UNIVERSITA' DEGLI STUDI DI SALERNO

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE ED ELETTRICA E MATEMATICA APPLICATA

Project Work - Embedded Digital Controllers

Nome e Cognome	Matricola	Email
Beniamino Squitieri	0622702021	`b.squitieri@studenti.unisa.it`
Francesca Venditti	0622701969	`f.venditti1@studenti.unisa.it`

ANNO ACCADEMICO 2023/2024

Contents

1 Motor Description

The Pololu 37D Metal Gearmotors are powerful brushed DC motors paired with 37 mm diameter gearboxes and are available with or without a 64 CPR quadrature encoder integrated on the motor shaft. These motors are designed to operate at 12 V, although in general, this type of motor can run at voltages higher and lower than the nominal voltage, and they can start rotating with voltages as low as 1 V.

The geared motors are available in a variety of different gear ratios, offering many different combinations of speed and torque. The figure below is a representation of the motor just described:

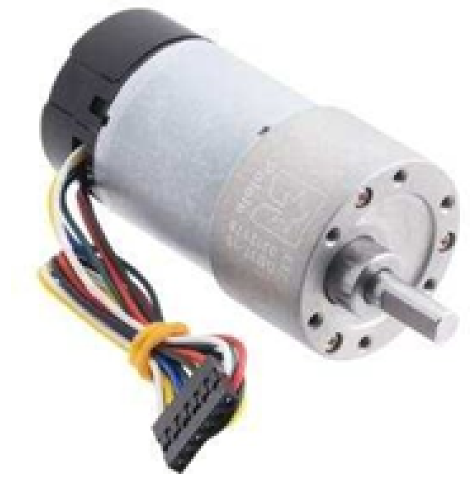


Figure 1: Rappresentation of DC motor 37D Pololu

1.1 Adding the Encoder

he versions with an encoder have additional electronics mounted on the back of the motor. Two Hall effect sensors are used to detect the rotation of a magnetic disc on a rear protrusion of the motor shaft. The encoder electronics and the magnetic disc are enclosed by a removable plastic cap.

The quadrature encoder provides a resolution of 64 counts per revolution (CPR) of the motor shaft when counting both edges of both channels. To calculate the counts per revolution of the gearbox output, multiply the gear ratio by 64.

The motor/encoder has six color-coded wires, 20 cm (8") long, terminated with a 1×6 female connector with a 2.54 mm (0.1") pitch. This connector works with standard 0.1" breakaway male headers and with our premium male jumper wires and pre-crimped wires.

Lead Color	**Function**
Red	Motor power
Black	Motor power
Green	Encoder ground
Blue	Encoder Vcc (3.5 V to 20V)
Yellow	Encoder A output
White	Encoder B output

Hall sensors require an input voltage, Vcc, between 3.5 V and 20 V and draw a maximum of 10 mA. The outputs A and B are square waves from 0 V to Vcc, approximately 90° out of phase. The motor speed can be determined by the frequency of the waves, and the direction of rotation can be determined by the order of the transitions. The following oscilloscope capture shows the encoder outputs A and B (yellow and white) using a motor voltage of 12 V and a Hall sensor Vcc of 5 V.

2 State Feedback

2.1 State Feedback Controller for Regulator System

Following figure shows the architecture of the full state feedback controller for regulator system (a system with zero reference signal or zero desired output). In the full state feedback controller analysis and design, the plant is represented by the following state-space model:

$$\begin{aligned} \dot{x} &= Ax(t) + Bu(t) \\ y &= Cx(t) + Du(t) \end{aligned}$$

where x(t), y(t) and u(t) are the n-dimensional state vector, output vector and input vector respectively. In addition, A, B, C and D are the plant parameters related to the physical parameters.

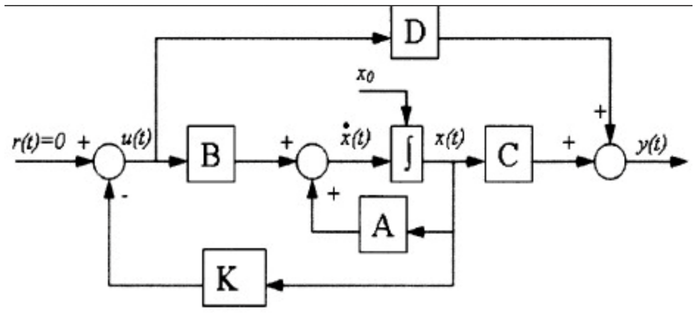


Figure 2: Scheme

Refereeing to the full state feedback control system shown in Figure 2, the closed loop system utilizes state feedback control laws which has the form

$$u = -Kx(t)$$

where is state feedback gain matrix. By substituting previous equation in the previous system, the closed-loop system is mathematically described as follows:

$$\dot{x} = (A - BK)x(t)$$
$$y = Cx(t)$$

The state feedback gain matrix K of dimension can be designed easily by using pole placement or approach.

2.2 State Feedback Controller for Tracking System

State feedback controller for regulator system can not be used directly for tracking system by simply letting r(t) different from 0 since the steady state error may exist. A modification has to be done to ensure that the closed loop output equals or closes to the desired output r(t). One method to allow full state feedback controller can be used for tracking system is by adding integral control as shown in the following figure:

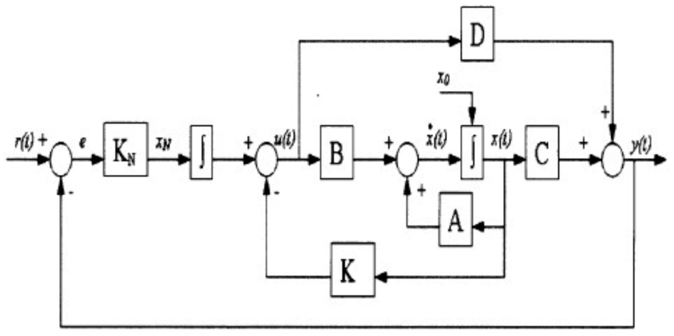


Figure 3: Scheme

In order to introduce integral control, an augment state is added to the closed-loop system. The augment state is defined as follows

$$z(t) = \int (y(t) - r)dt$$

Based on the augment state $z(t)$, the following augment state equation is obtained:

$$\dot{z}(t) = y(t) - r = Cx(t) - r$$

Then the original state and the augment state equations are combined as follows:

$$\begin{bmatrix} \dot{x}(t) \\ \dot{z}(t) \end{bmatrix} = \begin{bmatrix} A & 0 \\ C & 0 \end{bmatrix} \begin{bmatrix} x \\ z \end{bmatrix} + \begin{bmatrix} B \\ 0 \end{bmatrix} u + \begin{bmatrix} 0 \\ -1 \end{bmatrix} r$$

Finally, by considering the new control law for tracking system

$$u(t) = -Kx(t) + K_i \int (y(t) - r)dt$$

the closed-loop state equation is

$$\begin{bmatrix} \dot{x} \\ \dot{z} \end{bmatrix} = \begin{bmatrix} A - BK & K_I \\ C & 0 \end{bmatrix} \begin{bmatrix} x \\ z \end{bmatrix} + \begin{bmatrix} BK_R \\ -1 \end{bmatrix} r$$

The controller parameters K and Ki can be obtained by using similar pole placement method as discussed previously for regulator system.

In our case with the DC motor we have the following matrices: A = $\begin{bmatrix} 0 & 1 \\ 0 & -\frac{1}{\tau} \end{bmatrix}$ B = $\begin{bmatrix} 0 \\ \frac{K_m}{\tau} \end{bmatrix}$ C = $\begin{bmatrix} 1 & 0 \end{bmatrix}$ C = $\begin{bmatrix} 0 & 1 \end{bmatrix}$ D = 0

3 Choose of the Transfer Function

The selection of the transfer function was done using the tangent method. The tangent method is a graphical approach used to determine an approximate transfer function of a dynamic system. This method is particularly useful when the system's step response is available and a simplified transfer function describing that response is desired.

