UNIVERSITÀ DEGLI STUDI DI
NAPOLI FEDERICO II

Scuola Politecnica e delle Scienze di Base

Control Lab Project Report

Self-balancing motorcycle

Academic Year 2023/2024

Ch.mo prof. Raffaele Iervolino

Lucia Schiavone - P38000204
Mario Valentino - P38000205
Rita Silva
Mariana Almeida

Contents

Chapter 1

Introduction

1.1 Project description

In this project, the control of a self balancing prototype motorcycle with a reaction wheel as balancing mechanism will be investigated. Motorcycles, when they are standing still, are unstable systems, that dynamically resemble an inverted pendulum.
The main challenge is to balance the motorcycle while standing in place, then balancing with straight motion and while steering.

The prototype consists of a small scale motorcycle with actuated rear wheel and steering wheel, provided with a rotating disk, referred as inertia wheel, whose acceleration and deceleration will serve as counteracting action for gravity and external torque to guarantee the balance.

An Arduino MKR1000 constitutes the central controller together with the Arduino MKR Motor Carrier.
The actuators to move the back wheel and the inertia wheel are two DC motors, while the actuator that moves the front wheel for steering is a servo motor.
The sensory equipment includes an hall sensor for the inertia wheel speed, an encoder for the reading of the rear wheel speed, and a 6-axis Inertial Measurement Unit (IMU).

This project is approached with Model Based Design methods: first, a model of the system is built. Then, the controller is synthesized and tested on the simulated system. Finally, the controller is deployed on the physical system and the performances are validated through multiple experiments. To this purpose, a linear PID controller and a nonlinear Sliding Mode Controller are built.

1.2 Motors testing and characterization

Before proceeding with the design of the controller, all the components have been tested to ensure their working and to achieve a complete understanding of the interactions and communication between the hardware components.

The first piece under testing was the rear wheel DC motor and its embedded incremental encoder. The motor is controlled by using the PWM technique, a motor characteristic curve has been obtained by sweeping different values of the duty cycle from -1 to $+1$ with increments of 0.05. At each duty cycle, the number of pulses is read from the rotary encoder, and the motor raw speed is computed as:

$$\text{rawSpeed} = \frac{\text{endCount} - \text{startCount}}{\text{dt}} \tag{1.1}$$

The obtained speed is in $[counts]/[s]$, to convert it in $[rpm]$ the following formula has been used:

$$\text{speedRPM} = \text{rawSpeed}/12/100.37 \cdot 60 \tag{1.2}$$

The obtained values are used to construct a Lookup Table that relates each speed value to the corresponding duty cycle. Some filtering of the data has been carried out so that just the duty cycle 0 is corresponding to a null speed, and the values of speed are monotonically increasing with respect to the duty cycle. The rear wheel speed was

computed taking into account the number of counts per revolution of the motor shaft recorded by the encoder, which was 12.

The DC motor controlling the inertia wheel is controlled using the PWM techique as well. However, the motor is not used to spin the inertia wheel at a given speed, but to generate a given torque acting on the bike. For this reason, no Lookup Table relating the duty cycle to the speed is needed. A measure of the inertia wheel speed is still required as a safety measure in case it will spin too fast and is carried out using a magnetic encoder. The sensor measures the number of passages of the magnets in the inertia wheel through the sensor. The interval of time in which the passages are counted is the *Sample Time Ts* that has been set empirically to 0.5 to guarantee both accuracy in the measurements and adequate sensitivity to the changes in velocity. In general, if the Sample Time is set too short, there is a risk of encountering intervals with zero counts despite the inertia wheel being in motion. Conversely, if the Sample Time is set too long, there is a possibility of missing variations in the angular speed of the inertia wheel that occur within a single period.

The last motor used in this project is a servomotor needed to steer the bike. The PWM signal sent to the servomotor encodes the desired steering angle.

1.3 Mathematical Model

The problem of controlling the self-balancing motorcycle can be regarded as the control of an inverted pendulum, thus it will not be a surprise that the mathematical model is very similar.

Definition of Coordinates and Parameters

Looking at the motorcycle from behind, the origin of the lean angle θ is defined along the vertical axis, it is positive when evolving counterclockwise. The rotational displacement ψ of the inertia wheel,

which is relative to the whole motorcycle, is positive defined as counterclockwise.
The hardware design determines the distribution of mass, the centre of mass of the entire system is approximately located just below the inertia wheel.
The position of the center of mass is identified by its height h_{cm} along the vertical axis.

Derivation of the Dynamic Equations of Motion

The motorcycle is free to rotate about the axis connecting the rear and front wheels where they touch the ground. It is subject to the torque resulting from gravity, inertia wheel and external actions.

$$\tau = \tau_g + \tau_{IW} + \tau_{ext} \tag{1.3}$$

where the gravitational torque is $\tau_g = M \cdot g \cdot h_{cm} \cdot \sin(\theta))$, and the torque exerted by the inertia wheel is the reaction needed to compensate gravity and external perturbation. The external torque is ideally assumed to be null, the control technique shall be robust enough to synthetize a control signal able to induce the proper reaction of the inertia wheel motor shaft in order to stabilize the unstable equilibrium point $\left[\theta, \dot{\theta}\right] = [0, 0]$. Thus, the inertia wheel torque can be rewritten as $\tau_{IW} = -\tau_{motor,IW}$.

The mechanical system can be regarded as the composition of two rigid bodies, the motorcycle structure and the inertia wheel, whose dynamics are ideally decoupled.

The state space variables are chosen as following

$$x_1 = \theta$$
$$x_2 = \dot{\theta}$$
$$x_3 = \psi$$
$$x_4 = \dot{\psi}$$

The control input is the action exerted by the inertia wheel motor

$$u = \tau_{motor,IW}$$

The project's objective is focused on controlling the motorcycle dynamics only

$$\dot{x}_1 = \dot{\theta} = x_2$$
$$\dot{x}_2 = \ddot{\theta} = \frac{1}{I_M} \cdot (M \cdot g \cdot \sin(x_1) - u)$$

where I_M is the moment of inertia of the motorcycle.

Chapter 2

PID Control

2.1 Model In The Loop

The first step of Model-Based-System Design is Model in the Loop design. The hardware in the simulation environment is modeled. The controller is developed and tuned on the simulated environment.

