Self-Balancing Motorcycle

Control Lab

Master's degree in Automation Engineering and Robotics

Self-Balancing Motorcycle project

Authors:
Claudio Anzalone
Paolo Maisto
Antonio Manzoni
Gaetano Parigiano
Angelo Vittozzi

Students number:
P38000173
P38000191
P38000234
P38000195
P38000184

Contents

1 Introduction

In this the "Self-Balancing Motorcycle" project developed using the Arduino kit represents an innovative exploration of the dynamics of inverted balance. The key challenge addressed by this project is to design and implement a system capable of autonomous balancing on two wheels, emulating the complex dynamics of an inverted pendulum.
The motorcycle is controlled by an Arduino MKR1000, the Arduino MKR Motor Carrier, a DC motor to move the back wheel, an encoder, a DC motor to control the inertia wheel, a 6 axis IMU, a standard servo motor to steer the motorcycle's handle, a distance sensor, and a tachometer.
As first step, we have started to become familiar with the components of the our Arduino Engineering Kit and with the tools that we will be using. Next, we tested the components using Matlab and Arduino IDE.
In this project, we learned how to simulate the vehicle's overall behavior and create models to program the hardware components and improve the quality of the control algorithms. We've also explored how to program the motorcycle with Simulink, control its balance algorithm, make it move in a straight line, and make it curve.
In the chapters that make up this report of the project, it has been explained the following topics:

- Chapter 2: model vehicle and learn how to simulate its behavior:
  - introduction to the different tools; in particular, the *Simulation Data Inspector* that help us to display and compare signals between simulations;
  - PID controllers have been applied to controlling the angle at which the motorcycle leans;
- Chapter 3: understanding sensors and actuators and use Simulink and Arduino to program these;
- Chapter 4: building a balance control algorithm using the Simulink models that we've developed for the different hardware components;
- Chapter 5: balancing the motorcycle when it follows a straight line;
- Chapter 6: balancing the motorcycle while steering and using the lean angle to our advantage.

Figure 1: Participants of the project

2 Model vehicle and simulation

The motorcycle is a vehicle which regulates balance via an inertia wheel system which is controlled by implementing a Proportional-Integral-Derivative (PID) controller and in this chapter it will be described the work carried out to achieve this aim in simulation.
In particular, the following topics will be addressed in this chapter:

- it will be described the physical dynamics of the motorcycle-inertia wheel system;
- it will be explained the choices of the control terms in a PID controller and their respective effects in a closed-loop control system;
- it will be shown the carried out simulations;
- it will be explained the model developed algorithm that will balance the vehicle in a variety of external conditions.

2.1 Motorcycle equations of motion

Let's begin by understanding the equations of motion for the motorcycle-inertia wheel system. The examined system is a simplified version of the system such that the functioning simulation is not too complex.
We will make the following assumptions:

- the motorcycle is free to move only about the wheel-ground axis[1];
- there is no rotational friction between the motorcycle and the ground or between the motorcycle and the inertia wheel;
- the motorcycle wheels' thickness is negligible;
- the air resistance is negligible.

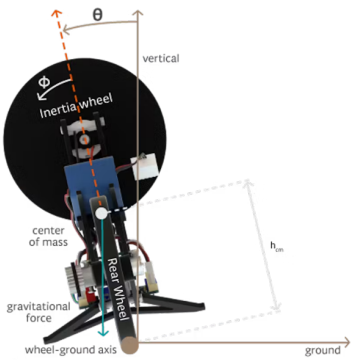

Figure 2: The diagram of motorcycle-inertia wheel system that shows the coordinates along with important physical quantities

In the Fig. 2 the motorcycle is looked at from behind. The most important coordinate shown in the Fig. 2 is the lean angle θ which is equal to 0 degrees when the motorcycle is perfectly upright. θ is positive when the motorcycle leans counterclockwise when viewed from behind, and negative when the motorcycle leans clockwise. Another important angle is the rotational displacement of the inertia wheel, ϕ. As the inertia wheel rotates relative to the rest of the motorcycle, ϕ varies in value, where positive displacement is defined as counterclockwise.
An important point of the system to the dynamics of the motorcycle is the center of mass of the entire motorcycle system (including the inertia wheel) because the gravitational force acts directly through this point. The center of mass location is approximately just below the inertia wheel. The height of the center of mass from the ground when the motorcycle is upright ($\theta = 0$) is defined as h_{cm}.
As already said above when the assumptions are listed, the motorcycle system is free to rotate around the wheel-ground axis. The inertia wheel can rotate along its own axis, which goes directly through the middle of the inertia wheel.

[1]The wheel-ground axis, which is an imaginary line that connects the points on the rear and front wheels where they touch the ground

Now let's think about the torques in the motorcycle system. In a system that is free to rotate along an axis, a torque occurs when a force is exerted on the system at some point of the rotational axis. The magnitude of the torque can be determined by using the following equation:

$$\tau = |F \cdot r \cdot \sin(\alpha)| \tag{1}$$

In the equation (1), the torque, denoted as τ, is expressed in units of force · length (e.g., Newton-meters). Here, F represents the applied force, r is the distance between the point where the force is applied and the axis of rotation, and α is the angle measured at the point of force application between the force direction and the axis of rotation. According to the equation, the maximum torque is achieved when the force is applied at a right angle (90 degrees). The torque is zero, when the force is applied directly toward or away from the axis of rotation. Furthermore, it is possible to get greater torque with the same force by applying the force farther from the axis of rotation.
For a given axis of rotation, there is a net torque, which is the sum of all torques acting on the system about that axis. The net torque is proportional to the angular acceleration of the rotating system:

$$\tau_{sys} = \sum_i \tau_{i,sys} = I_{sys}\ddot{\theta}_{sys} \tag{2}$$

In this equation, $\tau_{\mathrm{i,sys}}$ represents the torque exerted on the system around the rotational axis. I_{sys} is the moment of inertia of the system with respect to the rotational axis. The moment of inertia is a property of a rigid object that measures the mass distribution of the object with respect to a rotational axis. $\ddot{\theta}_{\mathrm{sys}}$ is the angular acceleration of the object about the rotational axis.
The motorcycle system includes the rear and front wheels, the steering column, the motorcycle body (along with all its embedded electronics), and the inertia wheel. The axis of rotation for this system is the wheel-ground axis, and the measure of rotation is the lean angle θ, defined earlier.
The torque acting on the motorcycle (denoted with M) about the wheel-ground axis has 3 major components:

- gravitational torque ($\tau_{\mathrm{g,M}}$);
- inertia wheel torque ($\tau_{\mathrm{IW,M}}$);
- external torque ($\tau_{\mathrm{ext,M}}$);

The gravitational torque is due to the gravitational force pulling down on the center of mass of the motorcycle system when it is not directly above the wheel-ground axis (i.e. when it is leaning). Using the first equation from above, the gravitational torque can be found as follows:

$$\tau_{g,M} = M_M \cdot g \cdot h_{cm} \cdot sin(\theta) \tag{3}$$

In the above equation, M_{M} is the mass of the motorcycle system, g is the constant gravitational acceleration, h_{cm} is the height of the center of mass from the wheel-ground axis, and θ is the lean angle. In this equation (and all subsequent equations in this chapter), a positive torque value is defined to be in the counterclockwise direction with respect to the understood rotational axis. Obviously the gravitational torque is zero when the motorcycle is perfectly upright ($\theta = 0$). For a small lean angle θ, a small gravitational torque is exerted on the motorcycle in the same direction as the lean. As the angle gets larger, the torque increases until the motorcycle falls 90 degrees. If this were the only torque acting on the motorcycle, the motorcycle system would have an unstable equilibrium at $\theta = 0$. This means that any small perturbation of the motorcycle from its equilibrium will cause the system to accelerate away from its equilibrium position rather than move back towards it.
However, there are other torques in the motorcycle system to prevent it from falling. When the inertia wheel motor is actuated, the motor shaft will exert a torque on the inertia wheel, accelerating it (rotationally). The inertia wheel will then exert an equal and opposite torque on the motor shaft due to conservation of angular momentum. Let's call this reaction torque $\tau_{\mathrm{IW,M}}$.
The last torque in the system in exam to describe is external torque, $\tau_{\mathrm{ext,M}}$. This torque encompasses all other torque sources acting on the system. This could include things like the USB cable pushing up against the motorcycle, air resistance, wind, and various other disturbances like pushing the motorcycle to the side with your finger. In this first theoretical part of the project, it will considered an ideal scenario in which $\tau_{\mathrm{ext,M}} = 0$. Ultimately, the goal of the project is to balance the motorcycle even when there is a reasonable amount of external perturbations acting on it.
Thus, we have the net torque on the motorcycle about the wheel-ground axis:

$$\tau_{net,M} = I_M \cdot \ddot{\theta} = \tau_{g,M} + \tau_{IW,M} + \tau_{ext,M} \approx \tau_{g,M} + \tau_{IW,M} \tag{4}$$

M_{M} is the moment of inertia of the motorcycle system about the wheel-ground axis.
Now let's look at another rotational axis: the one that goes through the inertia wheel motor shaft. On this rotational axis, only the inertia wheel is free to rotate. The torques acting along this axis are from the DC motor, which we will call $\tau_{\mathrm{motor,IW}}$, and from dissipative forces like friction between the moving parts and drag effects, $\tau_{\mathrm{fric,IW}}$. It will assumed that this mechanical resistance is zero in this system, so $\tau_{\mathrm{fric,IW}} = 0$. Using the same torque definition, we have the following equation:

$$\tau_{net,IW} = I_{IW}\ddot{\phi} = \tau_{motor,IW} + \tau_{fric,IW} \approx \tau_{motor,IW} \tag{5}$$

Here, I_{IW} is the moment of inertia of the inertia wheel with respect to its central axis, and $\ddot{\phi}$ is the angular acceleration of the inertia wheel. When the inertia wheel DC motor will be actuated through the MKR1000 board you will have direct control over $\tau_{\mathrm{motor,IW}}$.
Next, let's put it all together. Recall the torque that the motor shaft exerts on the inertia wheel is equal and opposite to the torque that the inertia wheel exerts on the motor shaft and thus, the rest of the motorcycle.

$$\tau_{IW,M} = -\tau_{motor,IW} \tag{6}$$

Substituting into the equation (4) we have:

$$\tau_{net,M} = \tau_{g,M} - \tau_{motor,IW} \tag{7}$$

So, using equations (4), (4) and this last equation we can say that:

$$I_M \cdot \ddot{\theta} \approx M_M \cdot g \cdot h_{cm} \cdot sin(\alpha) - \tau_{motor,IW} \tag{8}$$