To apply the tangent method, the first step is to obtain the system's step response as follows:

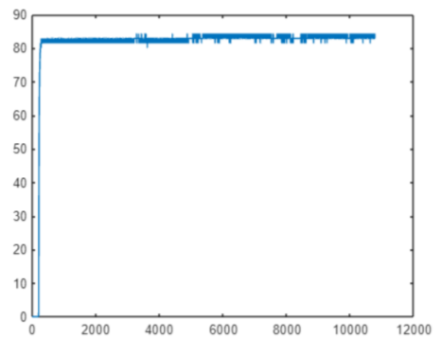


Figure 4: Open Loop Step response

The next step is to identify the inflection point on the step response curve. The inflection point is where the curve changes concavity; this point is crucial because it represents the point of maximum slope on the curve.

Once the inflection point is identified, a tangent is drawn to the curve at that point. The tangent represents the straight line that touches the curve at the inflection point and has the same slope as the curve at that point. The tangent is then extended until it intersects the time axis.

Subsequently, the time is measured from the point where the tangent intersects the time axis to the point where the tangent reaches the final value of the step response. This time is approximately the time constant of the system. For further implementation details, the Matlab code provided shows the solution for both motors.

3.1 Simulink fdt

To solve the procedure described in the previous paragraph, an open-loop representation of the system was used to obtain the system's step response. The following diagram represents the aforementioned setup:

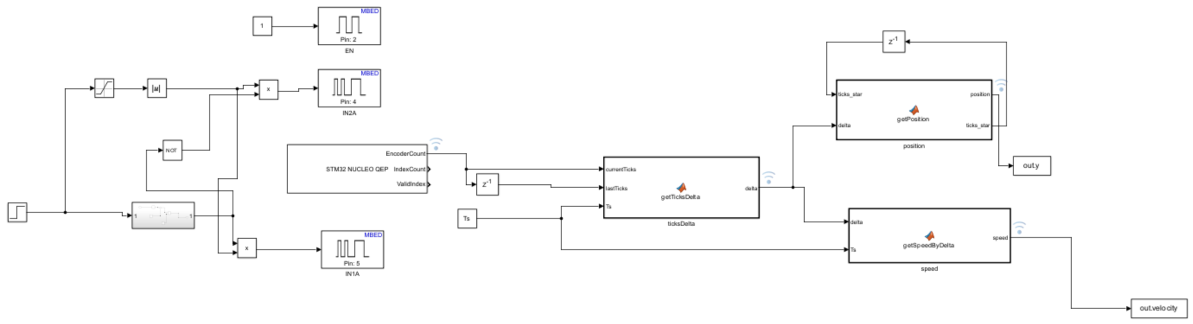


Figure 5: Open Loop System

4 Controller Design

4.1 LQR

LQR allows obtaining the gains of a state-feedback controller through an optimization procedure of a quadratic cost function. Linear is the constraint of our optimization problem, quadratic is the performance index (quadratic both in the state and in the control input) that we aim to minimize.
$min_{u(\cdot)}\tilde{J}$
$s.t\dot{x} = Ax + Bu$, in which $\tilde{J} = \int_0^{\infty} x^T Q_x x + u^T Q_u u dt$ and represents the cumulative performance index. In this technique, the key step that the designer must take is the choice of the matrices Q_x and Q_u, which must be positive definite for convergence of the Riccati equation. These matrices are respectively related to the control task and the control effort and are fundamental to achieve a good trade-off between system performance and the control effort required to achieve such performance. What is relevant is not the numerical value of Q_x and Q_u, but their relative ratio. Design steps:

1. Controllability: The controllability matrix is a matrix that represents the ability to drive the system from an arbitrary initial state to a desired state using the control input. It is defined as: $W_R = [B; AB; A^2B; A^3B; ...; A^nB]$. Therefore, the system is controllable if and only if the controllability matrix is full rank, that is, if its dimension is equal to the order of the system.

2. Choose Q_x, Q_u
3. Find : $k = lqr(A, B, Q_x, Q_u)$

We choose the two diagonal matrices, and this allows not only to simplify the calculations but also to give different importance to the various states.

4.2 Velocity LQR

To find the values of Q_x and Q_u, the first step was to reduce the dimensionality of our system. Our initial system consists of two states: position (denoted as x_1) and velocity (denoted as x_2), to which the presence of integral action has been added. Integral action is a component of a controller aimed at eliminating steady-state error, i.e., the residual error that remains when the system reaches a steady state. The formulation of our control input is as follows: $u = -K \cdot x + K_i \cdot \int y\, dt + z + K_r \cdot r$. Due to the presence of integral action, the term $K_r \cdot r$ is null because, by correctly designing the integral action, we can conclude that $y = r$, thereby concluding that we do not need the indicated term to adjust the error as it has already been included in the integral action.

After this analysis, an analysis of the system's dimensionality was conducted. Intuitively, we can conclude that by adding integral action to the system, it involved adding a new state within it. This addition led to an increase in the dimensionality of the system matrix, making the system not fully controllable. This problem prevented us from finding the values of Q_x and Q_u, as suggested by the output of the "lqi" command in Matlab, which failed to solve the Riccati equation.

For this reason, it was chosen to reduce the dimensionality of the matrix, excluding the state related to the position of our system, thus making the system dependent solely on velocity (x_2) and integral action. Additionally, in order to facilitate the resolution of the Riccati equation by the Matlab command "lqr", it was chosen to represent the system in its controllability canonical form. This was calculated using the "tf2ss" command, which, given the transfer function of the system, directly returns the matrices A, B, C, D in canonical form. Alternatively, the classical procedure could have been used, in which the transformation matrix T is calculated, and then the matrices A, B, C, D of the original system are multiplied by T. However, for ease of implementation, the former methodology was used.

```
clear all clc
pwm_freq=2000
Ts=0.005
Beniamino's motor
tau=1.01002;
k = 6.91219;
Francesca's motor
% tau=1.034;
% k = 6.9;
```

```

A=[-1/tau 0; 1 0]
B=[1; 0]
C=[1 0]
D=0
Wr = [B A*B]

r=rank(Wr)

Qx = diag([0.1, 70]);

Qu = 0.01

sys = ss(A, B, C, D);

[K_lqi,S,P] = lqr(A,B, Qx, Qu);
K_lqi
K = K_lqi(:, 1)
Ki = -K_lqi(:, 2)
```

The resolution of this script allowed us to obtain the values of K and K_i.

4.2.1 Simulink Velocity

The following Simulink diagram aims to highlight the behavior of the system just described. This diagram computes our control input u as described in Section 4.2. This result is obtained by summing Ki and K, previously multiplied by z and $x2$ respectively. In this case, we indeed emphasize the absence of the position state $x1$ for the reasons described earlier. This diagram, as we will see later, will also be used for Pole Placement.

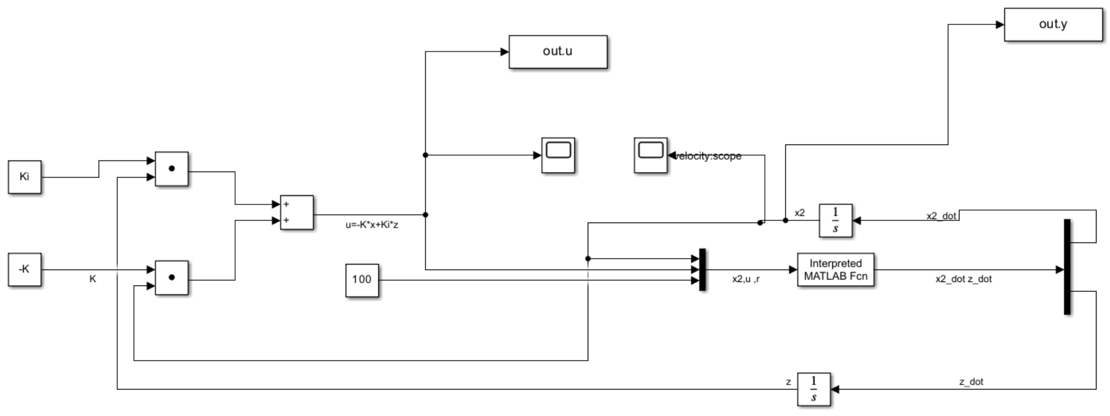


Figure 6: Simulink Velocity

4.2.2 Results Velocity

Regarding the results presented below, we specify that we show the results of a single motor, as the difference in performance between the two is almost negligible. For a better visualization of the results, we recommend referring to the Matlab scripts.

