da1^2];
16 G  = [p4*grav*sin(a1); p5*grav*sin(a2)];
17
18 % Define the state vector and input
19 x_notice = [a1; a2; da1; da2];
20
21 % Define the nonlinear dynamics f(x) and g(x)
22 syms f(a1,a2,da1,da2);
23 f(a1,a2,da1,da2) = [da1;da2;M\(-Vmda-G)];
24
25 syms g(a1,a2,da1,da2);
26 g(a1,a2,da1,da2) = [0;0;M\([k; 0])];
27
28 % Compute the Jacobian matrix A
29 A = jacobian(f, x_notice) + jacobian(g, x_notice)*t;
30
31 % Compute the Jacobian matrix B
32 B = g;
33
34 % Set the equilibrium point AROUND WHICH WE LINEARIZED
35 x_e = [4*pi/5; pi; 0; 0];
36
37 syms te
38 eqn = 0 == f(x_e(1),x_e(2),x_e(3),x_e(4)) + g(x_e(1),x_e(2),x_e
      (3),x_e(4)) * te;
39 t_e = double(solve(eqn, te)); % result : t_e = 1.1678
```

```
disp('te');
disp(t_e);


% Substitute the equilibrium point into the matrices A, B, and C
A_e = double(subs(A, {a1, a2, da1, da2, t}, {x_e(1), x_e(2), x_e
    (3), x_e(4), t_e}));
B_e = double(subs(B, {a1, a2, da1, da2, t}, {x_e(1), x_e(2), x_e
    (3), x_e(4), t_e}));

% Construct the linear state-space representation
sys = ss(A_e, B_e, eye(4), 0);
% Discretize
% Specify the time step for discretization
dt = 0.004;

% Discretize the system using c2d
sys_d = c2d(sys, dt, 'zoh');

% Display the sys matrices
disp('Continuous A_e and B_e:');
disp(A_e);
disp(B_e);
disp('Discretized A_d and B_d:');
disp(sys_d.A);
disp(sys_d.B);

% Control Law
%Q = [2, 0, 0, 0; 0, 10, 0, 0; 0, 0, 0.1, 0; 0, 0, 0, 1];
%R = 0.3;
Q = [1, 0, 0, 0; 0, 10, 0, 0; 0, 0, 0.1, 0; 0, 0, 0, 1];
R = 0.5;
K = lqr(sys.A, sys.B, Q, R);

% Displaying
disp('LQR, K :');
disp(K);

K = dlqr(sys_d.A, sys_d.B, Q, R);

% Displaying
disp('DLQR, K :');
disp(K);
```

B Controller C_1

Below is the code for controller C_1,

```
1
2  function u=pendubot_reg(alpha1r,x)
3  %function u=pendubot_reg(alpha21,x)
4  %
5  %INPUTS:
6  % alpha1r : reference value for the first link angle [rad]
7  % x       : measured state vector x =   [
8  %
                     x1 = alpha_1 - pi
9  %
                     x2 = alpha_2 - pi
10 %
                     x3 = dalpha_1/dt
11 %
                     x4 = dalpha_2/dt
12 %                                    ]
13 %
14 %OUTPUT:
15 % u       : control signal (motor torque) to be applied [Nm]
16
17 % Set the equilibrium point
18 x_e = [4*pi/5; pi; 0; 0];
19 t_e = 1.1678;
20
21 % Control law
22 K = [-0.5336   -17.2353    -1.5268    -2.4778];
23 corr_x_measured = [wrapToPi(x(1))+pi-x_e(1); wrapToPi(x(2))+pi-
      x_e(2); x(3)-0; x(4)-0];
24 v = -K*corr_x_measured;
25 u= v+t_e;
```

C C_1 Simulated Block Diagram

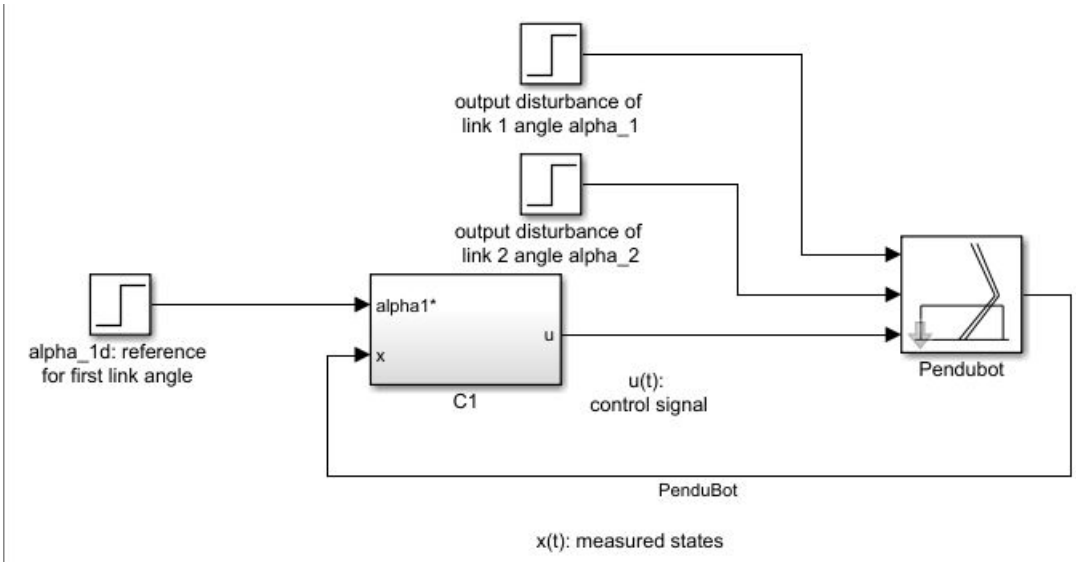


Figure 9: C_1 Controller's Block Diagram

D Swing-Up C_3 Controller

Below is the code for the **Swing-Up** feature of controller C_3,