The motion of the inertia wheel is defined by:

$$I_{IW} \cdot \ddot{\phi} = \tau_{motor,IW} \tag{9}$$

We now have the dynamics of the lean angle expressed in terms of measurable constant quantities (i.e. masses, lengths, and moments of inertias). The goal will be to know the $\tau_{\mathrm{motor,IW}}$ so that the motorcycle stabilizes at $\theta = 0$.
To start, let's examine the simplest possible corrective torque that may turn the unstable equilibrium point into a stable equilibrium point. Suppose the lean angle is very small (perhaps a few degrees). The following approximation holds for small values of θ (when measured in radians): $sin(\theta) \approx \theta$.
substituting this approximation in equation (8) We obtain this simpler differential equation for the lean angle:

$$I_M \cdot \ddot{\theta} \approx M_M \cdot g \cdot h_{cm} \cdot \theta - \tau_{motor,IW} sin(\theta) \approx \theta \tag{10}$$

Now, suppose we program the motor to exert a torque that is proportional to the lean angle itself:

$$\tau_{motor,IW} = K_p \cdot \theta \tag{11}$$

K_{textp} is a constant whose optimal value may be determined via simulation and experimentation. Substituting into the previous equation, we have:

$$I_M \cdot \ddot{\theta} \approx -(K_p - M_M \cdot g \cdot h_{cm})\theta \tag{12}$$

If K_{p} is greater than $M_{\mathrm{M}} \cdot g \cdot h_{\mathrm{cm}}$, then the angular acceleration of the motorcycle about the wheel-ground axis is in the opposite direction of the lean angle. Specifically, when the motorcycle leans one way, it will accelerate in the opposite direction. When the motorcycle moves to the other side of the equilibrium point, the angular acceleration will then switch directions to move the motorcycle back toward equilibrium. This is a stable equilibrium, which means when the system is perturbed from the equilibrium position, it will restore

itself back toward equilibrium.
Solving the differential equation, we have a solution of the form:

$$\theta(t) = Asin(\sqrt{\frac{K_p - M_M \cdot g \cdot h_{cm}}{I_M}}t + B) \tag{13}$$

In the above equation, A and B are constants that depend on the initial values of θ and $\dot{\theta}$. If you increase the constant K_{textp}, the oscillation frequency becomes faster. In this ideal scenario, in the absence of any perturbing torques, the motorcycle system will oscillate about the vertical position with constant amplitude forever. This is known as proportional control and is often a first attempt at developing a control algorithm to bring a system to some desired state. We will improve this behavior so that the oscillations decay in amplitude and eventually settle to a constant lean angle of 0 and the motorcycle system responds stably when random perturbations act on it. In the following paragraph, Simulink has been used to model these dynamics.

2.2 Physical modeling in Simulink

Once you understand the equations that govern the motorcycle's dynamics, the next step is to implement the motorcycle's physical behavior, as described in the previous section, in a subsystem block Simulink that will be called *Motorcycle*.
This block has as inputs that are an external torque from the environment and a torque applied to the motorcycle by the inertia wheel motor. The model inside the subsystem generates as outputs the physical quantities of interest: the lean angle θ, its time derivative $\dot{\theta}$, and the time derivative of the inertia wheel angle, $\dot{\phi}$.

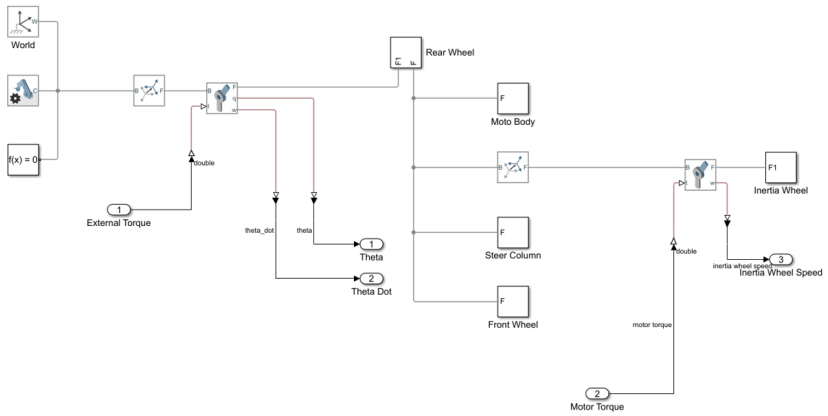

Figure 3: Physical model in Simulink

The physical dynamics of the motorcycle are modeled here using the Simscape Multibody add-on. This is a part of the Simscape modeling software package and enables you to simulate and animate multi-component systems in which physical masses are connected by geometric constraints. Notice in this subsystem there are green signals with no arrows which represent physical connections between components. The blocks that connect to these Simscape signals represent physical components and rotational and translational transforms between them and various reference frames. There are blocks for each of the 5 independently movable parts of the motorcycle: the motorcycle body, the rear and front wheels, the inertia wheel, and the steering column. These blocks contain the component geometries, masses, moments of inertia, and reference coordinates for each of the 5 parts. There are *Revolute Joints* block connected to the *Rear Wheel* and the *Inertia Wheel* component. Finally, Simulink-PS Converter and PS-Simulink Converter blocks let you convert Simscape signals to and from Simulink signals, so that it can interface with the rest of the model.

2.2.1 First tests

Now, starting from a Simulink file provided by Matlab called *motoSys0_start.slx*, we carried out some tests. The Simulink scheme is shown in the picture 4.
This model contains a *Motorcycle* subsystem block, *External Torque* block to generate a data array, *Gain* block that is a signal multiplier, and a *Constant* block to provide data input as motor torque.
This first test is carried out to see how the motorcycle behaves when no external torque and no motor torque are applied. Obviously, this is a situation when the inertia wheel is not applying any torque which is not how

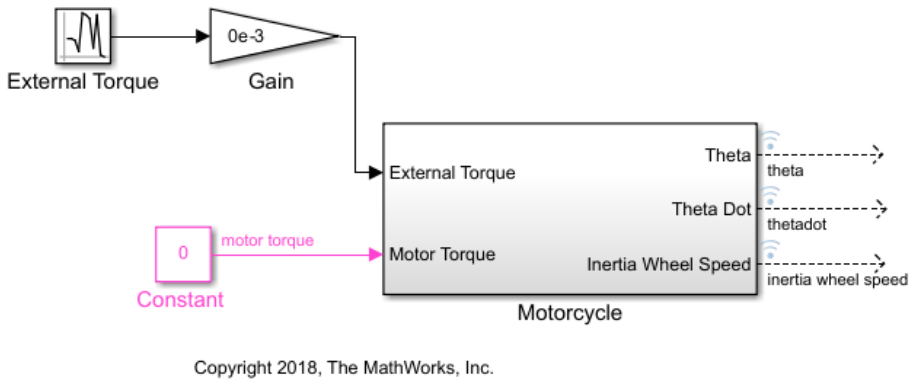

Figure 4: Simulink model of *motoSys0_start.slx*

the motorcycle will behave in the real world.
In order to watch 3D simulation when run the model, it has been used the Simscape Multibody animation when it pops up in the *Mechanics Explorer* window. The following image displays a frame of the simulation:

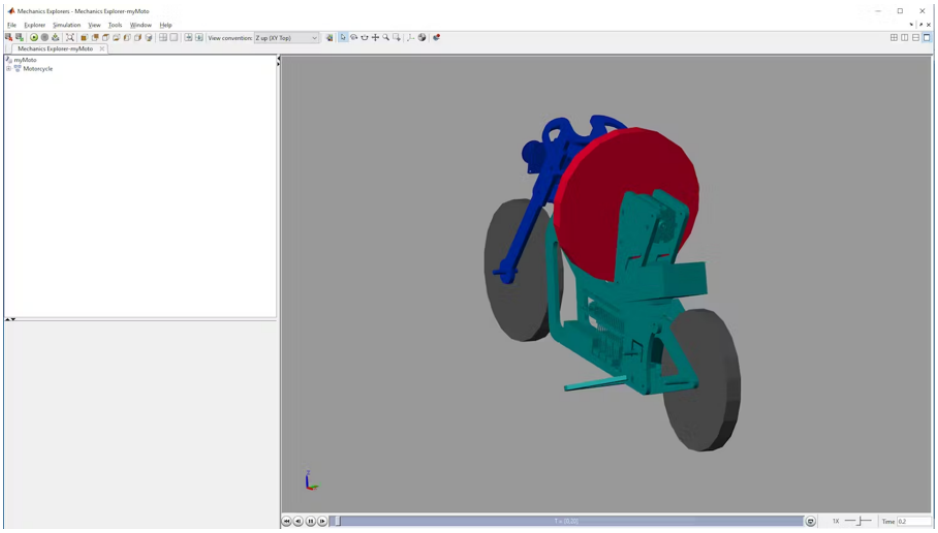

Figure 5: A frame of the simulation on the *Mechanics Explorer* window

In the animation window, you will see a 3D model with different objects represented in different colors. The models has been prepared in terms of the independent moving parts in the motorcycle: the front and rear wheels, the steer column, the motorcycle body, and the inertia wheel.
The simulation window has a full physics engine that can simulate gravity. Without any help from the inertia wheel motor, the motorcycle simply rotates about the wheel-ground axis, oscillating about the downward direction due to gravity. The model contains no physical "ground" and continues revolving right through where the ground would be. We will focus our attention on the behavior of the system when the motorcycle lean angle is close to zero.
In order to watch a quantitative look at the motorcycle's behavior we used *Simulation Data Inspector* (SDI) button in the *Simulink toolstrip*. SDI is an environment where you can view logged data from simulations, examine them in any combination of graphical axes, and compare the same signals across multiple simulations. It has been used SDI to observe the behavior of the motorcycle system and it has been useful to fine-tune the control algorithm to optimize performance examining the comparison between the behaviors of the different trials.
In the SDI window, we observed the *Motorcycle* block outputs graphs so of the *theta*, *thetadot* and inertia wheel speeds. If you remember the view of the Motorcycle block, there were just three outports: *theta*, *thetadot*, and inertia wheel speed. These signals present an oscillatory behavior and we observed the inertia wheel oscillates in response to the motorcycle's motion about the wheel-ground axis so its shape shows the same oscillatory behavior of *thetadot* with same frequency but inverted similar shape.
We have explored how the model behaves changing the initial conditions (*theta* and *thetadot*) thanks to com-

parison of the data behaviors of the previous simulations. Furthermore, we checked the simulation to see what happens to the motorcycle in the different trials.
The second test has been done adding torque motor on the inertia wheel so we have to first set the initial conditions and so we set the value other than zero in the *Constant* block. We observed the effect on the lean angle (*theta*) and the inertia wheel speed when compared to the previous run: the lean angle continues to oscillate but now has an added component that makes it decrease at a constant speed and in the real world, this means that the Motorcycle will fall, but since there is no "ground" as such in the model, the motorcycle rotates around its horizontal axis; while the average speed of the inertia wheel increases indefinitely and this is not possible in the real world since the motorcycle will break at some point.
At this point, we need to design a controller so that the motorbike stays upright and it doesn't fall over.