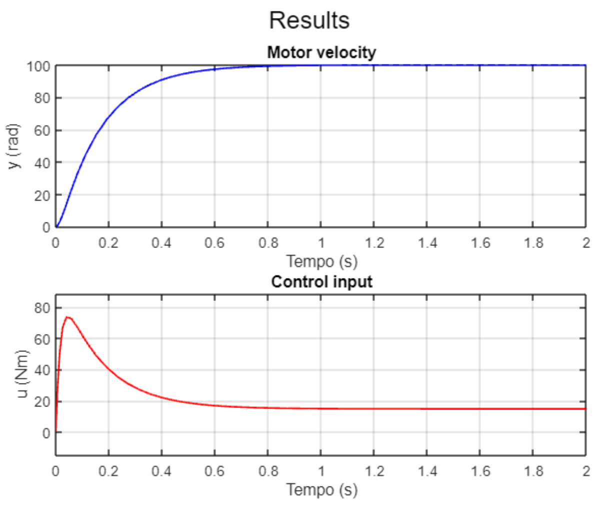


Figure 7: Results Simulation Velocity LQR

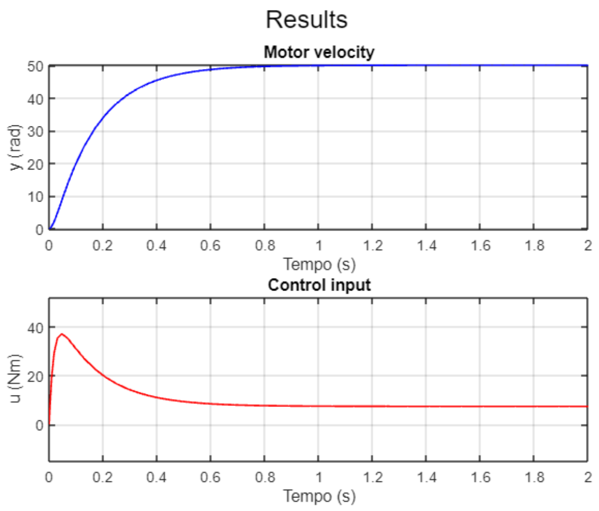


Figure 8: Results Simulation Velocity LQR

The preceding images show two simulations with different reference values. In both cases, we can appreciate how both enjoy an acceptable settling time (approximately 0.6) and low control effort. Therefore, the results obtained have been considered satisfactory.

4.3 Position LQR

The following script calculates the LQR weight matrices for position. As we can observe, the position state (x_1) has a higher weight compared to the velocity state (x_2).

```
pwm_freq=2000
Ts=0.005
%Squitieri's systems:
A=[0 1 ; 0 -1/1.01002]
B=[0; 6.913/1.01002]
C=[1 0]
D = 0;

% Venditti's systems
A=[0 1 ; 0 -1/1.01502]
B=[0; 6.9949/1.01502]
C=[1 0]
D = 0;

Wr = [B A*B]
r=rank(Wr)
%Squitieri's matrix
Qx = diag([30,0.1,2000]);
Qu = 0.1
%Venditti's matrices
Qx = diag([15,0.1,1000]);
Qu = 0.1

sys = ss(A, B, C, D);

[K_lqi,S,P] = lqi(sys, Qx, Qu);

K = K_lqi(:, 1:2) % State gain
Ki = K_lqi(:, 3)  % Integral gain
```

This script return the vector K and the value K_i

4.3.1 Simulink Position

The following plot calculates the control input u of the system as the summation seen previously. In this case, the constant K will be a vector of two elements, which is due to the dependence on the two states, namely $x1$ and $x2$ (position and velocity respectively).

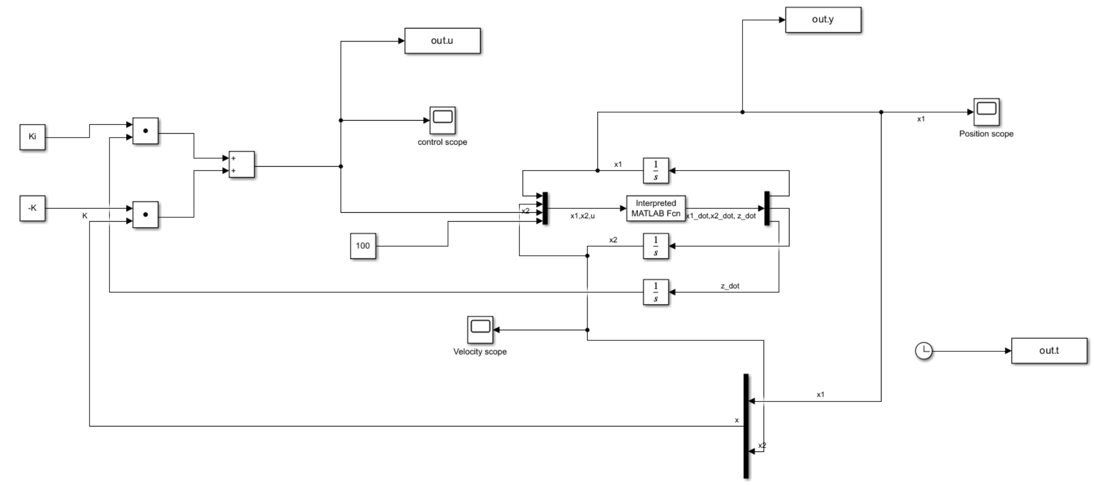


Figure 9: Position Simulink

4.3.2 Results Position

To validate the performance achieved, real tests were conducted with different reference values, evaluating the settling time of the system and its ability to reach a steady state. The following results were obtained:

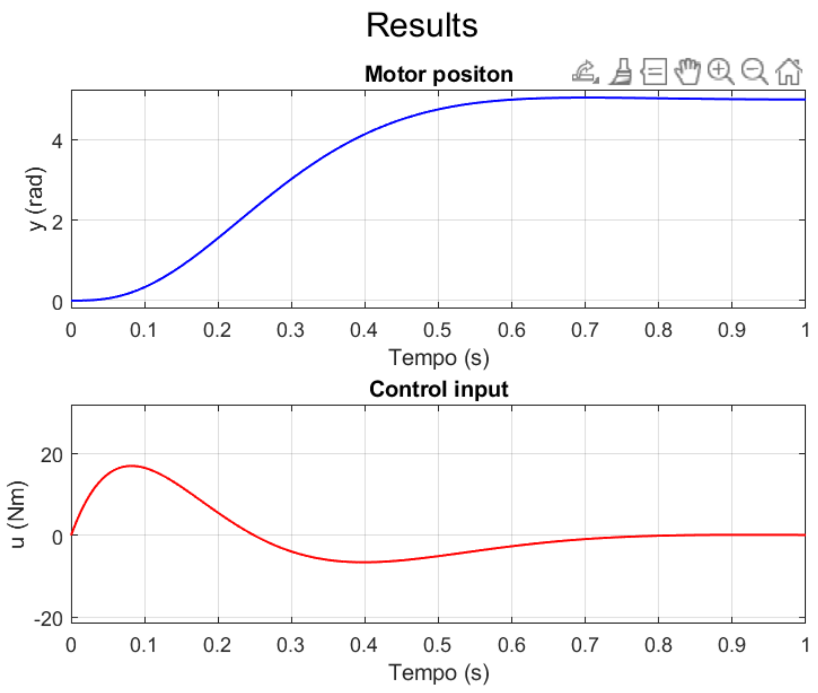


Figure 10: Results Position LQR

4.4 Pole Placement

Pole placement is a method used to calculate the gain matrix used to assign the closed-loop poles to specific positions, thereby ensuring the stability of the system. The positions of the closed-loop poles directly impact the characteristics of the time responses, such as rise time, settling time, and transient oscillations. For a given vector of desired closed-loop pole positions, the *place* command calculates a gain matrix such that the state feedback places the poles in the correct positions.