This process used in is useful to learn the behavior of the system, and provides valuable insights into the strengths and weaknesses of the model, which is vital for the development process.

The first stage of the design was to build an accurate model using Simulink Simscape. Figure 2.1 shows the resulting scheme and Figure 2.2 shows the 3D model used to visually assess the result.

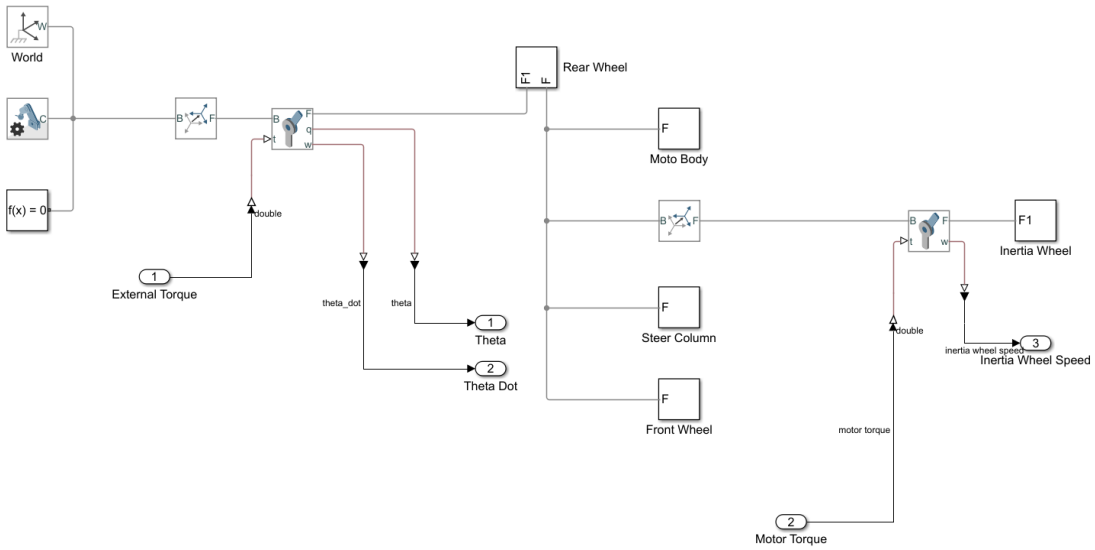

Figure 2.1: Motorcycle Simscape model

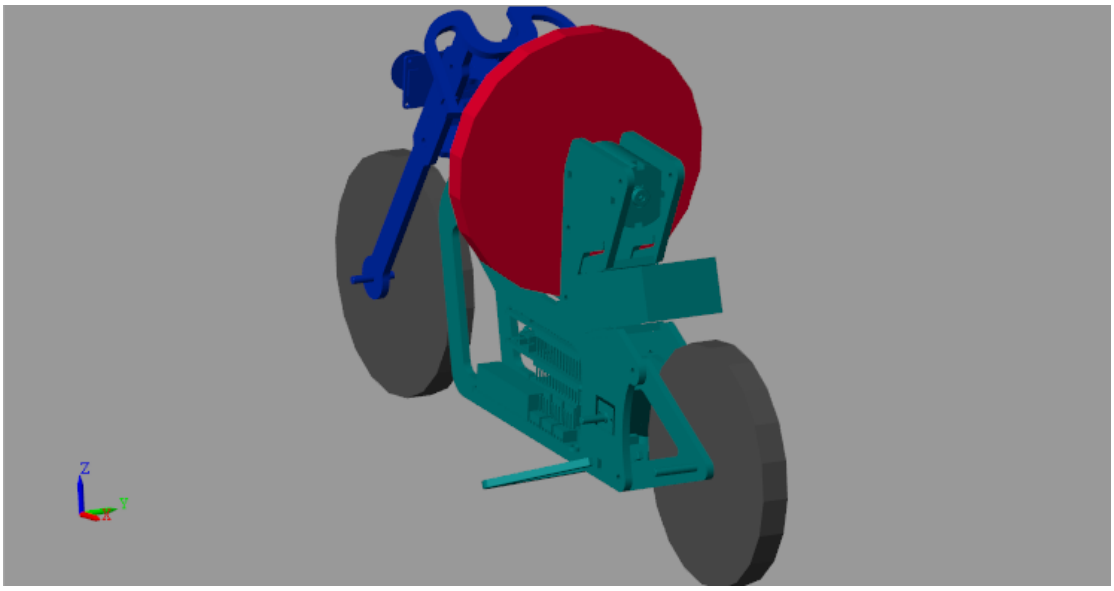

Figure 2.2: Motorcycle 3D model

A standard PD controller has been used, whose parameters have been set by trial and error. The scheme and the values used for the controller are represented in Figure 2.3. The constant $2 \cdot 10^{-2}$ is the stall torque constant of the motor.