2.2.2 Designing a Controller for Balance

In this paragraph it has been explained the implementation of the PD controller that will take the lean angle of the motorcycle as parameter, together with the variation of the angle over time and thanks to these it will control the inertia wheel's speed, which will, in turn, help balance the motorcycle.
Starting from the last model (*myMoto.slx*) we added some changes in order to balance the motorbike. To help keep the motorcycle balanced, we need to figure out how to use some of the information we are collecting from *theta* and *thetadot* so that we can compensate for the gravitational torque acting on the motorcycle. In terms of the equations derived earlier, we want to always keep θ (*theta* on the model) close to zero to balance the motorcycle. The most straightforward way to balance the motorcycle is to apply a torque that is proportional and opposite to θ. To do this, we deleted the *Constant* block and added a *Gain* block, that was as *Kp* to signify that this is a classic proportional control (the "P" of PID), and we set a gain value. So we applied a proportional torque to the inertia wheel in the same direction as θ.
So, we tried to run the simulation with different values of the gain and we examined the signal *theta* in the SDI window for different simulations in order to see the stability of the motorcycle and the oscillatory behavior.
The increasing the proportional gain causes the oscillatory frequency to increase, and the amplitude of oscillation to also increase. If you set the gain too low, there will not be enough torque from the motor to keep the motorcycle balanced against the gravitational torque. If the proportional gain is set too high, the motorcycle will overcompensate and push too far past the center of balance, causing a fall.
Found the value for the proportional gain that stably balances the motorcycle, we ran the simulation to get a set of baseline data into SDI so we will use this data later as a reference while improving the design of the controller. Although the found controller is stable, θ oscillates about the desired lean angle with constant amplitude indefinitely. So we modified the control algorithm so that the oscillations in theta dampen over time. To do that, you not only need to counteract the lean angle, but you also need to counteract the angular rate of the lean angle. So we added another *Gain* block and an *Add* block to the model. Set the gain value of the new *Gain* block to 5e-3. This *Gain* block is labeled the new "*Kd*" as it will be introducing the derivative term of the PID controller. Make the connections so that the final configuration looks the same as the following schematic:

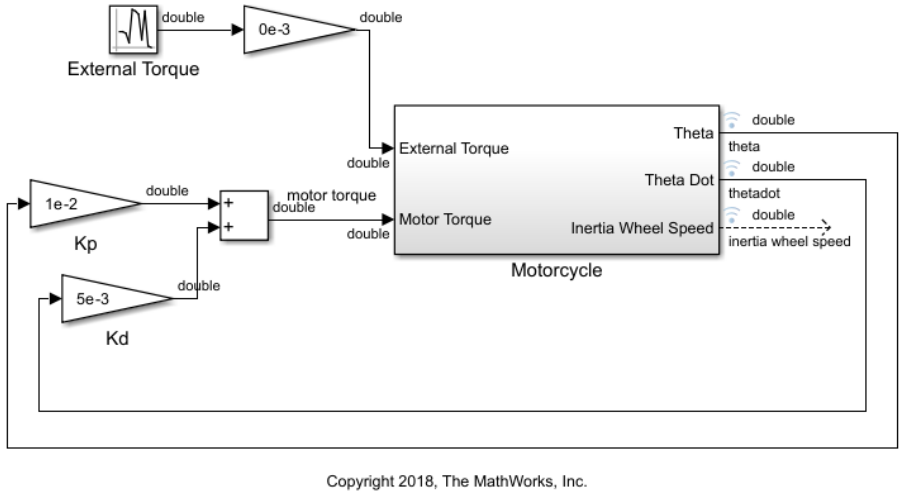

Figure 6: Simulink model of *motoSys0_start.slx* with proportional term Kp and derivative term Kd of PID controller

Running the model and watching the animation we examined theta in the SDI view and compared it to theta

from the previous run. The derivative gain impacts the oscillations in theta, in particular the oscillatory behavior is now eliminated or "damped" over time, making *theta* become constant and thus, reaching a steady state. If you think about it, once the motorcycle has reached the perfect vertical position ($theta = 0$), you don't need to keep the inertial wheel accelerating since there is no external force to compensate against. However, if the steady state is reached before the motorcycle has reached the vertical position, stopping the acceleration of the wheel is not such a good idea. Note that the steady state is reached having *theta* at a constant value different from zero, which means that the motorcycle will not be in balance. To summarize, we want the system to reach a steady-state of zero in a certain amount of time (not too soon, not too late).
Now, We experimented with the derivative gain. When the derivative gain is increased, there are fewer oscillations before the lean angle reaches a steady value, and the amplitude of oscillation decreases faster. When it is decreased, there are more oscillatory cycles, and it takes longer to reach a steady state. If the derivative gain is set too high, the motorcycle's lean angle may change abruptly, which can cause it to become unstable. For the motorcycle's balance controller oscillations in the lean angle are not desirable, but neither is having a large change in the lean angle over a short time.
So, the PD controller responds to different random stimuli which factors (Kp and Kd) matter the most to counter the undesired torque on the motorcycle that could make it fall. Both the proportional Kp and derivative gain Kd can be tuned so that the system behaves as required when perturbed from the target state. We tried different combinations of Kp and Kd gain values to see if we can achieve the desired behavior empirically i.e. that the motorbike does not fall. We found values of the proportional and derivative gain which allows theta to stay within 0.05 degree of its steady-state value before 1 second has elapsed. The following SDI diagram shows what we observed when we set Kp = 1e-2 and Kd = 1e-3:

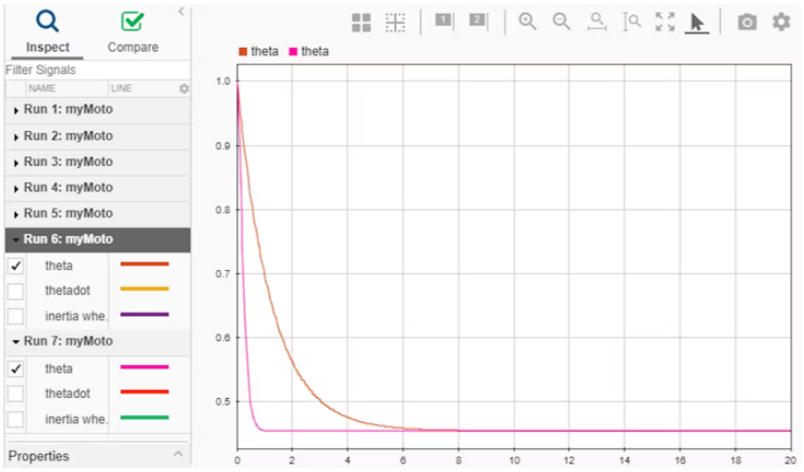

Figure 7: SDI diagram shows *theta* with this conditions: Kp = 1e-2 and Kd = 1e-3

We grouped of all elements of controller and we created a subsystem named "*Controller*".
Now, let's take care of the steady state error. A pure proportional-derivative controller, or PD controller, commonly suffers from steady state errors following a perturbation from the target state. This can be solved this by either increasing the proportional gain or by adding an integral term to the controller that counteracts the accumulation of errors over time. The accumulation of errors over time can be disadvantageous when you are trying to balance the motorcycle later. Because of this, we used the first approach of increasing the proportional gain.
We tuned the PD (proportional-derivative) controller, with different values for the gains until we found a good compromise i.e. the inertia wheel on your motorcycle should be able to help compensate the lean angle at a reasonable speed such that there are no oscillations on the lean angle. The behaviors with Kp = 5e-1 and Kd = 8e-2 gains are good.
To simplify parameter tuning and prepare for a motor torque constant later, we added an overall gain to the sum of the control terms. To do so, add a *Gain* block after the final *Add* block and set the gain value. Then, we corrected the Kp and Kp gain values by multiplying the existing values by gain value. By adding the final multiplier, it is easier to play with the model when it comes to fine tuning the controller to respond to random variations of the torque.
Until now the PD controller works well under ideal conditions with the setup described previously. It remains to be seen whether it will perform well with random perturbations. To adding a random noise we set the *External Torque* gain value to 1e-2 in the Simulink model. After running the simulation we examined *theta* in the SDI view and compare it to the previous run. According to the data obtained from the simulation, we tried

changing the Kp and Kd values to see whether you can compensate for the error. The iterative tuning process that we have done is testing different noise levels and continuing to tune the proportional and derivative gains to optimize stability, reaction speed, and steady state accuracy.
Changing the values empirically will have given you an idea of which control terms are doing most of the work to stabilize the motorcycle. To understand much more we observed at some of the signals in the PD controller. In particular, we plotted the output signals of the Kp and Kd gain blocks that are in *Controller* subsystem block. We ran the simulation and examined the control terms in this SDI view:

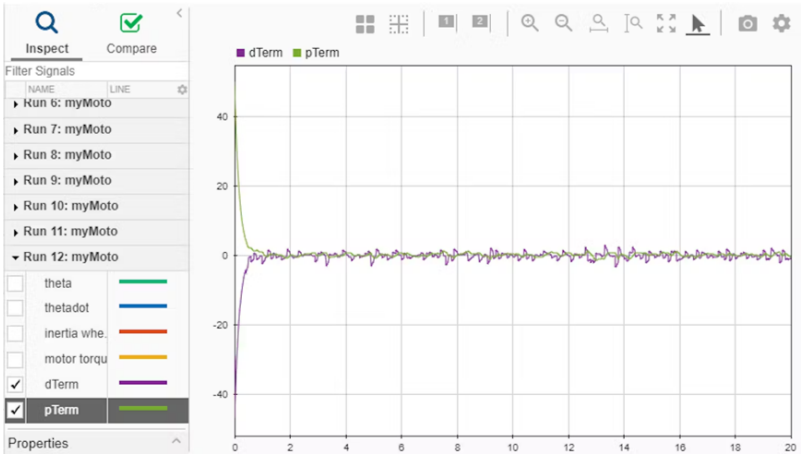

Figure 8: SDI diagram shows $pTerm$ and $pTerm$ with this conditions: Kp = 5e-1, Kd = 8e-2 and *External Torque* gain value set to 1e-2

They are both contributing of the motor torque because they are both providing comparable effort because of the initial non-zero lean angle.
As of now, it has been simulated the motorcycle and PD controller while adding a normal distribution of external torque noise to the system. This work has been implemented in a simulation of a 3D model that responds to behaviors determined by Simulink blocks, but in later chapter it will apply to a real physical motorcycle.

3 Program motorcycle components

The hardware components of motorcycle can be controlled with a Simulink models in particolar we focus on just a few components: the DC motor controlling the inertia wheel, the tachometer sensor and the Inertial Measurement Unit (IMU). also it has been implemented a self made PD controller and it is tested with Simulink.