4.4.1 Velocity Pole Placement

In the case of velocity, please refer to the reasoning presented in the paragraph 4.2.1

```
clear all clc
pwm_freq=2000
Ts=0.005
s = tf ('s')
% Squitieri's motor
G=6.7917/(s*(1+s*1.015))
[A,B,C,D]=tf2ss([0 0 6.91219],[1.01002 1 0])

tau=1.01002;
k = 6.91219;
% Venditti's motor
G=6.9949/(s*(1+s*1.01502))
[A,B,C,D]=tf2ss([0 0 6.9949],[1.01502 1 0])R = [B, A*B
   ]
sys= ss(A,B,C,D)
pole=pole(sys)
% Squitieri's poles
p_desired = [-6 2.4];

%Venditti's poles
p_desired = [-6.5 2.5];

K_place = place(A, B, p_desired);
K = K_place(:, 1)
Ki = K_place(:, 2)

tau=1.01502;
k = 6.9949;
```

4.4.2 Results Velocity

To validate the obtained performance, real tests were carried out with different reference values, where the settling time of the system and its ability to reach a steady state were evaluated, obtaining the following results:

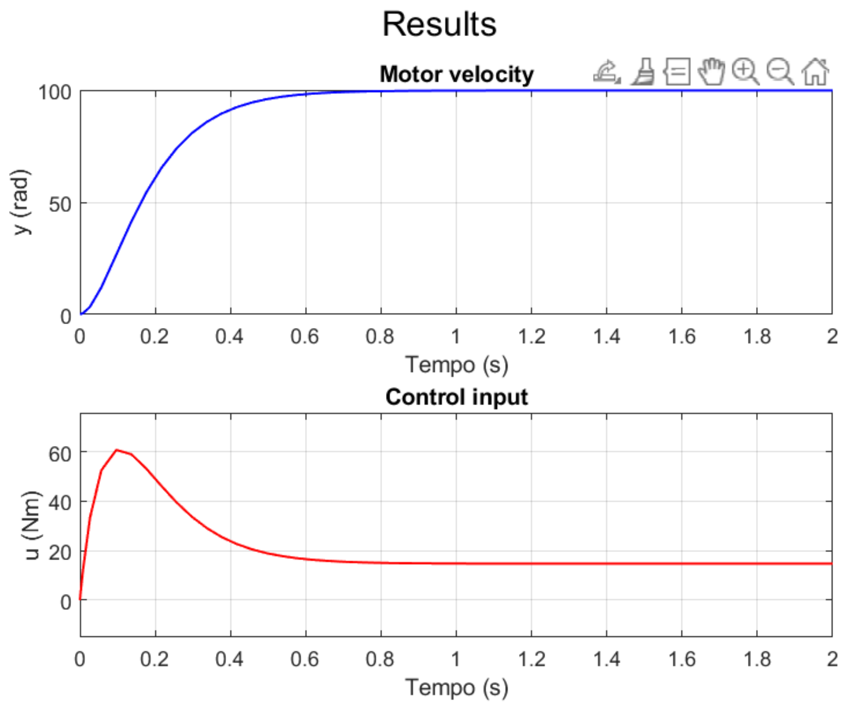


Figure 11: Results Simulation Velocity LQR

Similarly, in this case, the results of simulations conducted on a single motor have also been depicted.

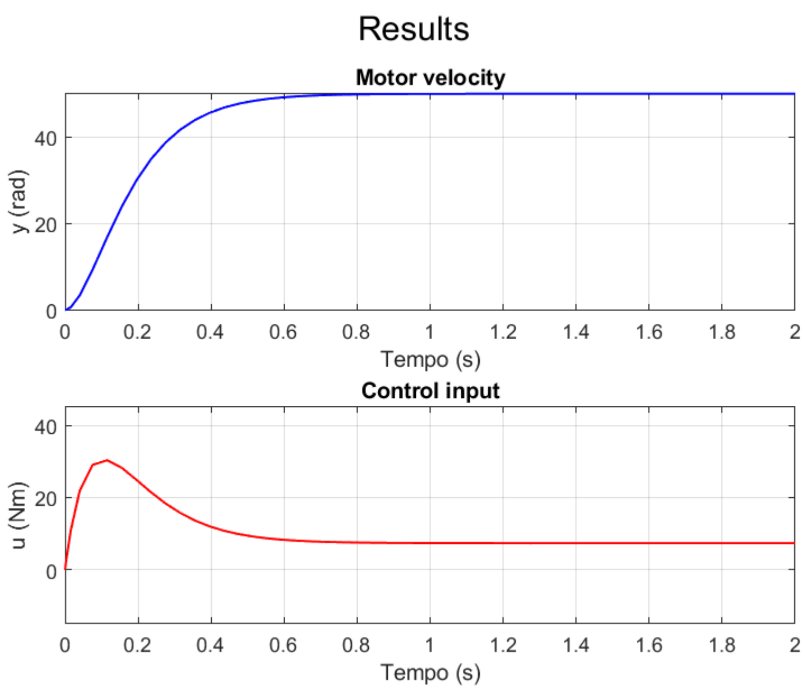


Figure 12: Results Simulation Velocity LQR

4.4.3 Position Pole Placement

For the simulink see 4.3.1

```
clear all clc
pwm_freq=2000
Ts=0.005
s = tf ('s')
%Squitieri's system
A=[0 1 ; 0 -1/1.01002]
B=[0; 6.913/1.01002]
C=[1 0]
D = 0;
%Venditti's systems
A=[0 1 ; 0 -1/1.01502]
B=[0; 6.9949/1.01502]
C=[1 0]
D = 0;
Wr=[B A*B]
eig(A)
% Squitieri's poles
```

```
p = [-10 -30 -15 ]
% Venditti's poles
p = [-15 -20 -10]
% Augmented system with integral action
Ai = [A, zeros(2, 1); C, 0];
Bi = [B; 0];
Ci = [C 0];
% Compute the state feedback gain matrix K for the
    augmented system
[K_place] = place(Ai, Bi, p);
Ri=[Bi Ai*Bi];
% Extracting the state feedback gain for the original
    system
K = K_place(:, 1:2)
Ki = -K_place(:,3)
desired_autovalues=eig(Ai-Bi*K_place)
```

4.4.4 Results Position

To validate the obtained performance, real tests were conducted with different reference values where the settling time of the system and its ability to reach a steady state were evaluated, thus obtaining the following results:

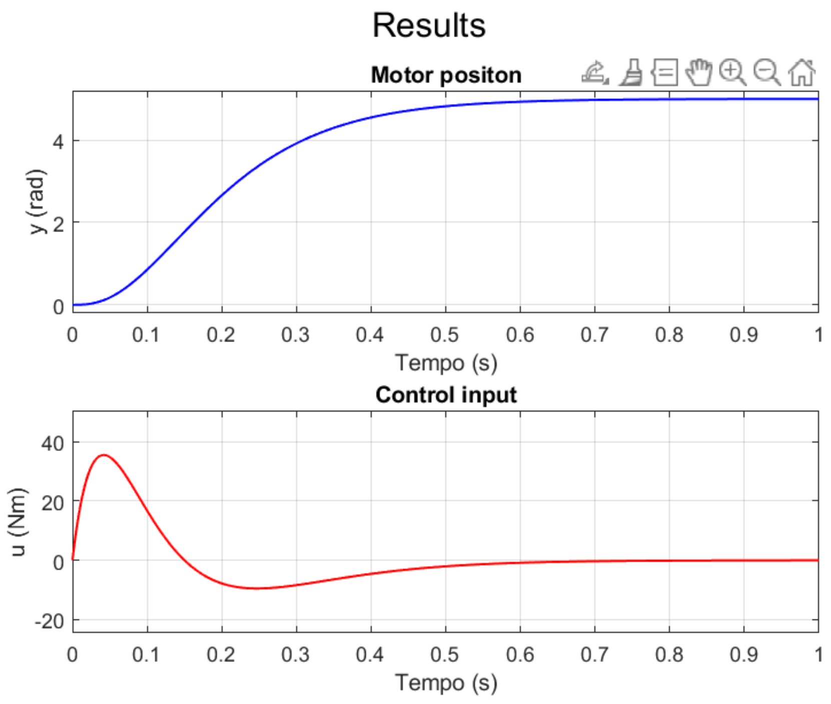


Figure 13: Results Simulation Position Pole Placement

5 Real Controllers

The next step was to connect the proposed controllers with the blocks representing the DC motor seen during the course, obtaining a closed-loop system. This allowed us to carry out real-time simulations. The Simulink diagrams presented below are valid for both the Pole Placement and LQR controllers, but before we should analyse two problems that characterized digital systems, wind-up and bumpless transfer.

5.1 Wind-up

Control system with the wide range of the operations may result in the condition that the control signal reaches the actuator limits. In this case, the feedback is broken and the system runs as an open loop system since the actuator will remain at its limit independently of the plant output. Since the controller contains integrator action, the error will be integrated. This means the integral term become very large or, it is "windup". Then the error must have opposite sign for the long time before things returned to normal. In order to reduce the effect of the integrator windup in the integral state feedback controller, the compensator with tracking anti-windup system is used instead of pure integral state feedback compensator. The structure of the tracking anti-windup controller is illustrated in the following figure:

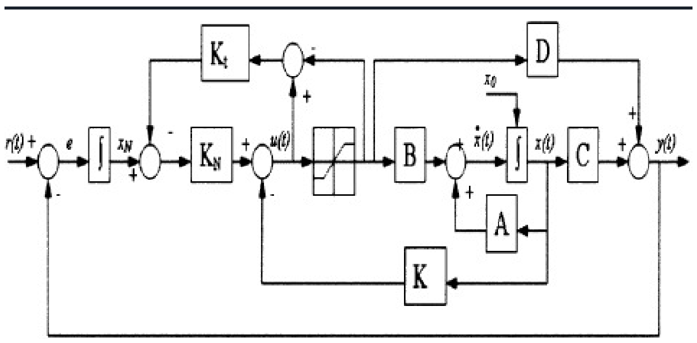


Figure 14: Windup Scheme

where K is called tracking gain . As long as the actuator output is equal to the controller output, anti-windup scheme will not be activated and the controller is in normal operation (control mode). Once controller output u(t) exceeds the actuator limits, a feedback signal will be generated from the difference of the saturated and the unsaturated signals. The anti-windup scheme will be activated and prevent the controller output from wandering away. This signal is used to reduce the integrator input. This methodology is called "integral subtraction". Furthermore in order to avoid wind up problem there are also other strategies. Firstly we have: "setpoint variation" and "fixed limits on integral term". The first one have not been considered because the introduction of limiters on the setpoint variations in order to not allow the controller to reach the actuator limits leads to conservative bounds and poor performances and furthermore it doesn't avoid wind up caused by disturbances. The second one has not been considered because the introduction of a maximum and minimum value for integral summation leads to better performance compared to the ones of the previosu methodology but it is also similar to the "stop summation" methodoligy which has better performance. An other methodology is the "integral subtraction". The idea behind this method is that the integral value is decreased by an amount proportional to the difference between the calculated value of the manipulated variable and the maximum value allowable. The rate of decrease is dependent on the choice of the parameter K; if it is not properly chosen then a continual saturation/desaturation oscillation can occur. The benefit of this method is that the system comes out of saturation as quickly as possible; there is, however, no attempt to match the integral term to the requirements of the plant and the value of K must be chosen by experience rather than by reference to the plant characteristics. As anticipated before, one other methodology is the "stop summation". It consists in controlling the saturated control action and the non saturated one, if the first one is less than the second one and the error is greater than 0, or the error is less than 0 and the not saturated control action is less than the minimum value of the saturated control action so we stop the integral summation, if not we let the control action be as always. The pros of this techinique is that it gives better response since the integral term is unfronzen once the sign of the errore changes because this occurs before tha actuator comes out of saturation. Then there is also the "incremental/velocity" alghorithm but it derives from pid so it does not fix for our case. Finnally there is the "back calculation "methodology which has been left as a future improvements.

5.2 Bumpless

Bumpless transfer is a critical concept in digital control systems, especially during transitions between different control modes, such as from manual to automatic control or when switching between different controllers. The goal of bumpless transfer is to ensure smooth transitions without causing sudden changes (or "bumps") in the control signal, which can lead to system instability, wear and tear on mechanical components, or undesired behavior in the controlled process. With respect to general control systems in state feedback control systems, bumpless transfer becomes even more complex due to the direct feedback of state variables into the control signal. State feedback controllers often have multiple state variables influencing the control action, and any discontinuity in these variables can result in a significant "bump." If the computer system keeps track of the value of the manipulated variable u, at the point at which change-over is made, the value of u is stored in a variable uc . Two methods of transfer can be used:

- U is not preset and change-over is made when the error e is zero, posing U = u_c

- U is preset but the change-over is made when the error is not zero, so the integral action term needs to be set to an initial value $Ki * s(k) = u_c - K_p e_c - U$ where e_c is the error value at change-over and $s(k)$ is the error sum at that time point. This is the one we used.

5.3 Simulink Real Controller Velocity

In the following section we will analyze the simulink implementation of the wind-up and bumpless transfer methodologies.

5.3.1 Real Controller Velocity with integral subtraction

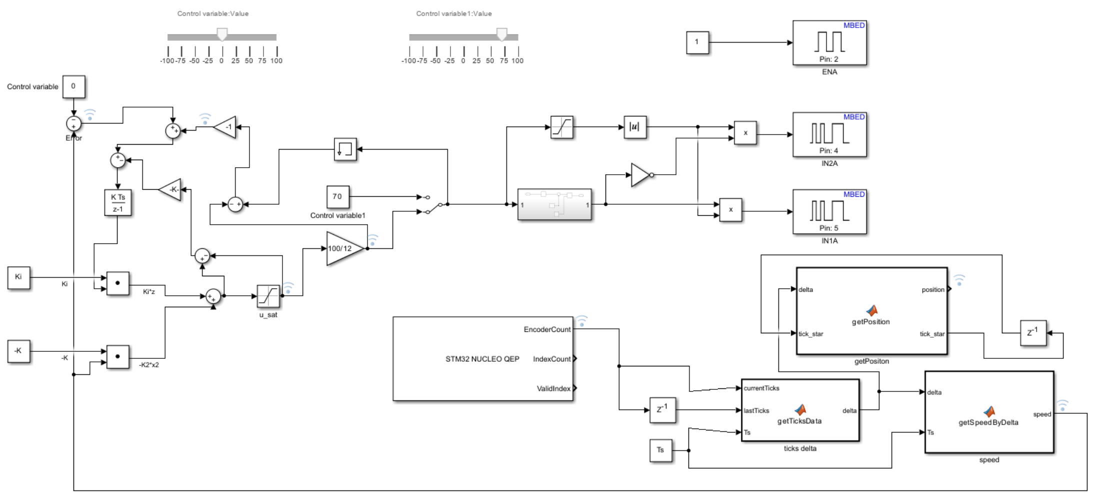


Figure 15: Real Controller Velocity with integral subtraction

Firstly we implemented the anti-windup technique of the integral subtraction, so we took the difference between the u with saturation and without saturation and subtracted it from the error, and this allowed us to avoid wind-up also in boundaries conditions. However we already said in the previous section that the limit of this technique is about the "K" parameters. In fact it could be tuned better , but this is left as a future improvement and also this is the reason why we also implemented the "Stop summarization" techinique.