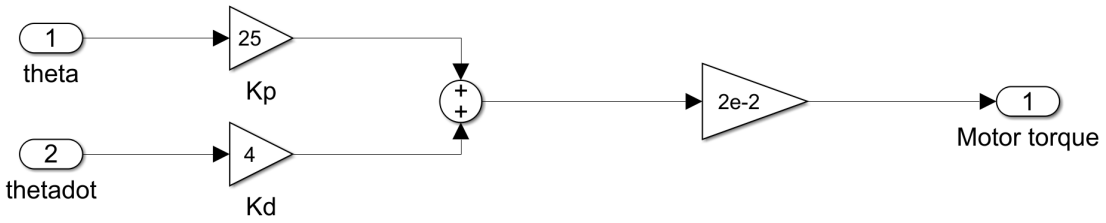

Figure 2.3: PD Controller

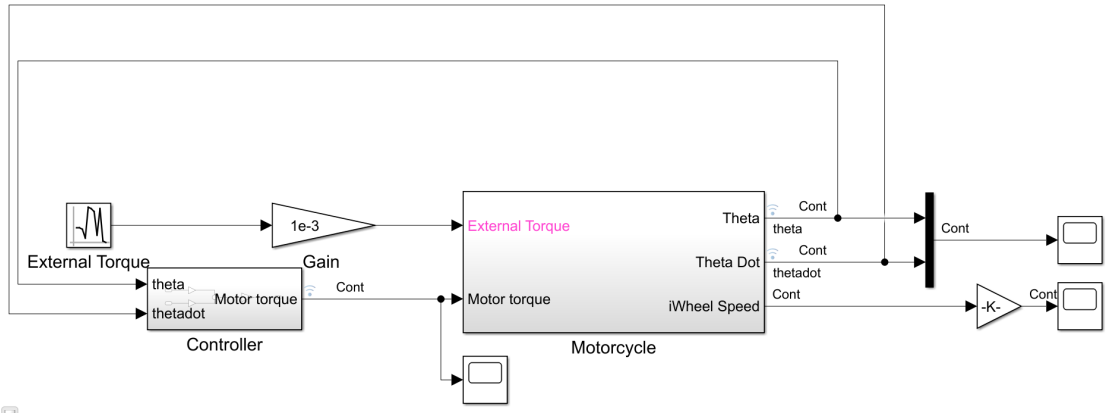

Figure 2.4: Closed loop scheme for MIL

The following pictures show the results of the simulation by also considering a small external torque of gain $1 \cdot 10^{-3}$ to simulate the external disturbances acting on the motorcycle.