3.1 Inertia Wheel DC Motor

Now, starting from a Simulink file provided by Matlab called *iWheel0_start*. From this file we added a DC Motor for inertia wheel that is used to apply a torque to the last one, which then causes the motorcycle to exert an equal and opposite counter-torque. In the control algorithm we will determine the necessary motor torque in terms of signed fractional duty cycle (-1 to 1). We have added a tachometer to the model to measure the angular speed of the inertia wheel. The *M3M4DCMotors* block has been taken from the library "Simulink

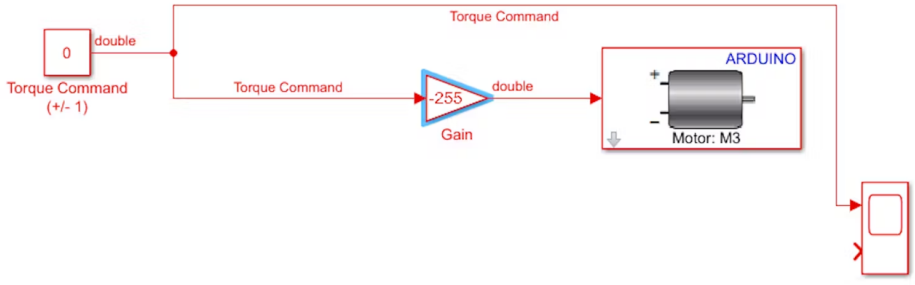

Figure 9: Simulink model for inertia wheel DC motor

Support for Arduino MKR Motor Carrier" and it is a block with one input that indicates both the voltage to be applied across the motor leads and the direction in which to apply it. A positive input between 0 and 255 indicates that a positive torque should be applied to the motor, while a negative input from -255 to 0 indicates a negative torque. The input value gets normalized, rounded, and transmitted to the appropriate PWM channel for positive or negative motor torque as an 8-bit unsigned integer between 0 and 255. The controller that we saw in the previous chapter generated a motor torque command between -1 and 1. To simulate the behaviour of the motor is used a different torque commands between -0.5 and 0.5. Now we have two choices for the gains which are 255 or -255, if the gain is positive we can see that the wheel spins in clock-wise sense (seen from behind), otherwise if the gain is negative the wheel spins in counter-clockwise sense but in this case we have to change the wire connection to *-M3*.

3.2 Tachometer

The tachometer measure the inertia wheel speed this is a very important parameter because if the inertia wheel spin very fast for a long time it can permanently damage many components of the motorcycle as the DC Motor, the motor carrier, and the structural members for the vibration. The goals is to spin the inertia wheel fast enough to keep the motorcycle balanced but not so fast to end up damaging it. The Hall-effect sensor's digital circuit contains logic to keep track of how many magnetic field pulses have taken place since power was last reset. We can use this information to determine the approximate angular speed of the inertia wheel over some time interval. Since the tachometer executes at 10 Hz (the period of 0.1 in sampling rate corresponds to 10 Hz, and the rest of the model at 100 Hz (0.01 sampling rate), it is needed a *Rate Transition* block to hold the tachometer output for 10 controller execution cycles. To do this, is added a *Rate Transition* block. the output is then connected to the model. The inertia wheel doesn't accelerate indefinitely when torque is applied because there is friction. Most restorative forces, including solid-solid friction, have a term that increases with the relative speed of the objects. As the inertia wheel speed increases, the frictional torque in the motor also increases until a terminal angular speed is reached. Here the frictional torque is equal and opposite to the applied motor torque. Thus, it isn't possible to reach infinite angular speed. However, when the frictional force

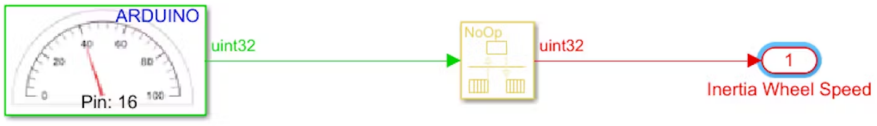

Figure 10: Simulink model for tachometer

is high, lots of heat is generated by the motor and could lead to irreversible damage. As we can see from the

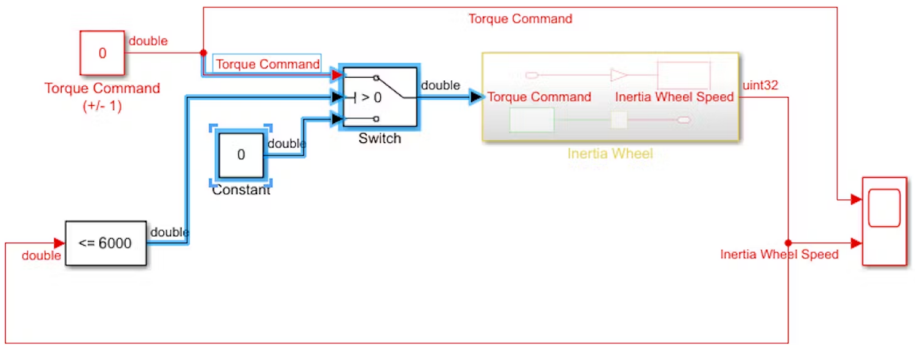

Figure 11: Simulink model for inertia wheel with switch

picture 11, it has been added a limit that if it is exceeded the switch pulls the 0 constant, instead it takes the torque command. If the speed threshold is exceeded we can observe that the motor doesn't get switched off. We set the torque command to 0. Instead of shutting off the motor permanently, it switches off each moment the inertia wheel speed exceeds the threshold and turns back on. By "permanently," we mean until the application is restarted. This way you can give the DC motor and motor carrier some time to cool down before attempting to drive it again. To do this, we will add a latch to the output signal from the *Compare To Constant* block. You want the *Switch* block's control signal to be 1 if a high inertia wheel speed has never occurred and 0 if a high inertia wheel speed has occurred even once.

Moreover it is created a new subsystem called *Get iWheel Saturation*.

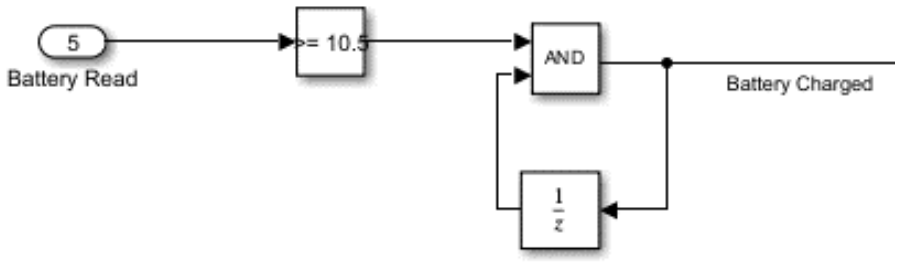

Figure 12: Simulink model of control on battery status

3.3 Inertial Measurement Sensor (IMU) - BNO055

We have also a inertia sensor that measures the lean angle θ and its derivative in time $\dot{\theta}$. On the motorbike we have an absolute orientation sensor that detect the gravity vector and the magnetic field and from these quantities it is possible to define the rotational and the linear motion.
To test the IMU we open the file *IMU0_start*.
The block of the sensor is available inside the library Simulink Support for Arduino Sensors, we have taken and configured it.
Depending on the configuration the block will send its absolute orientation with respect to the magnetic field and the gravitational vector and also its angular velocity along the coordinates of the IMU.
The protocol that the IMU uses to comunicate with the MKR1000 is the I2C.
Another option to set inside the block is the sample time and in particular in our case it is set at 0.01 seconds.
For the IMU to give accurate outputs, each sensor needs to be initialized to locate the gravity vector and

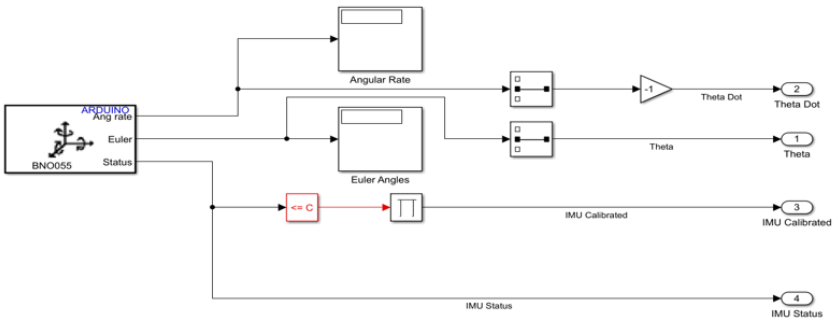

Figure 13: Scheme within *IMU* subsystem block

magnetic field and offset the coordinates accordingly. The elements of the calibration status vector respectively represent:

1. IMU system
2. Gyroscope
3. Accelerometer
4. Magnetometer

Each calibration status vector element has a value of 0 to 3, indicating the degree to which the sensor is calibrated. For the motorcycle, you need the gyroscope and the magnetometer to be fully calibrated (3). To isolate the rotational measurements of the motorcycle, it has been used the selector block that in this case we take just the second value of the array. To calibrate the gyroscope we rotate, as we can see from the picture, the motorcycle at least 90 degrees along each of the 3 spatial axes. This calibration process must be done every time the sensor is powered on. Given the outputs of the sensor we manipulate them to get the information we need for balance control. We can define the minimum calibration status as a 4x1 vector: [0;3;0;3], which, compared to the vector signal coming out from the sensor, would give to us a much simplier view about the correct calibration. Rather than seeing whether each of the sensor components are sufficiently calibrated, we want to know whether all the sensor components are sufficiently calibrated. To isolate the rotational measurements, we have used a selector block, in particular we are interested only in the second parameter of the matrix 3x1 and also we have added a negative gain because we want the opposite value in output. At the and it is created a submodel that contains the previous system created.

Figure 14: Rotations along each of the 3 spatial axes

4 Closed-loop control (balance in place)

In this chapter, it will combine the controller that we designed in the chapter 2 and the component models we completed in chapter 3 to implement a complete balance control application that will run on the MKR1000 board. In this chapter we will test out the algorithm on the motorcycle and will refine it to account for model imprecision and hardware-specific concerns. By the end of this chapter, we will have a motorcycle that balances in place.
In this chapter, the following topics will be addressed:

- it will be interfaced the hardware sensors and motors with the balance control algorithm;
- it will be implemented safety features to ensure that the motorcycle operates with minimal stress to hardware components;
- it will be monitored the motorcycle's physical response to the control algorithm;
- it will be tuned the PID algorithm and gains such that the motorcycle balances in place.