About the bumpless transfer methodology we used a simular approach of the integral subtraction with the windup, beacuse we stored , thanks to a "memory" block the value of the u before the change between manual and automatic behaviorual, simulated thanks to the "switch" block, and this difference is given into the integrator, which eventually considers this "difference" before integrate the error.

5.3.2 Real Controller Velocity with Stop Summarization

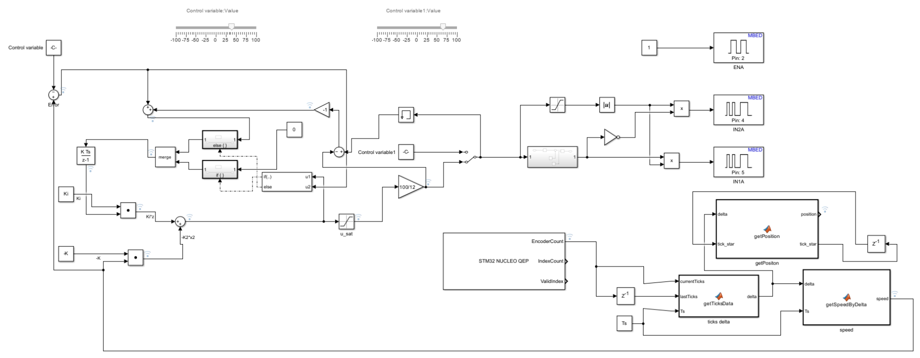


Figure 16: Real Controller Velocity with Stop Summarization

About the anti windup technique as discussed before we implemented the "stop summarization" technique also. In this case we controlled that the input control did not overcomme the saturation limits and that the absolute value of the error was greater than 0. Then if this condition is true we stop the summation of the integral term, otherwise every thing is left as normal.

5.4 Simulink Real Controller Position

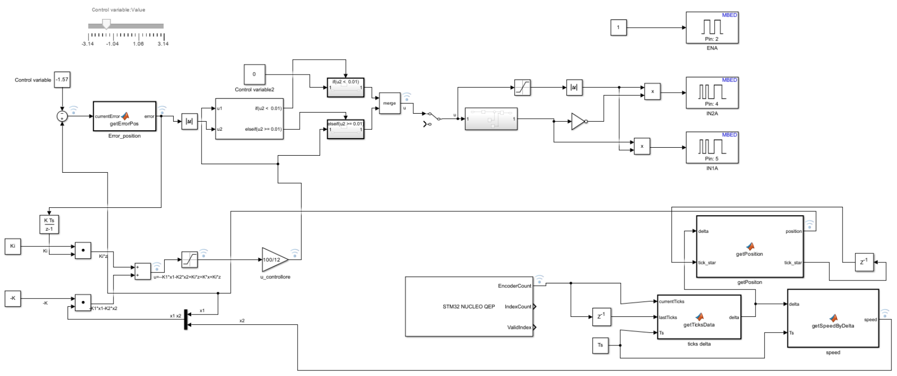


Figure 17: Rapresentation of Real Controller Position

In the case of the of the position control, we modified the error calculated with the *getErrorPos* function, that confront the error with the π value and allow the motor dc to reach the position in a more comfortable way, avoiding longer paths not needed. Furthermore we noticed that with our implementation of our position control our dc motor took a few time in order to get the exactly position decided by the referment, so we also implemented an if statement , which automatically set the speed equal to 0, when the position is approximately the one of the referement. This implementation improved the performance, due to the fact that the controller had less unusual movements as we can see in the following pictures:

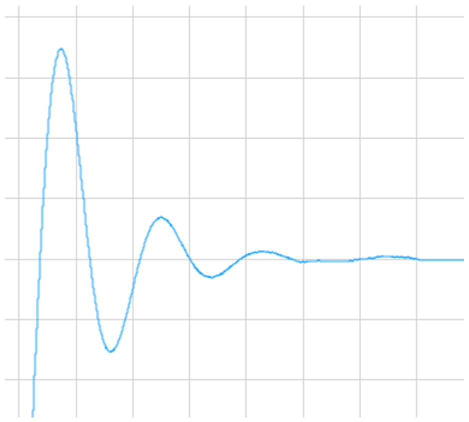


Figure 18: Position control with no if statement

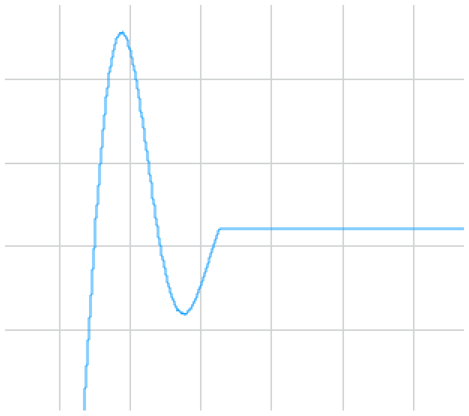


Figure 19: Position control with if statement

6 StateFlow

The final step of our design was to insert the controller inside a Chart in State-Flow and give the system not a single reference but simulate with multiple references. Our goal was to move from position control carried out with Pole Placement to position control with LQR to evaluate its performance, this was also done for speed control.

6.1 Control in StateFlow position

6.1.1 References

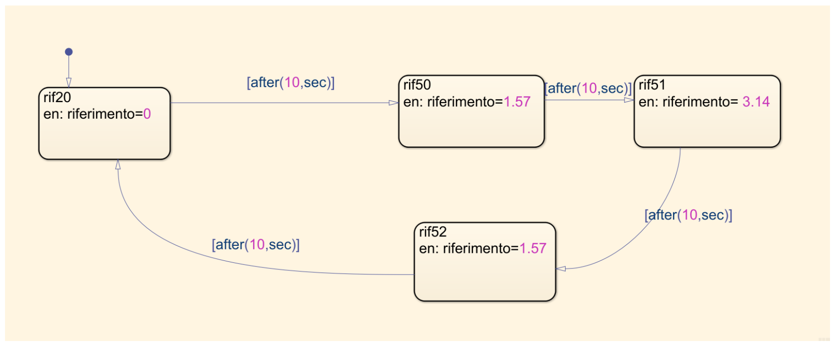


Figure 20: Position References

6.1.2 Control

As regards control, we have created two charts, connected by two arcs capable of making a switch between the two controllers. To be precise, this step is made by the control in position achieved with the gains obtained from the application of LQR and the control in position with the gains obtained from Pole Placement. This was done in order to appreciate which of the two controllers created provides us with better performance

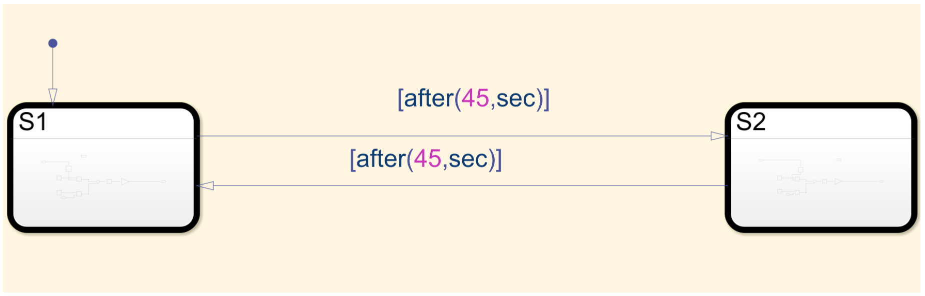


Figure 21: Controller Position in StateFlow

6.2 Control in StateFlow velocity

For velocity case we have the same as the position one.