```
1
2  function [u, V, dV] = pendubot_C3(alpha1r,x)
3
4  a1 = x(1);
5  a2 = x(2);
6  da1 = x(3);
7  da2 = x(4);
8
9  % Define Constants
10 p1 = 0.0148;
11 p2 = 0.0051;
12 p3 = 0.0046;
13 p4 = 0.1003;
14 p5 = 0.0303;
15 grav  = 9.81;
16 torque = 3.9621;
17 k = 3.9621/8;
18 % tau = [torque ; 0];
19
20 % Dynamic Matrices
21
```

```
M  = [p1+p2+p3*cos(a2), p2+p3*cos(a2); p2+p3*cos(a2), p2];
C = p3*sin(a2) * [-da2 , -da1 - da2 ; -da1 , 0];
G  = [p4*grav*cos(a1) + p5 * grav * cos(a1 + a2); p5*grav*cos(a2
   + a1)];
F = [0.00545 * da1 + 0.0023 * tanh(30 * da1) ; 0.00047 * da2 +
   0.0025 * tanh(30 * da2)];

% Energy Data

U = p4 * sin(a1) + p5 * sin(a1 + a2);

K = 0.5 * [da1 , da2] * M * [da1 ; da2];
% dK = [da1 , da2] * (tau - C * [da1 ; da2] - G - F) + 0.5 * [da1
    , da2] * dM * [da1 ; da2];

E = K + U;

E_TOP = (p4 + p5) * grav;

E_est = E - E_TOP;

% Lyapunov Data

kk = 0.1;  %6.359;
kd = 30; %8.71;
a1_est = a1 - alpha1r;
a2_est = a1 + a2 - alpha1r;

V = kk * K + 0.5 * [a1_est , a2_est] * kd * [a1_est ; a2_est];
dV = -da1^2;
% Control Law
G_1 = p4 * grav * sin(a1);
tau = -1/kk * (da1 + kd * a1_est) + G_1 ;
u = tau;
end
```

E Catch C_3 Controller

Below is the code for the **Catch LQR** (similar to C_1) feature of the controller C_3,

```
function u=pendubot_reg(alpha1r,x)
%function u=pendubot_reg(alpha21,x)
%
%INPUTS:
% alpha1r : reference value for the first link angle [rad]
```

```
% x        : measured state vector x =    [
%
                    x1 = alpha_1 - pi
%
                    x2 = alpha_2 - pi
%
                    x3 = dalpha_1/dt
%
                    x4 = dalpha_2/dt
%                                          ]
%
%OUTPUT:
% u        : control signal (motor torque) to be applied [Nm]

% Define the constants
p4 = 0.1003;
grav  = 9.81;

% Set the equilibrium point
x_e = [pi; pi; 0; 0]; %[4*pi/5; pi; 0; 0];
t_e = 0; %sin(x_e(1))*p4*grav;