So, we have integrated the inertia wheel motor, the IMU sensor, and the tachometer into the main system model so that they correctly interface with the PID control algorithm. We then added safety features to minimize the risk of damaging the hardware. Finally, we tested the balance control algorithm on the real hardware and fine-tuned the control algorithm and its PID gains to achieve balance in place. The complete simulink schematic is shown below:
At the first, we started from the Simulink models developed in the previous chapters and in particular we integrated the *IMU* block, developed in the chapter 3, and the *Controller* block, developed in the chapter 2, and so we have modified them appropriately.
We read hardware specifications of the appropriate inertia wheel motor. Since the motor is 12V we accessed in the *Controller* subsystem and edited the *Gain* block just be just before the outport to a gain value of 2e-2. The *Kp* and *Kd* have been divided by 2 to account for this increase in Gain value, so that we have the same controller that worked on the simulation model.
The *theta* and *thetadot* signals of the *IMU* subsystem is connected as inputs to the *Controller* subsystem and these signals are visualized through *Scope* block.
Also the *torque command* signal of the *Controller* block is visualized and so it is connected to *Scope* block.
We have noticed a few characteristic behaviors of the controller algorithm. First, when the motorcycle is held at a constant non-zero lean angle, the motor is generally commanded to generate torque in the same direction. This is consistent with what you should expect from the proportional term of the PD controller.
In order to better understand the relationship between the sensor information and the way the PD controller responds, we monitored the P and D terms individually, in particular the *pTerm* and *dTerm* signals are plotted.

In the figure 16, P is *PD terms: 1* and D is *PD terms: 2*. You can observe that the proportional (P) term has the same shape as theta (but a different amplitude) and the differential term is smaller in magnitude compared

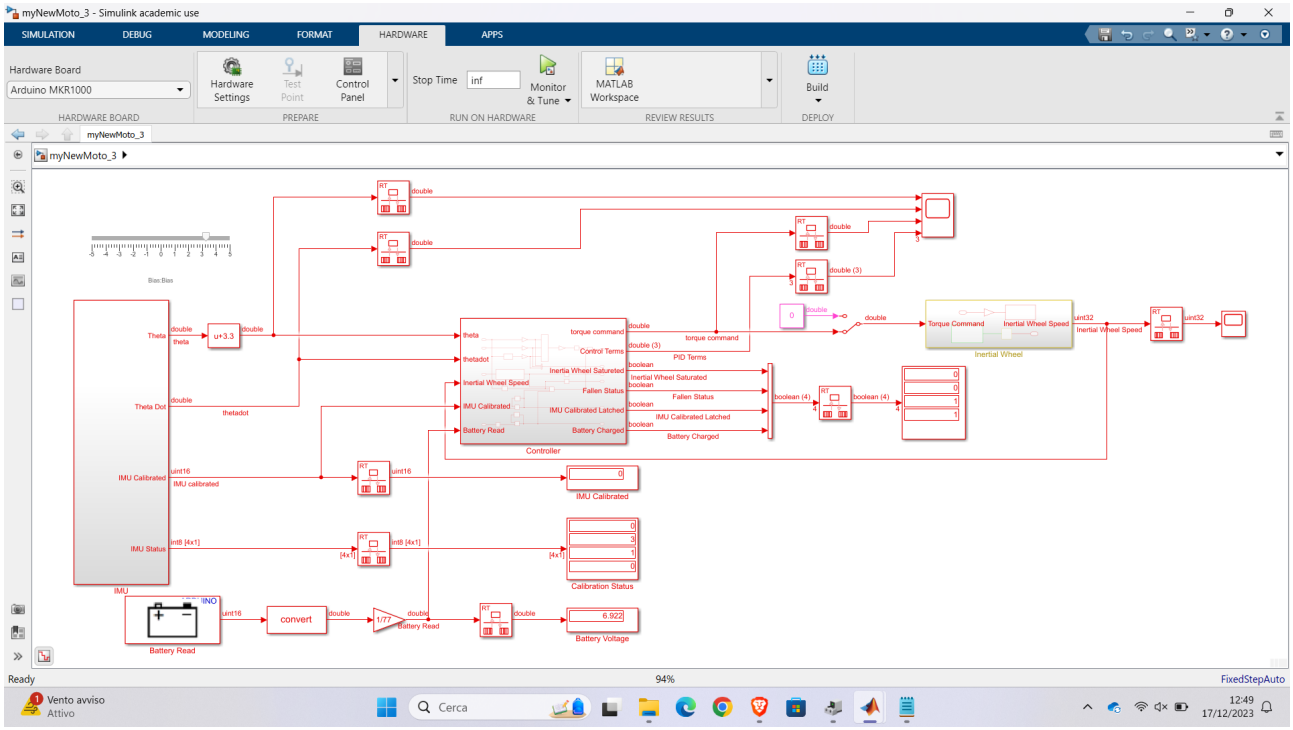

Figure 15: Completed Simulink model to allow the motorcycle to stay balanced in place.

to P term. This is a good rule of thumb, the P term should generally dominate the torque command, while the I and D terms merely augment the proportional response (in this case we have just the D term).
Then we inserted a *Saturation* block to the *Controller* subsystem placing immediately before the *Torque Command* outport block and set the saturation limits to 0.7 and -0.7.

You generally want the torque command to saturate very rarely because it can damage the motor and surrounding hardware if it persists for too long. It is also not good for balance since saturation makes it impossible for the motor torque to respond to variations in θ and $\dot{\theta}$.
In the figure 15 you can see *Inertia Wheel* subsystem that is the copy of *myIWheel.slx* model. Between torque command and *Inertia Wheel* subsystem we inserted a *Manual Switch* block. We added this switch to the model so that you can enable or disable the inertia wheel motor at any time from Simulink and this is useful if you want to calibrate the IMU before attempting to balance the motorcycle or for safety reason i.e. if the inertia wheel picks up too much speed. The inputs of this *Manual Switch* block are torque command and a *Constant* block which the constant value is set to 0.

4.1 First simulations

At this point, we have carried out simulations but we want just tested that the inertia wheel responded to the lean angle direction so we didn't necessarily expect the motorcycle to balance stably.
In order to make these simulations we followed some steps:

- we set the *Gain* inside the *Controller* to 2e-2 and the *PD* gains as $Kp = 25$ $Kd = 4$;
- we ran the model with the inertia wheel disabled (manual switch toward 0);
- we calibrated the IMU, so we spinned the motorcycle along all three principal axes;
- we held the motorcycle close to the vertical plane so that we saw the theta signal close to zero in the *Scope* opened when the application started running;
- we adjusted the lean angle until the torque command signal approaches zero as well and then we enabled the inertia wheel motor while holding the motorcycle using the manual switch;
- we tried leaning the motorcycle a few degrees left and right. The inertia wheel should accelerated in the same direction as the motorcycle is leaning;

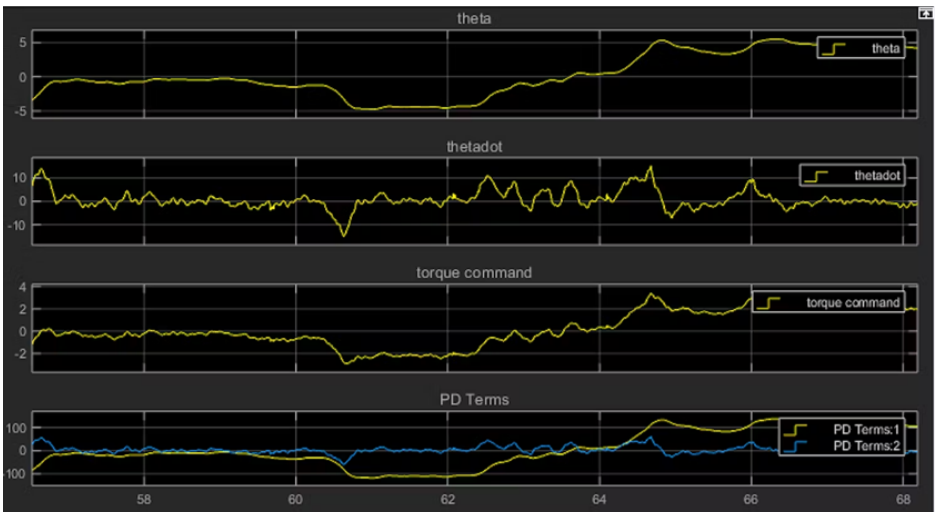

Figure 16: Theta, thetadot, torque command and P and D terms plots

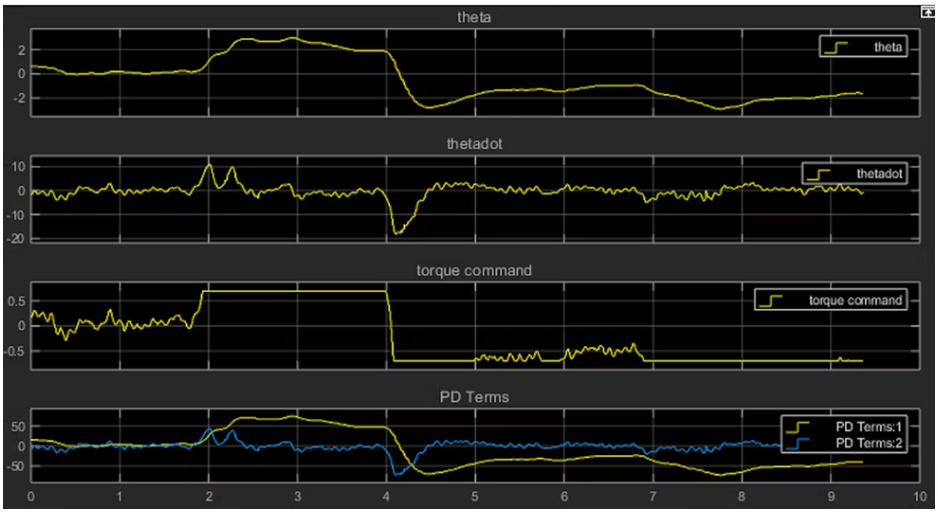

Figure 17: Theta, thetadot, torque command and P and D terms plots with saturation on the torque command

- we tried releasing the motorcycle at $\theta = 0$ and we saw if we get any degree of balance;
- when we finished, we disabled the inertia wheel motor with the manual switch, and stop the model.

Regardless of whether you achieve marginal balance, we observed that the inertia wheel accelerates in the same direction as the lean angle. With the drive gain set low, we saw that the torque command rarely reached the saturation limits at 0.7 and -0.7 when the lean angle is close to zero. When the lean angle was a high value (greater than +/-3 degrees), the torque command was at +/- 0.7 and still the motorcycle could not reverse its direction. When oscillations occured about $\theta = 0$, noticed the similar oscillatory frequency of the P and D terms. A useful way to describe a PID control is as follows: the proportional term is correcting for the error being observed right now, the integral term is correcting for the errors of the recent past, and the derivative term is correcting for errors that are projected to occur in the near future. This explains why D term peaks occured before those of the proportional term.
When we tested the system model on the MKR1000 and the motorcycle hardware we frequently felt the temperature of the inertia wheel motor to make sure it was not overheating because, as described earlier, any motor can become permanently damaged if it gets too hot for too long. Additionally, the Motor Carrier can become damaged from having so much current drawn in a short period of time. While the Motor Carrier comes with firmware that automatically shuts down the motor leads under certain dangerous conditions, you should not rely on that completely to safeguard the Motor Carrier. The safety was part of our algorithm!
In the following section, it will described some safety mechanisms that we added to the controller algorithm that make it very unlikely (but still possible) that hardware damage will occur.