6.2.1 References

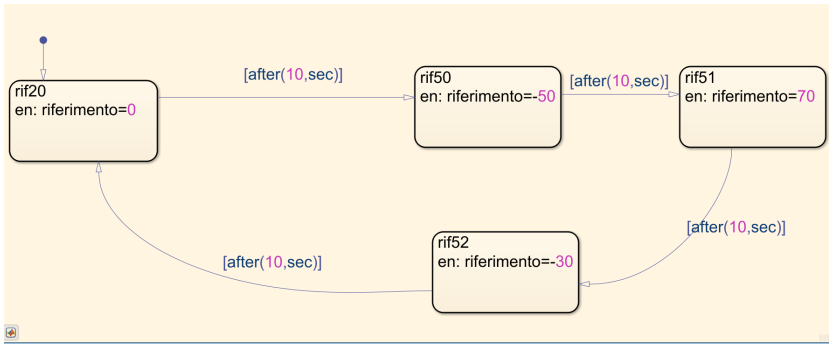


Figure 22: Speed References

6.2.2 Control

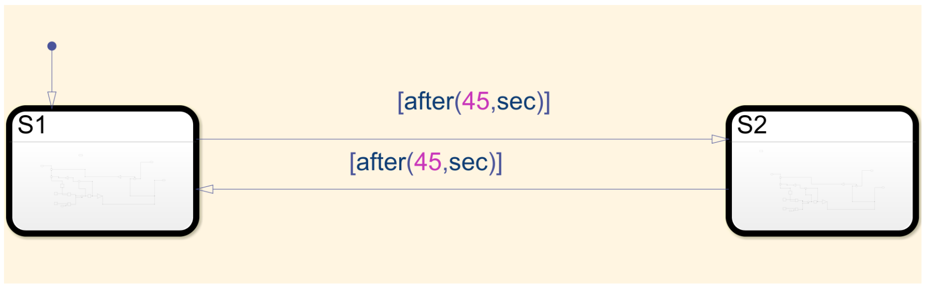


Figure 23: StateFlow Controllers

6.3 Real Controllers

Below we present the final Simulink representations of our system, one for position and one for velocity. This allowed us to carry out a real simulation capable of automatically varying the reference values and changing the controller. This allowed us to fully appreciate the advantages and possible future improvements that can be applied to the creation of our system:

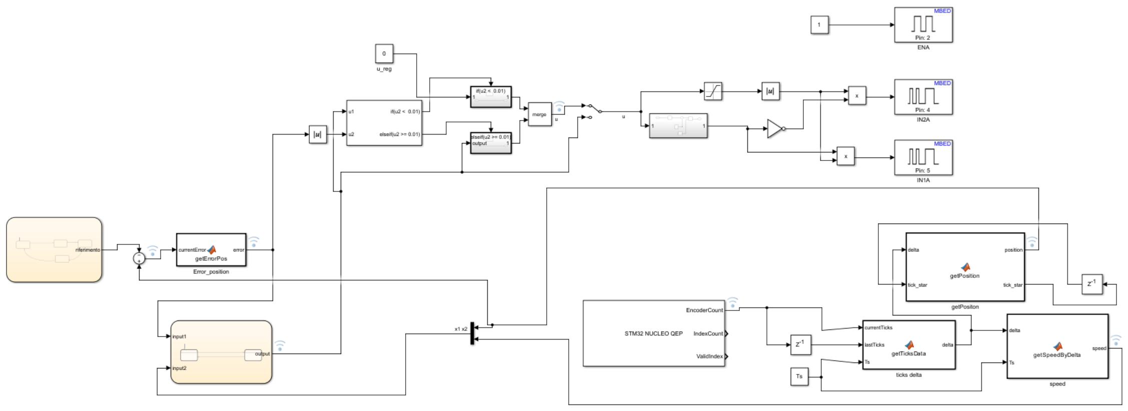


Figure 24: Real Controller Position State Flow

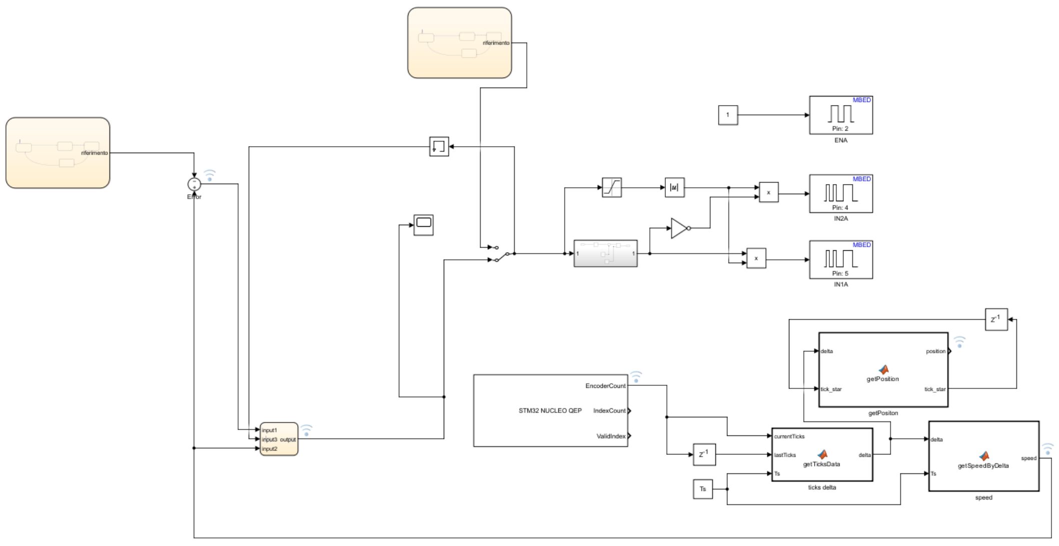


Figure 25: Real Controller Velocity State Flow

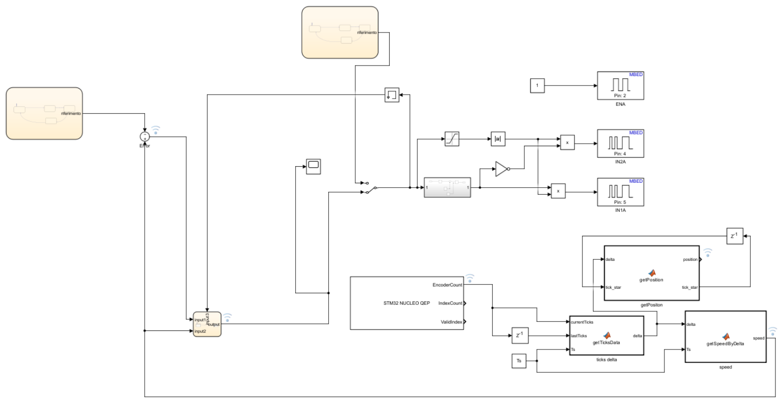


Figure 26: Real Controller Velocity StateFlow with Stop Summarization

7 Simulations

The following figures show the performance of our system with the reference values shown in the previous paragraphs. Below we show the results obtained:

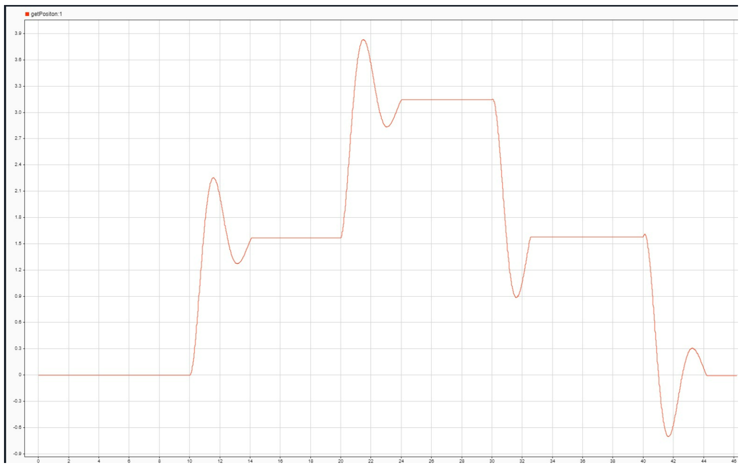


Figure 27: Results State Flow Position

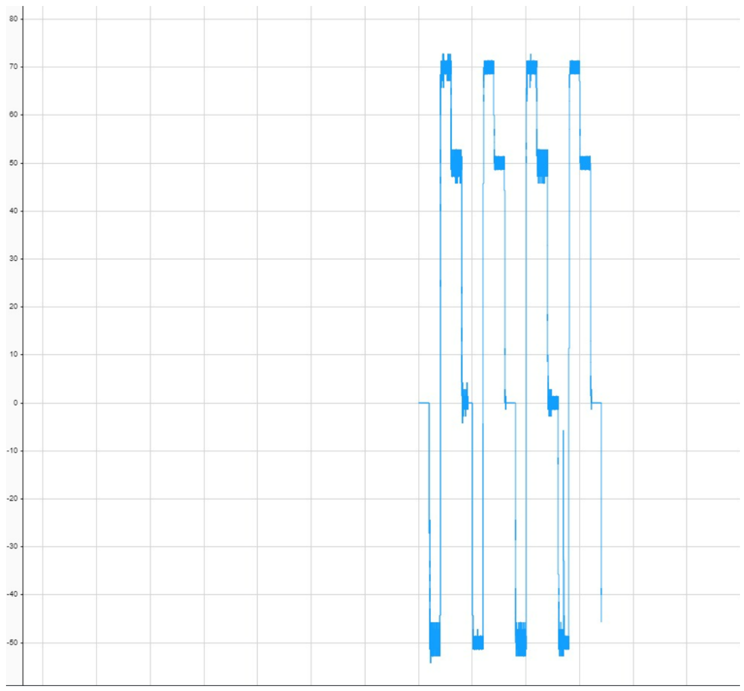
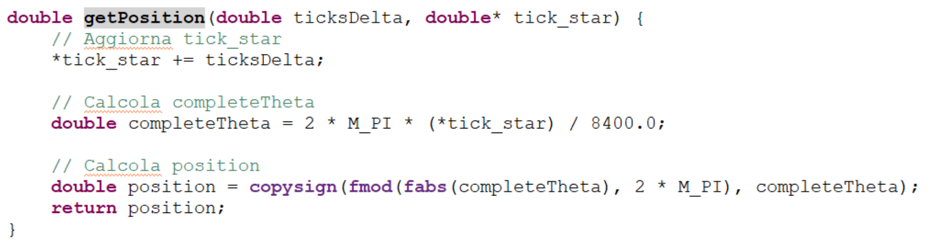


Figure 28: Results Velocity

8 Direct Coding

The final step was to carry out direct coding. Below we report the highlights of the code both in position and speed.

8.1 Position



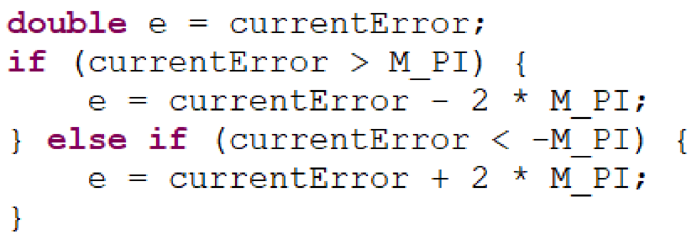
```c
double getPosition(double ticksDelta, double* tick_star) {
    // Aggiorna tick_star
    *tick_star += ticksDelta;

    // Calcola completeTheta
    double completeTheta = 2 * M_PI * (*tick_star) / 8400.0;

    // Calcola position
    double position = copysign(fmod(fabs(completeTheta), 2 * M_PI), completeTheta);
    return position;
}
```

Figure 29: Funzione *getPosition*

The getPosition function is used to calculate the angular position of the motor, based on the increment of encoder ticks. The function updates an accumulated tick value and then converts this value to a normalized angular position between $-\pi$ and π .

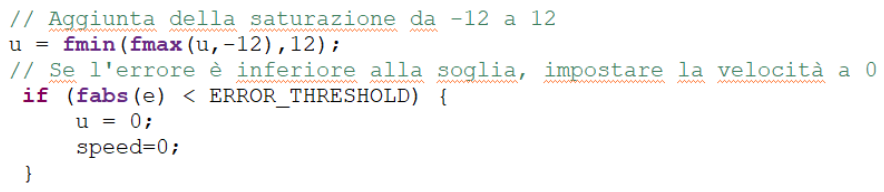


```c
double e = currentError;
if (currentError > M_PI) {
    e = currentError - 2 * M_PI;
} else if (currentError < -M_PI) {
    e = currentError + 2 * M_PI;
}
```

Figure 30: getError

These lines of C code are used to normalize an angular error *currentError* so that it falls within the range $[-\pi, \pi]$. The variable e represents the normalized error. Normalizing an angle to this range is useful for avoiding problems in calculations with angles, such as sudden jumps or discontinuities when the angle exceeds threshold values.

f_max and *f_min* are used to limit the input value within a range of $[-12, 12]$. The presence of this saturation serves to prevent the control signal u from exceeding the physical limits that the system can manage. Subsequently, a verification of the error was carried out. If the latter is lower than a predefined ERROR_THRESHOLD threshold then the input and the speed of the system are set to 0. This serves to avoid oscillations of the system when the error is



```
// Aggiunta della saturazione da -12 a 12
u = fmin(fmax(u,-12),12);
// Se l'errore è inferiore alla soglia, impostare la velocità a 0
 if (fabs(e) < ERROR_THRESHOLD) {
     u = 0;
     speed=0;
 }
```

Figure 31: Enter Caption

small, therefore indicating that the system is sufficiently close to the desired error. Last step was to formulate the discretized control input as below:

```
    double u = u_last - k_pos * position + k_pos * last_pos - 
        k_speed * speed + k_speed * last_speed +  ki * Ts/2 * e + 
        ki * Ts/2 * e_last;
```

8.2 Velocity

As regards the speed, the control input has been defined as follows:

```
   double u = u_last - k_speed * speed + k_speed * last_speed +  ki 
        * 0.0025 * e + ki * 0.0025 * e_last;
```

Here is the implementation of the anti wind-up strategies for velocity control; Integral subtraction:

```
      /*Integral subtraction implementation
    double K_aw=0.01;
    double diff=u-u_sat;
    diff=diff*K_aw;
    u=u-diff;
    // end integral subtraction implementation*/
```

Stop summation:

```
    if((u > 12 & e > 0) | (u< -12 & e < 0)){
      e=0;
    }
    else{
      ;
    }
```

WARNING: Within Direct Coding it is possible to pass from one controller to another (resp. Pole Placement and LQR) by uncommenting and commenting respectively the necessary K and K_i values (resp. for Position and Velocity).

9 Possible improvements

With the conclusion of this project we can evaluate possible future improvements that can be included within the latter. Below we present a list of possible improvements that can be made:

- Improve the gains K and Ki of the pole placement and LQR controllers. In particular, we can conclude that due to the slightly elevated gain values, we still observe slight oscillations in the real system, although this is not visible in the simulations.
- Improve the K of the speed controller with integral subtraction.
- Develop an additional system with Stateflow capable of switching from a speed controller to a position controller.