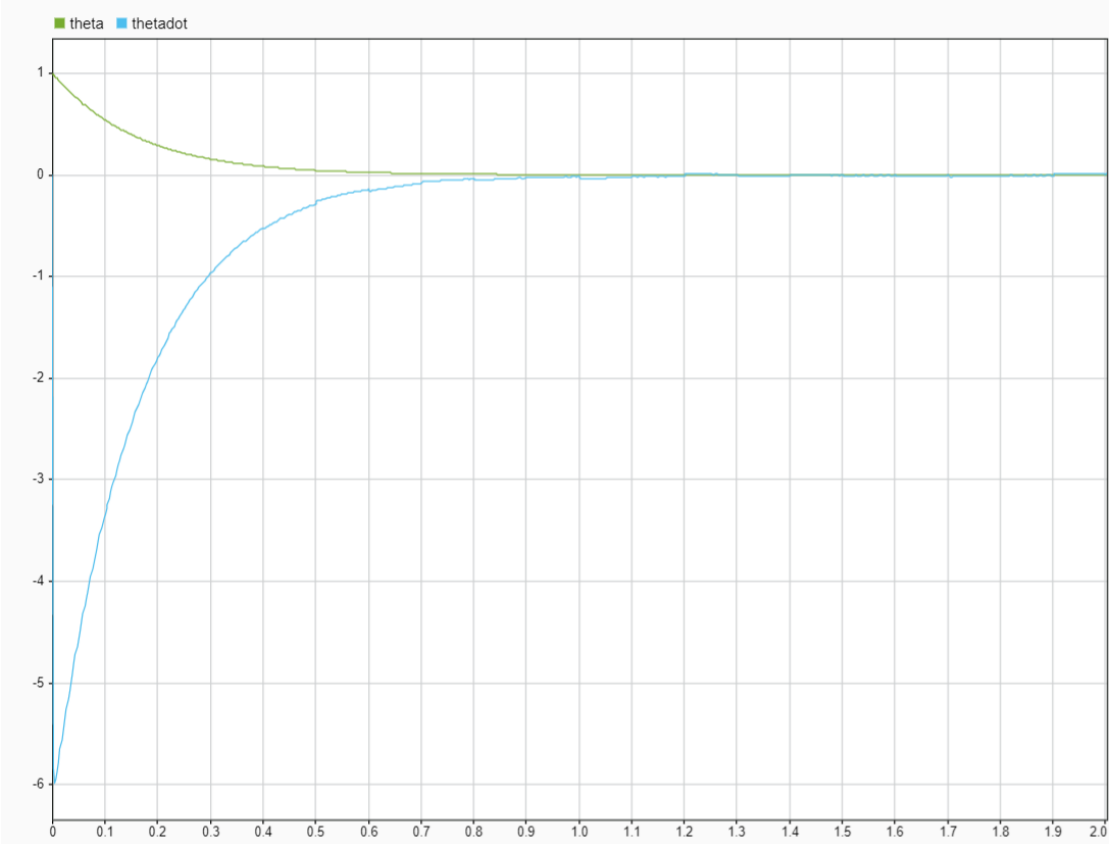

Figure 2.5: Position and velocity with respect to the vertical axis

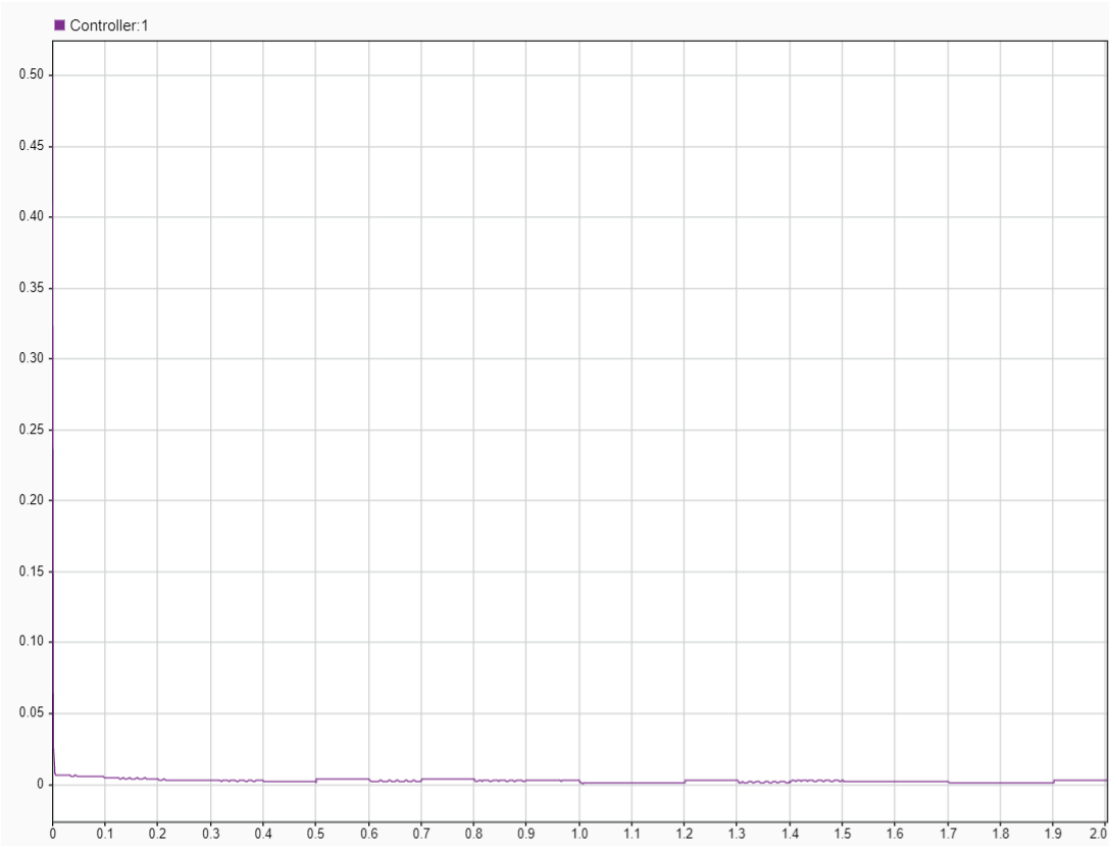

Figure 2.6: Control values

2.2 Software In The Loop and Processor In The Loop

Software-in-the-loop (SIL) is a step that utilizes a simulated environment to evaluate and validate the performances of the controller, when the actual code is run. It involves running the software code on the simulated model. This allows us to identify and correct software errors early in the development cycle, before implementing it on the hardware.

Processor-in-the-Loop testing (PIL) refers to the testing and validation of embedded software on the processor that will later be used as a controller. PIL tests are performed to check whether the compiled code also works on the target processor, and the control action is performed on the simulated environment.

The performances of the controller should be the same with either the simulated controller, the compiled controller, and the embedded controller.

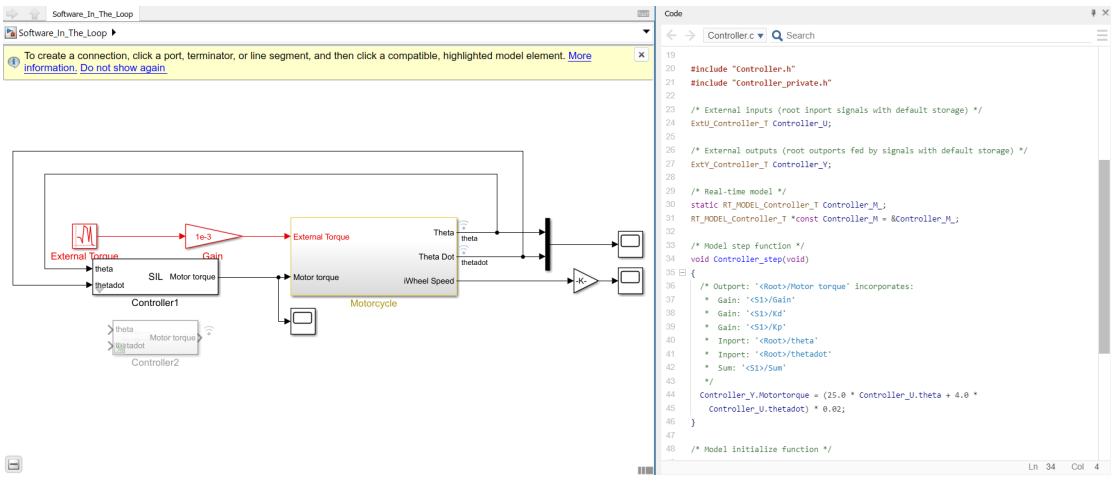

Figure 2.7: SIL Simulation scheme

Both in the SIL and PIL simulations, the performances of our controller were unchanged. Therefore, it was possible to continue with

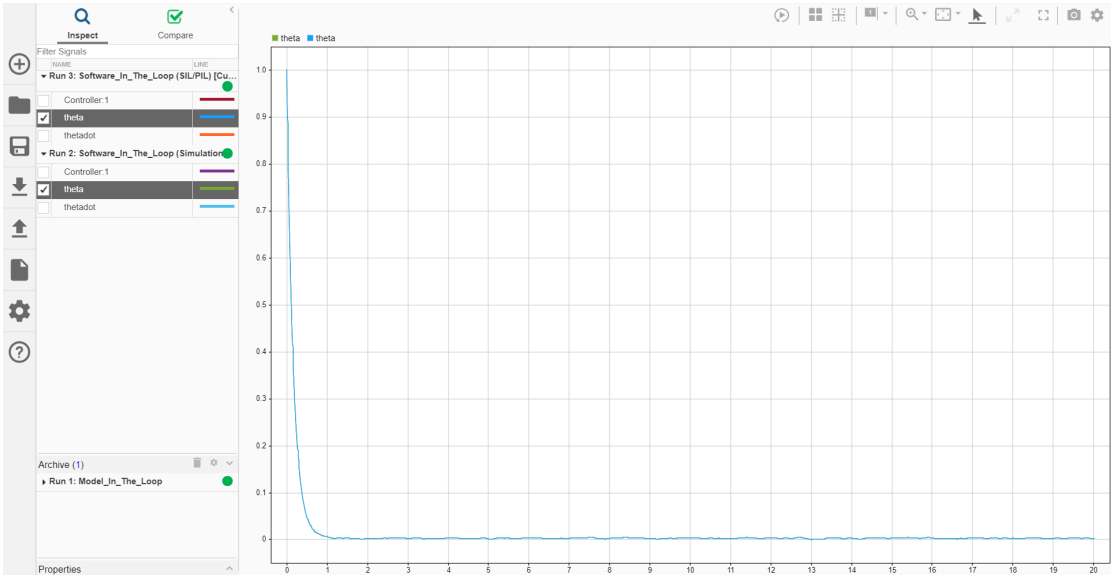

Figure 2.8: SIL Performances

the next step.

2.3 Hardware Implementation

2.3.1 Safety features

While implementing the controller on the hardware, it was crucial to add some failsafe features. Since it is essential to maintain overall stability and safety, the system must run between its operational limits.

At first, the controller was producing torques outside of the allowable range when evaluated in the scope window. To rectify this issue and enforce torque limits, a saturation block was introduced into the Controller subsystem immediately before the Torque command outport block and the established torque limits were set between -0.7 and 0.7.