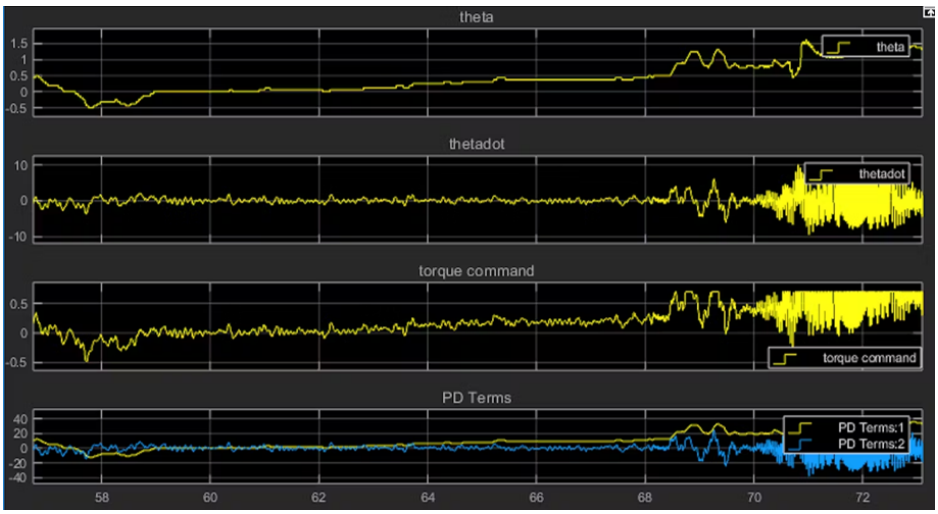

Figure 18: Theta, thetadot, torque command and P and D terms plots in simulation

4.2 Adding failsafe features

At this point we have a controller that accelerates the inertia wheel in the correct direction according to the sign of the lean angle. Moreover we have also a manual switch to the inertia wheel in ordr to shut it off before and after balancing and when abnormal conditions arise, such as the inertia wheel spinning too fast or the motorcycle falling over. To make the controller robust and autonomous, we added some safety features to the controller algorithm to shut off the inertia wheel automatically under the following conditions:

- inertia wheel spins too fast (risk of damaging motor);
- motorcycle "falls" outside a range of "normal" lean angles (risk of damaging inertia wheel or ground surface);
- IMU is not calibrated (risk of destabilizing controller due to inaccurate set point);
- battery is undercharged.

Inertia wheel spinning too fast, so we added a mechanism to automatically shut off the inertia wheel motor if it spins too fast. We used the tachometer to measure the inertia wheel's angular speed and implemented some logic to shut off the inertia wheel motor when a speed threshold is exceeded (as shown in Fig. 19.

The output of the *Get iWheel Saturation* subsystem (this block is shown in Fig. 20) is a logical value of 0 or

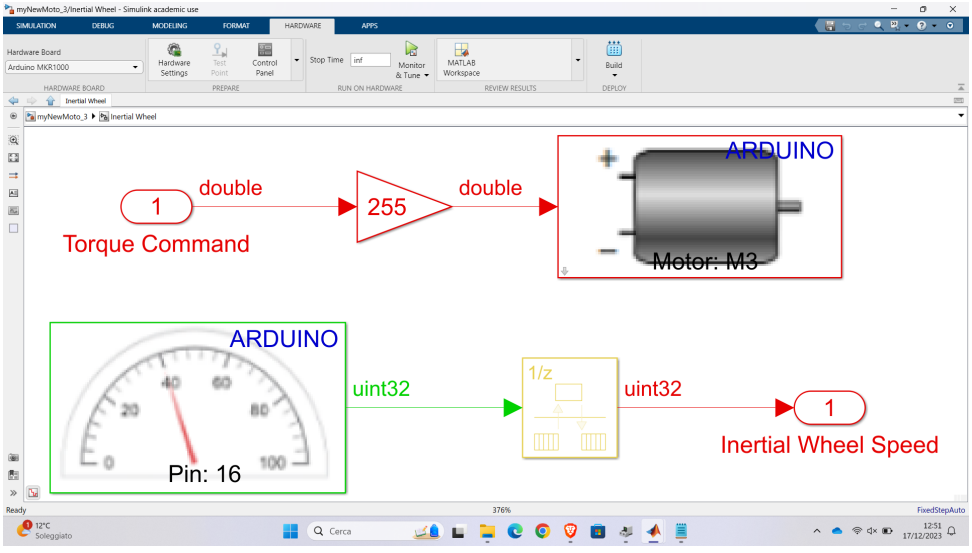

Figure 19: Simulink model of the *Inertial Wheel* block

1. When the value is 1, the inertia wheel motor should operate according to the controller output. When the

value is 0, the inertia wheel motor should stop. In other words, the ultimate torque command should be the product of the controller output and the subsystem output.

When the inertia wheel suddenly stops while the application is running on the hardware, it is useful to know

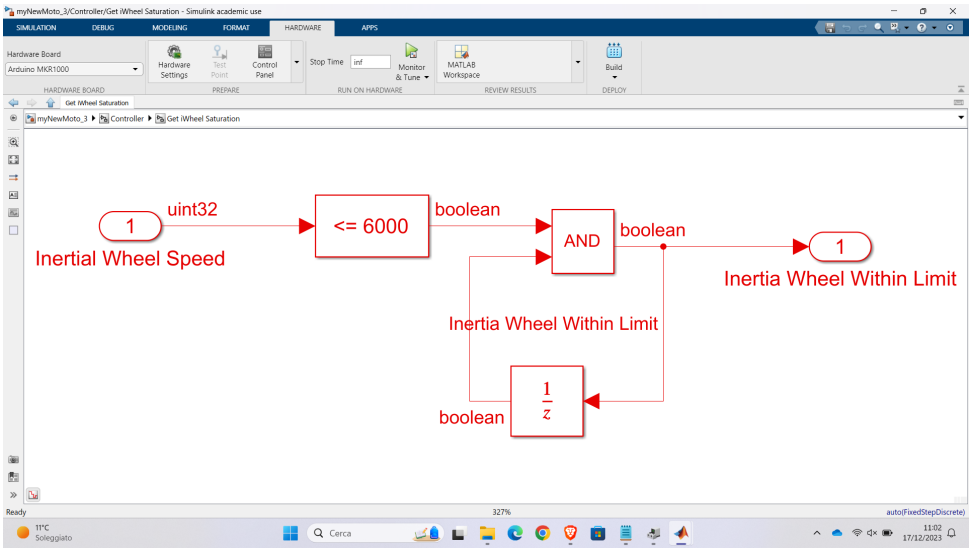

Figure 20: Simulink model of the *Get iWheel Saturation* block

the reason.

Therefore, you should output the inertia wheel saturation status to a *Display* block.

The value will be 1 when the inertia wheel is spinning, and 0 if it has saturated. The outport label is *Inertia Wheel Saturated*, which has the opposite implication. Therefore, you should negate this logical value before it reaches the *Inertia Wheel Saturated* output block.

We have considered the motorcycle to be "fallen" when the magnitude of the lean angle exceeds 6 degrees and therefore turn off the inertia wheel.

The output of the *Interval Test* block is 1 when the lean angle is in an acceptable range and 0 when it is not.

For the next failsafe measure, you want to make sure that the inertia wheel motor does not drive unless the IMU has been calibrated.

The IMU sometimes loses calibration to some degree for one or more sensors after it has initially been calibrated. We want to consider the IMU to be calibrated if it has met the calibration thresholds at some time in the past within the current power session. Therefore, we want to latch the *IMU Calibrated* signal to a value of 1 as soon as the calibration thresholds are met for the first time and leave the signal at that value indefinitely.

We ran the model and enabled the inertia wheel motor using the manual switch. We have ensured that the inertia wheel motor didn't begin to drive until the *IMU Calibrated* signal hits 1 for the first time. Then, we have check the failure status displays to ensure that the *IMU Calibrated Latched* signal remains at 1 permanently once it latches. At the end we have ensurred that the other two failsafe mechanisms (inertia wheel speed cutoff and "fall" sensing) still work as before. *IMU* block is shown in Fig. 21.

Now, let's add the final failsafe feature: shutting off the inertia wheel when the battery level is too low. Operating the inertia wheel with insufficient battery charge can damage the motor hardware since it will not be able to supply enough torque to correct the lean angle in the way it was designed. While the controller will automatically account for environmental variations, including slightly decreased motor torque, a significantly low battery level is outside the range at which the controller is stable.

Notice that the data type of the battery measurement is a 32-bit unsigned integer (uint32). To convert the value to a real-world voltage,we need a floating-point data type that can represent non-integers.

If the battery is fully charged, we should have a reading of at least 12 V. This is optimal for the inertia wheel motor to work properly, but a voltage as low as 11 V is acceptable.

A battery at rest (or close to it) has a higher voltage than a battery that is actively driving a motor due to back electromotive force (back EMF), which is a consequence of Lenz's Law. Due to this phenomenon, we should consider an expected drop in voltage, as well as occasional dips and spikes in the voltage, when setting the threshold for the battery charge failsafe feature. For the purposes of this project, the lowest safe voltage for any portion of the application's life span will be 10.5 V.

The output of the *Compare To Constant* block indicates whether the battery is sufficiently charged at each moment in time. However, we wanted the **Battery Charged** signal to go to zero permanently after the first instance of a low battery detection even if the charge goes back above the voltage threshold.

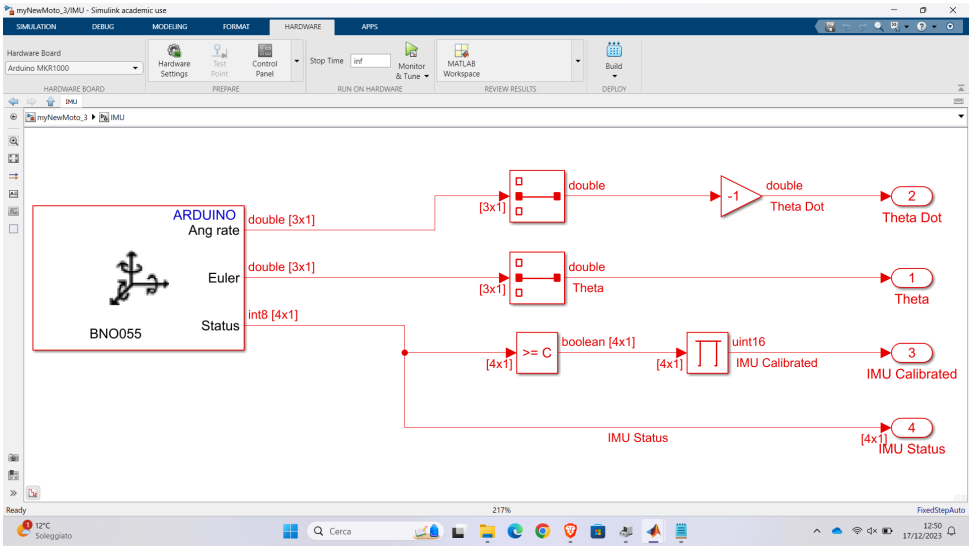

Figure 21: Simulink model of the *IMU* block

Therefore, we wanted to latch off this signal so that the first instance of low voltage brings the *Battery Charged* signal permanently to 0. When this occurs, we removed the battery from the motor carrier and charge it until full.