K = [-0.4519  -17.1142   -1.6944   -2.4519];
corr_x_measured = [wrapToPi(x(1))+pi-x_e(1); wrapToPi(x(2))+pi-
   x_e(2); x(3)-0; x(4)-0];
v = -K*corr_x_measured;
u= v+t_e;
```

F C_3 Simulated Block Diagram

The logic processor after the *wrapToPi* function serves as the dynamic Switch Condition, as described in 4.2.1. The necessity for a delay block between the *AND* gate and the "Link 1 Threshold" and "Link 2 Threshold" MATLAB functions serves to allow the correct implementation of the dynamic thresholds in a Simulink environment. Without it, the simulation generated an "Algebraic Loop" error.

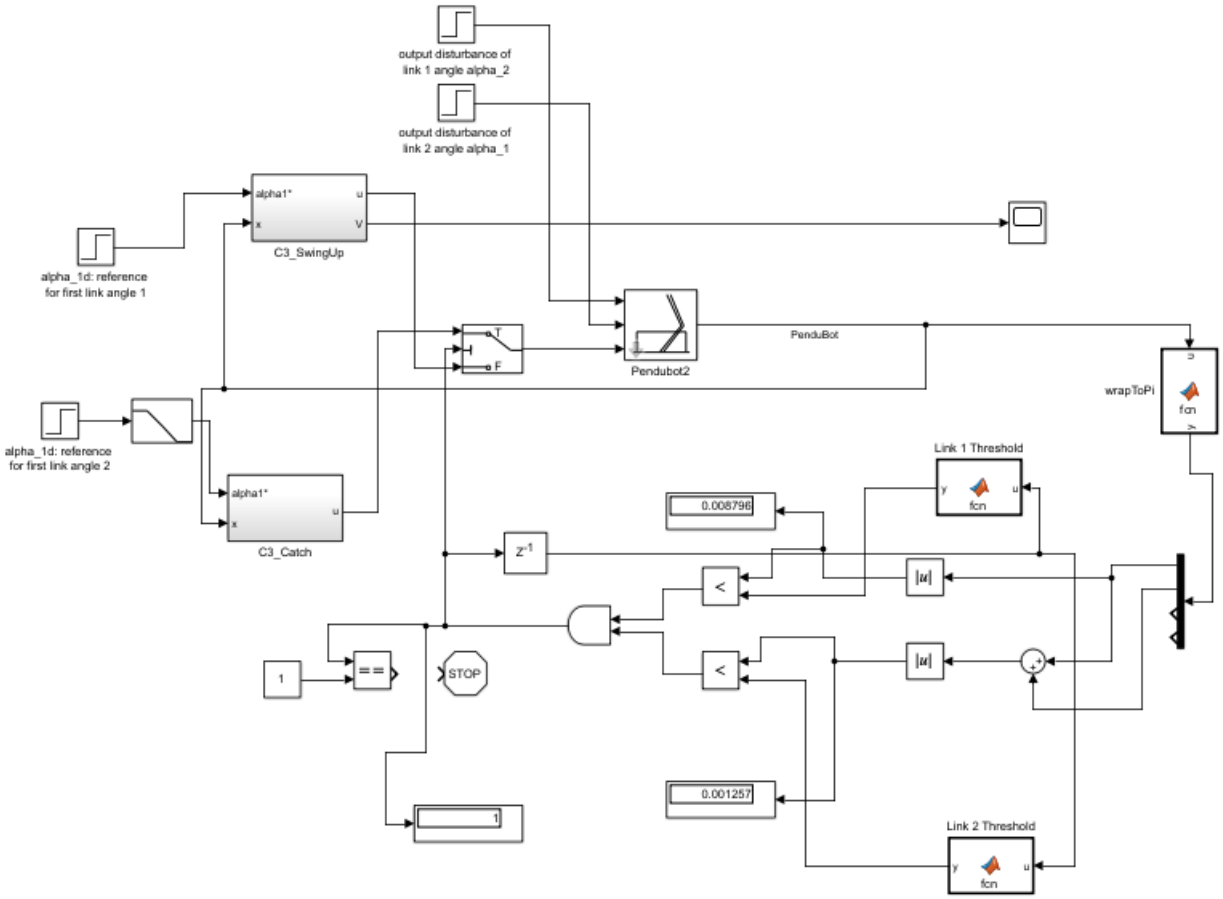


Figure 10: C_3 Controller's Block Diagram

References

[1] MathWorks. "Control System Toolbox - ss.lqi." (2023), [Online]. Available: https://www.mathworks.com/help/control/ref/ss.lqi.html.

[2] S. Jesus Patricio Ordaz Oliver Omar Arturo Domınguez Ramirez and O. Lequin, "Control based on swing up and balancing scheme for an underactuated system, with gravity and friction compensator," *Fourth Congress of Electronics, Robotics and Automotive Mechanics*, 2007.