Furthermore, the hardware has a controller that accelerates the inertia wheel in the correct direction according to the sign of the lean angle. So, it may be necessary to shut off the inertia wheel automatically under some circumstances. One of those risks, is that the inertia

wheel could spin too fast which would damage the motor. In order to avoid this possibility, a third inport was introduced in the Controller subsystem, and the signal was routed into a new scope block for real-time observation during simulation. Then, a product block was introduced to determine the ultimate torque command, considering the controller output and the subsystem output.

After this safety procedure, it was developed a mechanism to automatically shut off the inertia wheel motor when the lean angle deviates significantly from its stable balancing range in order to prevent the motorcycle from entering an unstable state where balance cannot be effectively maintained. While watching the behavior of the lean angle, it showed that if it was beyond more or less 6 degrees it could make the motorcycle unstable and incapable of reversing direction. It was added an interval test block that served as an indicator of the motorcycle's stability. A logical operator block was inserted to implement an AND gate with inertia wheel speed as an additional condition.

Then, the focus was directed towards preventing the inertia wheel motor's operation unless the IMU is calibrated. To address potential variations in calibration, the IMU is only considered calibrated if it met calibration thresholds. The subsequent step involved latching the IMU Calibrated signal to a value of 1 upon initial calibration, maintaining this state indefinitely. This way, it was implemented an inport for the IMU Calibrated signal, introducing a Unit Delay block, and a Logical Operator block within the Controller subsystem for effective latching. The IMU Calibrated latched signal was then integrated into existing logic and displayed in the status panel.

The ultimate failsafe feature involves deactivating the inertia wheel when the battery level drops too low. It was inserted a battery read block in order to provide real-time voltage readings for continuous observation.

Afterwards, to ensure the voltage remained within an acceptable range, the measurement was converted to real-world voltage. Then,

the converted voltage was integrated into the controller subsystem and compared to a 9 V threshold.

2.4 Balancing the motorcycle

2.4.1 Balancing in place

The controller tuned in the Model in the loop step is then implemented on the hardware.

From a practical point of view, the IMU sensor may be affected by a bias. At each run, it is necessary to find the equilibrium point of the motorcycle and sum a bias so that the equilibrium point corresponds to $\theta = 0$ in the controller.

Consequently, with the use of a Constant Block, a sum of the symmetrical value of the bias to the angle of the motorcycle would then be performed.

After some tests were made, an integral action has been added to increase the robustness of the controller. The integral action, that was not needed in the simulated environment, becomes useful in the actual environment because of all the external disturbances and errors in the sensor reading.

Figure 2.10 shows the result of a simulation on the hardware.

2.4.2 Balancing in straight motion

When balancing the motorcycle with straight motion, the connection between the motorcycle and the computer was made via Wi-Fi. In this part, the motorcycle was required to balance itself while driving forward with the use of both the inertia wheel and the control of the rear wheel.

After the torque command of the rear wheel, a filter was made to smooth out the step reference when turning on the control. This was made to prevent the motorcycle from falling when turning on.

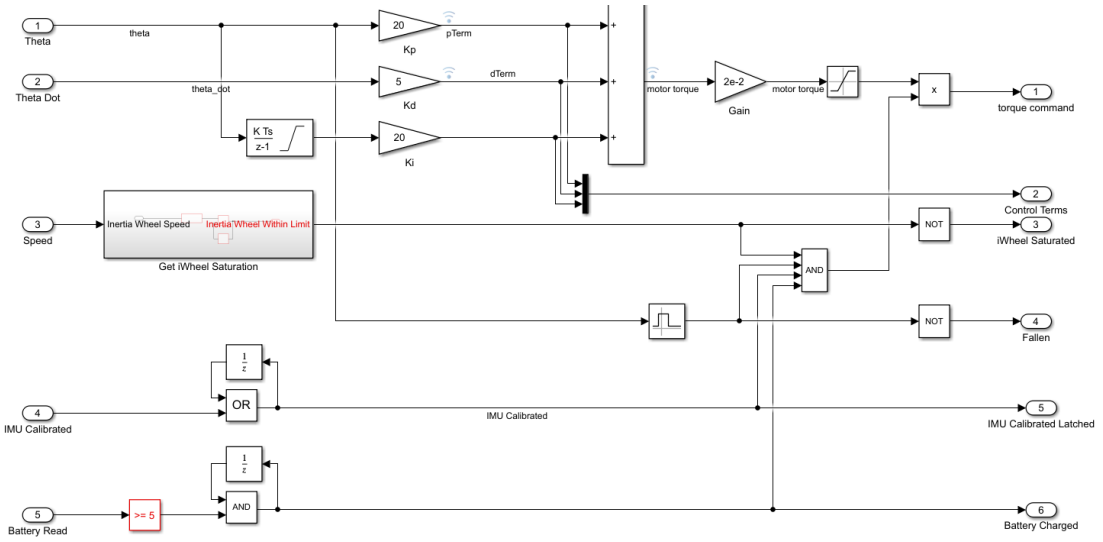

Figure 2.9: Simulink scheme of the PID controller

The value of the torque was adjusted while testing the straight motion, taking into account that the front wheel was pointing directly forward. This was done by adjusting Servo Motor 3 to minimize the bias, using a Constant Block.

The image below pictures the motorcycle moving in a straight line for about 20 seconds. This can be seen from the perturbation at the 90 second mark until the 110 second mark where the motorcycle runs with approximately constant velocity and theta, while compensating it.