4.3 Testing and refining the controller algorithm

Next, let's attempt to balance the motorcycle using the inertia wheel. Initially, we needed to shift the center of mass of the motorcycle so that the center of mass point is directly above the wheel-ground axis when the IMU measures θ to be zero. We can do this, for example, by shifting the position of the power supply on the rear of the motorcycle.
If the center of balance has a negative bias, it is possible shifting the power supply to the right and repeating the test. If the center of balance has a positive bias, it is possible shifting the battery to the left and repeating the test. Keep repeating until the center of balance is within 1 degree of 0.
Set the center of balance:
Sometimes sensors can have what we call Bias. In the case of IMU sensor, it could be because of the way it is mounted or that there is an external force on the motorcycle such as the USB cable pushing up against it. Let's use the inertia wheel motor to fine tune the center of balance point and eliminate bias.
Adjust the lean angle until the angle at which the motor torque is zero (where it changes direction). Note this lean angle somewhere, this is the bias in our IMU.

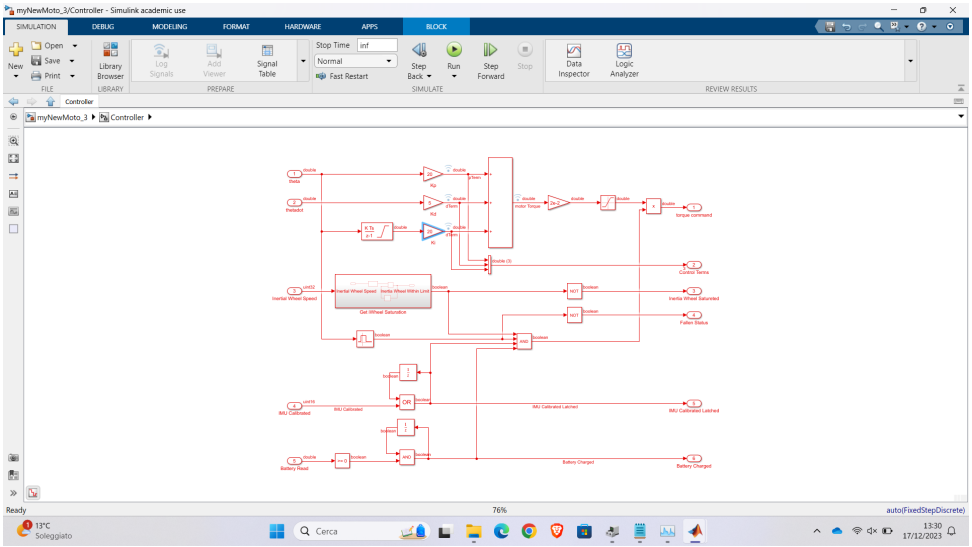

Figure 22: Simulink model of the *Controller* block

4.4 Tune the Controller gains

Now it is time to tune the control gains to find the optimal control law for balancing the motorcycle. There are many approaches to systematic PID controller tuning in the engineering literature. Similarly, there are many software tools available, including *Simulink Control Design*, that attempt to determine the optimal control gains by observing the state errors over time and characterizing the behavior of the system under control. For the initial tuning of the controller, it is helpful to attempt the tuning manually to get intuition for the system behavior and the effect of each control gain. The P and D gains are still set to 25 and 4, respectively, and the Gain is still set to 2e-2.
If the motorcycle is experiencing a steady state error, it is possible to increase Kp to eliminate it, if increasing Kd, Kp needs to be increased as well.

At the end of this point in order to improve the control, a PID controller was implemented (then used in all tests), so the integrator contribution was added with a discrete time integrator. Obviously several tests have been carried out in order to obtain the ideal gains of the controllers, that is $Kp = 20$, $Kd = 5$ and $Ki = 20$.

4.5 Final tests

At the end we carried out a lot of tests in order to balance the motorbike for as long as possible. In this specific

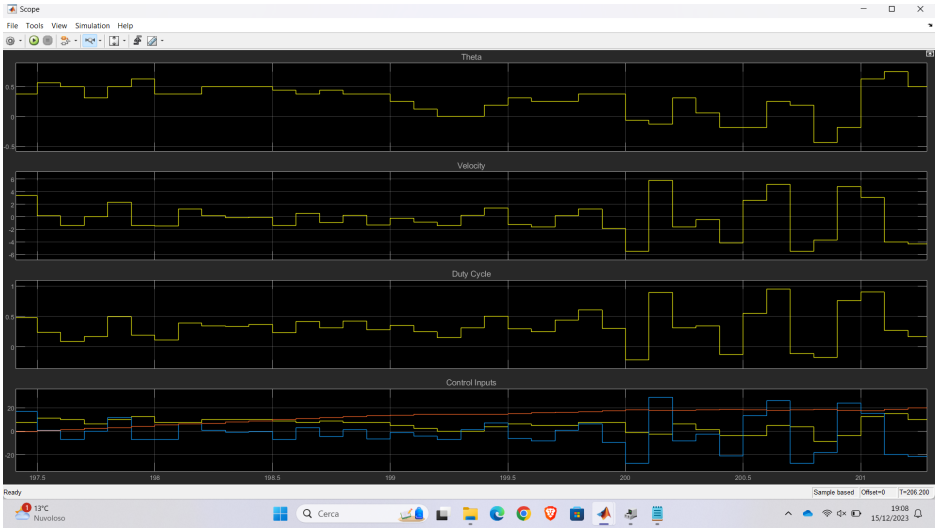

Figure 23: Theta, thetadot, torque command and control inputs plots during the test of the balance of the motorbike

case the motorbike remained in balance for 16 seconds but you can see that the plot shown only part of the test. By carrying out many tests and appropriately modifying the bias we obtained results better than 16 seconds, in particular we obtained the balance of the motorbike for 3 minutes, a record!

5 Balancing with straight motion

The goal of this chapter is to balance the motion of the motorcycle while moving in a straight line either forward or backward. Therefore it is necessary that the MKR1000 shield is disconnected from the PC. Now you can also run the Simulink model on the motorcycle without connection, but there is no communication between the motorcycle and the Simulink model so you can not control the system and not know the system states. So to achieve the goal is implemented a Wi-Fi connection between motorcycle and PC in order to observe wirelessly the system states and control the system from Simulink.
The focus points of this chapter are as follows:

- control and monitor the motorcycle remotely via Wi-Fi;
- acknowledge the capabilities and relevant specifications of the rear wheel DC gear motor and the rotary encoder;
- integrate and test the rear wheel motor to enable the motorcycle's forward and backward motion;
- balance the motorcycle while moving forward, backward, and in place.

The complete Simulink scheme is shown in the following figure:

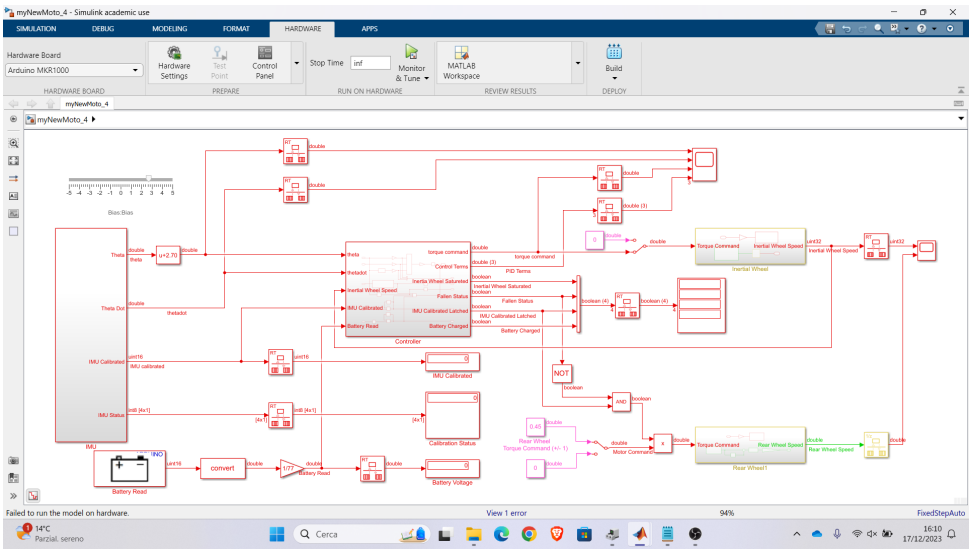

Figure 24: Completed Simulink model to allow the motorcycle to stay balanced in place and moving in a straight line.

5.1 Controlling and monitoring motorcycle via Wi-Fi

So first of all the connection via Wi-Fi is established between PC and MKR1000 shield. This requires both devices to be connected to the same Wi-Fi network. The shield is connected to the Wi-Fi defining as communication interface the Wi-Fi to which connect.
Now the connection is established and it has been tested to verify that the motorcycle sends the states to the PC and vice versa you can control the motorcycle from Simulink.

5.2 Driving rear wheel

Then the second focus point, drive the motor of the rear wheel, is put into practice and in the following steps it will be integrated to the control as previously done for the inertial wheel. To test the rear wheel it is necessary to place the stand of the motorcycle in vertical position in order to elevate the wheel and prevent the motorcycle from walking. In this regard, a first Simulink model has been implemented in which a manual switch enables and disables the rear wheel and the subsystem used previously to control the rear wheel is imported, in fact, just change the motor port to which the rear wheel is connected and eliminate the tachometer, because the measurements of this wheel are made by the encoder. A first test was carried out with speed values included in the range +/- 0.5, for security reasons, both through the connection via USB and Wi-Fi, in particular we tried to drive the motorcycle with boot speed of 0.2. The various balancing and driving tests were carried out by placing the stand in a diagonal position, in order to ensure freedom of movement to the motorbike.