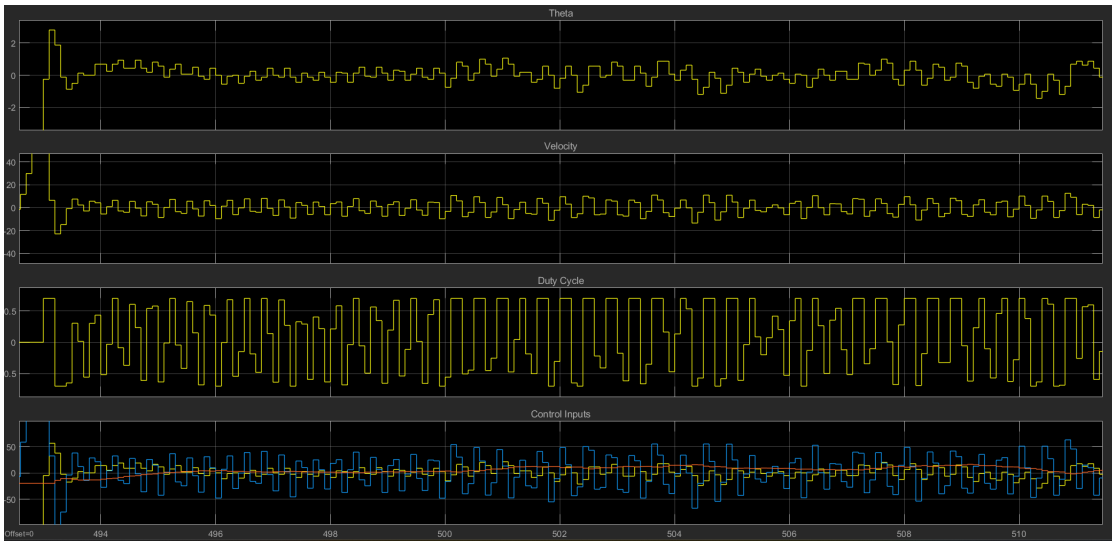

Figure 2.10: Balancing in place results

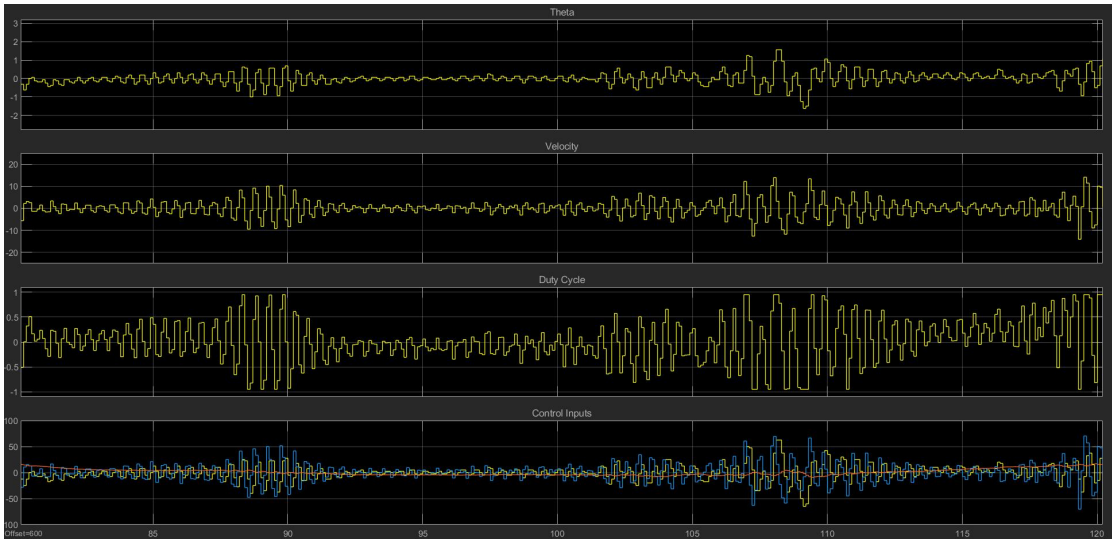

Figure 2.11: Balancing while moving straight results

2.4.3 Balancing while steering

The final challenge of our project was to keep the motorcycle balanced while it was steering. The front wheel of the motorcycle has servomotor that allows us to turn the wheel to a desired angle. Figure 2.12 shows the block connection that allows us to control the servomotor. A bias

block is used to compensate for possible misalignment of the wheel, and a first order filter is used to smooth out the control input. A saturation is used to avoid the servomotor impacting with the mechanical limits of the motorcycle.

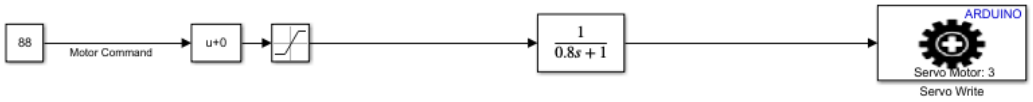

Figure 2.12: Front wheel servomotor control

Chapter 3

Sliding Mode Control

3.1 Mathematical Formulation

Given the motorcycle SISO nonlinear system of the form

$$\dot{x} = f(x) + g(x)u$$

where $f(x) = \begin{bmatrix} x_2 & \frac{1}{I_M}\left(M_M g h \sin(x_1)\right) \end{bmatrix}^T$ and $g(x) = \begin{bmatrix} 0 & -\frac{1}{I_M} \end{bmatrix}^T$.
The main control specification to achieve balance is the stabilization of the point $[x_1 \quad x_2]^T = [0 \quad 0]^T$.
Sliding Mode Control requires a switching control function

$$\sigma : \mathbb{R}^n \to \mathbb{R}$$

and a switching control

$$u(x) = \begin{cases} u^+(x,t) & \text{if } \sigma(x,t) > 0, \\ u^-(x,t) & \text{if } \sigma(x,t) < 0. \end{cases}$$

such that:

1. when $\sigma(x) = 0$ the control specifications are met
2. the sliding surface $\Sigma := \sigma(x) = 0$ is globally attractive

3. the trajectories on Σ evolve according to a sliding vector field $F_s(x)$ with the desired properties

The selected scalar function of the state is $\sigma = p_1 x_1 + p_2 x_2$ with $p_1, p_2 > 0$. It is verified that, when $\sigma = 0$, then $[x_1 \quad x_2]^T = [0 \quad 0]^T$.

Recalling the Lyapunov function method leads to the following choice for the control law:

$$u(x) = -\mathcal{L}_g(\sigma) k \cdot \mathrm{sgn}(\sigma) - \frac{\mathcal{L}_f(\sigma)}{\mathcal{L}_g(\sigma)}$$

Attractivity is guaranteed since the transversality condition $\mathcal{L}_g \neq 0$ is met, meaning that the vector field $g(x)$ is not tangent to Σ in such a way to be effective in pushing the trajectories towards Σ itself.
Computation of Lie derivative is the following

$$\begin{aligned}
\mathcal{L}_f &= \nabla\sigma \cdot f = [p_1 \quad p_2] \begin{bmatrix} 0 & -\frac{1}{I_M} \end{bmatrix}^T = -\frac{p_2}{I_M} \\
\mathcal{L}_g &= \nabla\sigma \cdot g = [p_1 \quad p_2] \begin{bmatrix} x_2 & \frac{1}{I_M}\left(M_M g h \sin(x_1)\right) \end{bmatrix}^T \\
&= p_1 x_2 + \frac{p_2}{I_M}(M_M g h \sin(x_1))
\end{aligned}$$

The control law is then

$$u = \frac{I_M}{p_2} k \cdot \mathrm{sgn}(\sigma) - \frac{I_M p_1 x_2 + p_2 \left(M_M g h_{cm} \sin x_1\right)}{p_2}$$

Deriving the equivalent control, it can be shown that the closed loop system is:

$$\begin{cases} \dot{x}_1 = x_2 \\ \dot{x}_2 = -\frac{p_1}{p_2} x_2 \end{cases}$$

3.2 Model In The Loop Simulation

Since we changed the structure of the controller, we had to go back to the first stage of the model based design and test the proposed controller in a simulated environment. In the simulation scheme represented in 2.4, the controller subsystem has been replaced with the one in Figure 3.1.

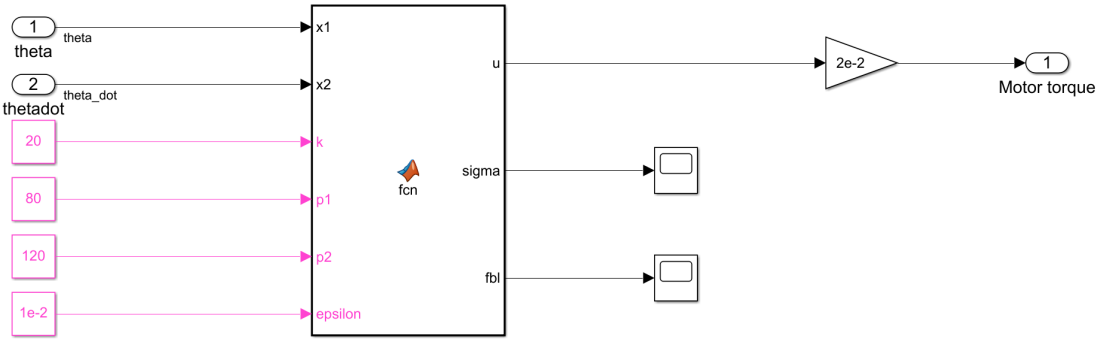

Figure 3.1: SMC Controller

The following code has been implemented, where the parameters have been estimated experimentally. The results are shown in Figure 3.3. The control action is much more aggressive than the PID. Therefore, validation on the hardware is required to check whether the actuators are able to deliver such control action.

3.3 Hardware Implementation

The SIL and PIL steps followed as the PID part. The results were the same as the MIL so it was possible to proceed to the hardware implementation of the controller in Figure 3.4.

Sliding Mode Controller was able to reduce the oscillatory behaviour of the motorcycle, keeping it almost perfectly still. However, the stress induced on the actuator was higher leading to a faster heat up of the DC Motor moving the inertia wheel.

```
1   function [u,sigma, fbl] = fcn(x1,x2,k, p1, p2, epsilon)
2   Im = 10;
3   Mm = 0.470;
4   g = 9.81;
5   h = 7.5;
6
7   sigma = p1*x1 + p2*x2;
8
9   lieG = -p2/Im;
10  lieF = p1*x2+p2/Im*Mm*g*h*sin(x1);
11
12  fbl = -lieF/lieG;
13
14  if sigma <= -epsilon
15      u = k/lieG + fbl;
16  elseif sigma >= epsilon
17      u = -k/lieG + fbl;
18  else
19      u = 0;
20  end
21
```

Figure 3.2: MATLAB Code implementing SMC

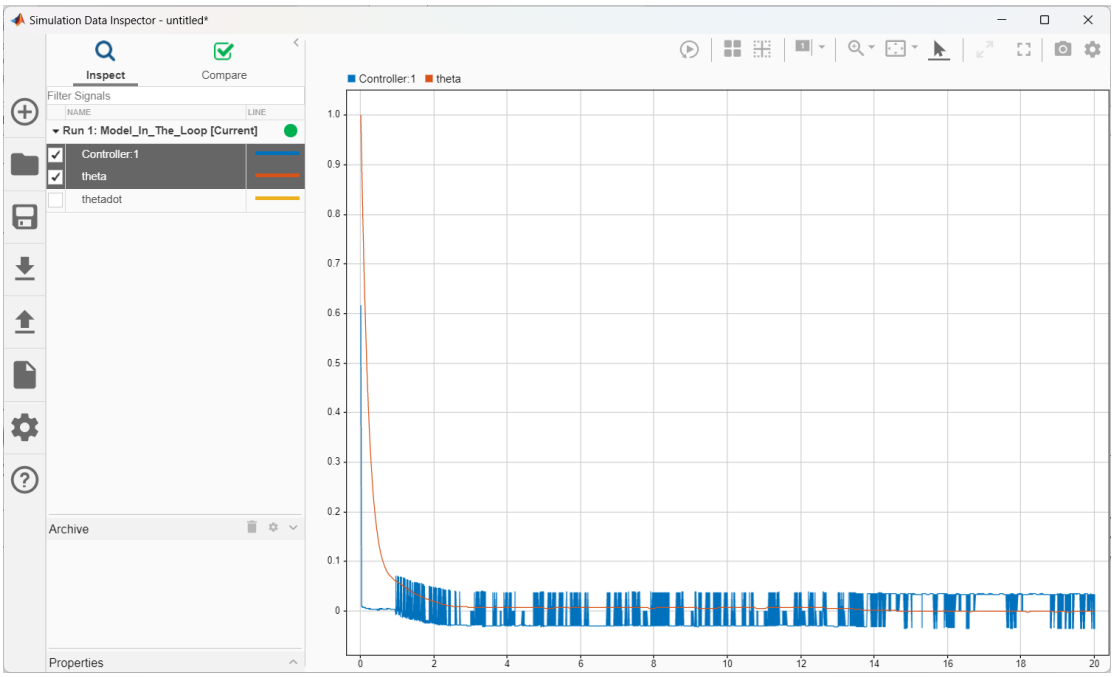

Figure 3.3: Results in the MIL simulation

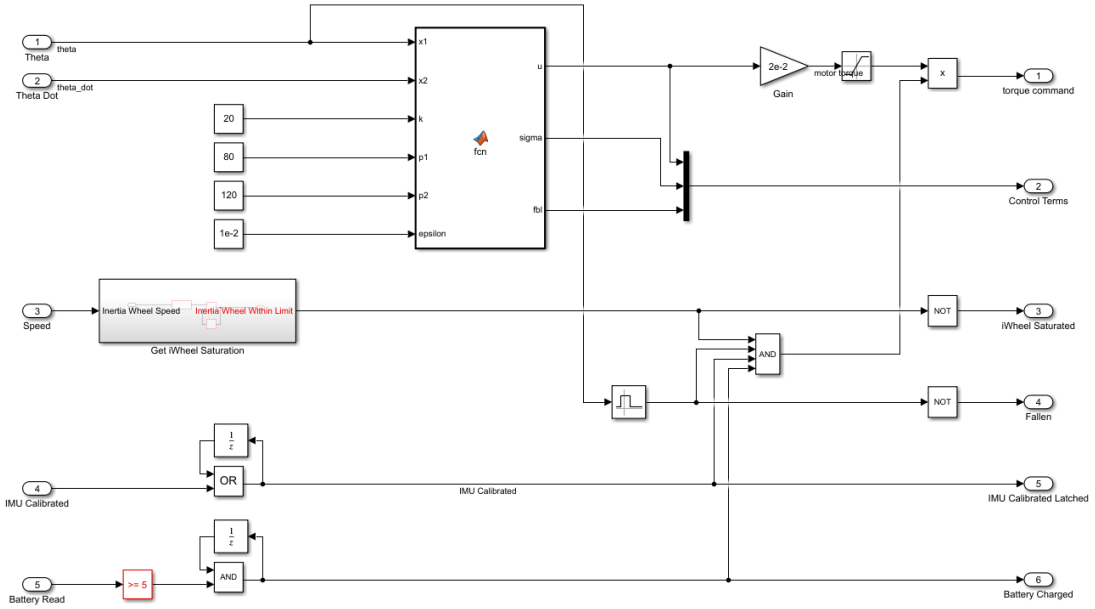

Figure 3.4: Hardware implementation of SMC

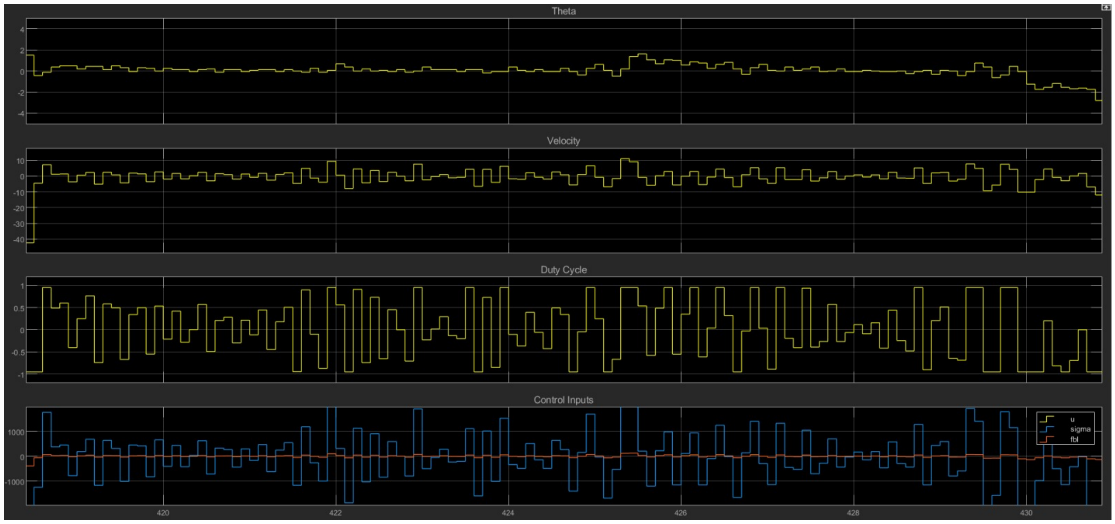

Figure 3.5: Results of the SMC

Chapter 4

Conclusion

In conclusion, we used the Model Based Design approach to build a controller that stabilizes the vertical, unstable, position of the motorcycle.
We first built a model of the system, used to simulate and identify its behaviour. Then, a control strategy has been decided. The first control strategy was the PID that according to Model Based Design has initially been implemented in the simulation environment (MIL). After assessing the performances of the controller on the simulated environment, the actual code has been compiled and its performances were evaluated against the MIL ones (SIL). Since the code actually runs on a microcontroller, the next step is to verify if the microcontroller is capable of executing such code (PIL). A further step would be required: assessing the performances of the control law on a real time hardware that simulates the plant. This step has been skipped due to the lack of the necessary hardware.

Finally, the PID control law is implemented on the physical system and the performances are validated. Some fine tuning of the parameters was required to get the same performances on the real system since there are always some unmodeled dynamics, mostly actuators nonlinearities. In particular, a small integral action has proved to be useful in robustify the controller.

Model Based Design is a flexible design approach that allows us to easily revise earlier stages of the design and adjust it if the performances degrade as we approach the final hardware implementation. As we decided to implement a Sliding Mode Controller to balance our bike, we had to go back to the Model In The Loop stage. The same process MIL - SIL - PIL has been carried out and afterwards, the performances have been validated on the real system.

Both controllers allowed us to achieve balancing of the bike, PID with a greater oscillatory behaviour, and SMC with a more robust dynamic, but with more stress on the actuators.

Many problems naturally araised during the implementation of each of these stages, some of which include:

- IMU not calibrating properly
- Bias on the IMU changing over time leading to an incorrect reading of the vertical axis
- Maximum sampling time given by the transfer rate between Simulink and Arduino, degrading the performances of the controller
- Servomotors performances not adequate
- Exhausted battery
- ...

Each problem has been successfully solved by the team at its stage of the design, allowing us to achieve the final requirement of the controller.