5.3 Measuring motorcycle speed

To finish the second focus point the rotary encoder device attached to the rear wheel motor is used to measure the longitudinal velocity of the motorcycle. Recall from the guide in Chapter 2 that a magnetic rotary encoder detects the rotational motion of a motor shaft by measuring the field of a magnet that is rotating with the motor shaft. The motor carrier keeps track of the number of times the magnetic field has risen and fallen, as seen from two different rotational angles, and converts this information into an integer value. This integer value represents the angular displacement of the motor shaft since the counter was reset.
The encoder has been placed in the subsystem where the speed is given to the engine. The corresponding port and the sample time at 0.01 is assigned to the encoder block. Once tested it is seen that it does not return the speed yet because this encoder has 12 counts per revolution of the motor shaft. The DC gear motor attached to the rear wheel spins the output shaft at 1/100 the rate of the motor shaft. This means that for each forward revolution of the rear wheel, we expect to see the encoder count increase by 12 x 100, or 1200 counts. Therefore to get the speed of the motorcycle in m/s you need to multiply the speed in counts/s by 0.2827 m/1200 counts. So the Gain value is set to "(1/Ts)*(0.2827/1200)". From the tests of the encode it has been noticed that the speed of the motorcycle decreases for a given motor command when the wheel is pushing the weight of the motorcycle.

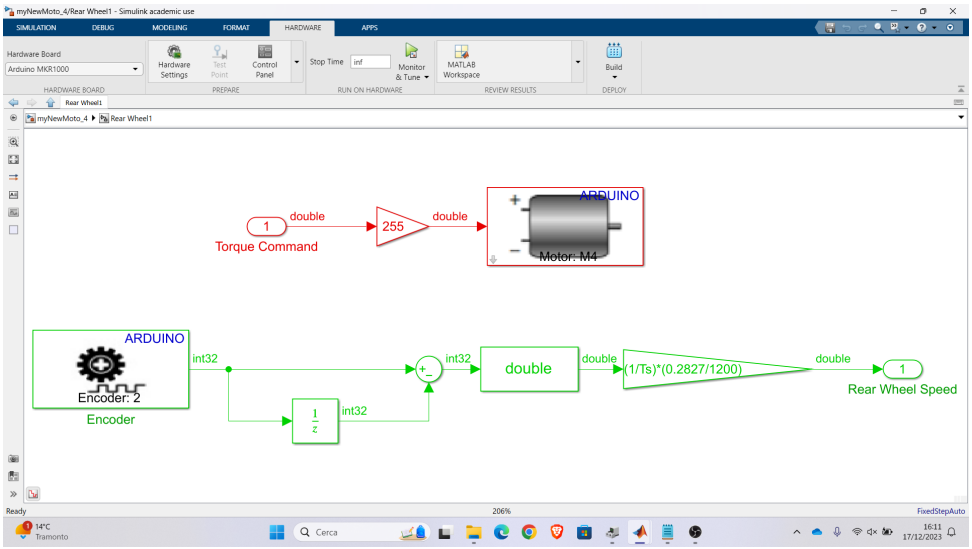

Figure 25: Simulink model of the *Rear Wheel* block.

5.4 Balancing while driving in a straight line

At this point, all that remains is to test the motorcycle in the balance on the place and with the motion forward and backward. So you have to integrate the Simulink model made so far, that concerns the rear wheel, in the model of the complete system.
Remember that the various tests must be performed with the stand in a diagonal position and at each test the IMU sensor must be calibrated. Then first comes was tested if the rear wheel functions, disabling the inertial wheel. Then the balance of the motorcycle in place and at the end enabling the rear wheel was tested the balance of the motorcycle in motion by changing the control of the rear wheel torque in order to test this balance forward and backward. This last test required several error tests in order to find the right rear wheel torque controls.
As mentioned earlier in this chapter the Wi-Fi communication has been configured in order to perform and monitor the balance of the motorcycle without USB cable. The motor and rear wheel encoder have also been integrated to achieve the objectives of this point. Then the balancing on the place and in motion was tested.
In the current model, a PWM command are send to the rear wheel motor to exert some known amount of torque on the rear wheel to propel the motorcycle forward or backward. But the resulting speed of the motorcycle depends on many factors, such as the slope of the surface it is driving on, the weight of the motorcycle, the remaining charge in the battery and so on. So setting the motor torque to a certain value does not result in the motorcycle moving at a precisely known speed (this was seen by observing the graphs during the various tests).

5.5 Final tests

At the end we carried out a lot of tests in order to balance the motorbike and move it in a straight line for as long as possible. In this specific case the motorbike remained in balance and it moves for about 10 seconds but

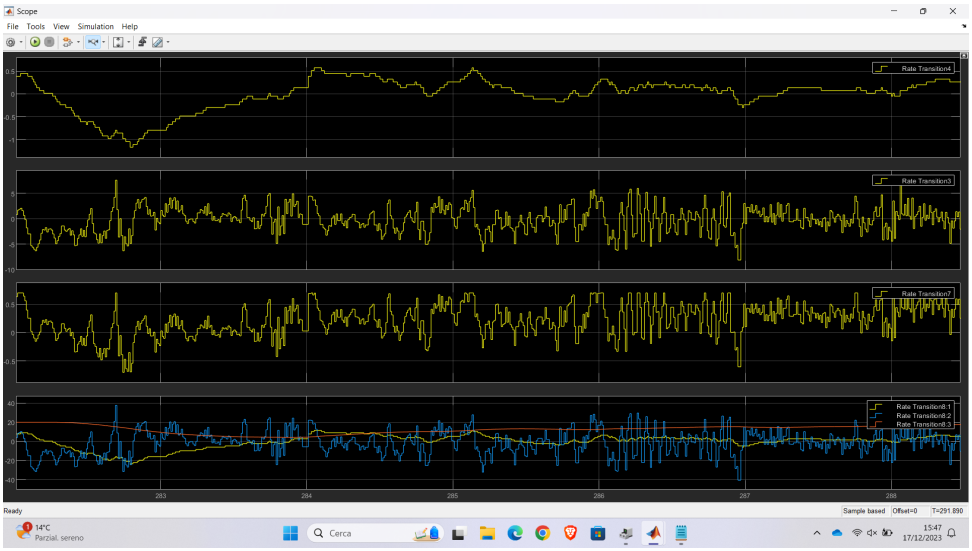

Figure 26: Theta, thetadot, torque command and control inputs plots during the test of the balance of the motorbike while it moves in a straight line.

you can see that the plot shown only part of the test. We carried out many tests appropriately modifying the bias in order to obtain better results.

6 Balancing while steering

The goal of this chapter is to ride the motorcycle along arbitrarily curved paths, but only on turn with a large turn radius. So check if the control algorithm implemented so far and expanded is able to achieve balance while the motorcycle turns.
You will notice that the turning action causes an extra torque component to be applied to the motorcycle around the wheel-ground axis, which then requires the inertia wheel to compensate.
To achieve this goal, the following steps have been taken:

- familiarize with the capabilities and relevant specifications of the servo motor;
- the steer servo motor has been tested and integrated into the motorcycle application to enable it to move along arbitrary paths;
- balance the motorcycle while driving with a large turn radius (small steer angle).

The complete Simulink scheme is shown in the following figure:

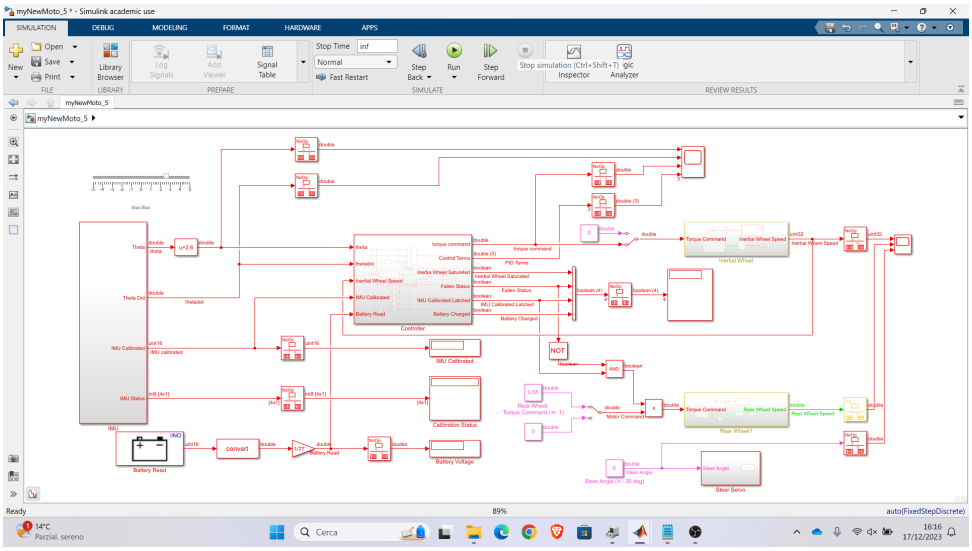

Figure 27: Completed Simulink model to allow the motorcycle to stay balanced while it curves.

6.1 Controlling steer servo

The motorcycle is equipped with a servo motor in the steering column. For its specifications it is able to rotate the lever arm by any angle between 0 and 180 degrees. but you can see that the steering column does not cover the whole range, but an smaller interval than about 60 degrees. Notes the specifications the servo motor has been tested. First through the use of Arduino IDE with which a program, that runs the servo motor 180 degrees continuously, was loaded on the shield. Later it was implemented in Simulink with the use of the *Servo Write* block, to which must be specified the port of the motor to which it is connected.
The servo motor has 3 wires. The brown wire is connected to electrical ground. The red wire connects to the reference voltage, which, in this case, is 5 V. The orange wire is the signal wire and transmits the desired servo angle as a PWM signal. The wires are oriented such that the ground wire (brown) is closest to the rear of the motorcycle and the signal wire (orange) is closest to the front of the motorcycle, as it should be.
A value, between 0 and 180, can be given to the servo motor; but it is noted that there are physical limits due to the mechanical construction of the motorcycle. So several tests were carried out in order to understand the limits and what is the "straight" angle. The angle at which the steering column hits a hard stop on the left (the value at which the servo can no longer turn left) was found; it is 120 degrees and it is the maximum value that can be given to the servo motor. Actually higher values can be given, but the servo motor in order to reach these angles of rotation can damage itself. Then the same tests were carried out for the right curve, after which it was noticed that the maximum angle is 60 degrees and that corresponds to the minimum value to be controlled to the servo, otherwise as previously the servo will attempt to drive through the motorcycle frame.
Finally the "straight" angle was found which is 90 degrees. In order to avoid damage to the motorcycle was introduced a saturation for the steering control between the values seen above. So also the *Bias* block was

introduced, setting it to 90, which allows us to give values between -30 and 30 as input values; in which the value 0 corresponds to the "straight" angle of the servo, positive values give left steer angles and vice versa for negative values. At the end this subsystem has been tested which does not allow to go beyond +/-30 degrees. In order to make tests for the balancing during the steer we launch the Simulink model in Wi-fi mode firstly to balance the motorcycle in straight motion and then we give from the PC in remote a little steer angle (about 5 degrees). The balancing on wide turns may be a little bit troublesome and so to resolve that it is necessary decrease the steer angle or the rear wheel speed. Both actions will decrease the centrifugal force acting on the motorcycle.While turning the inertia wheel must counteract the inertia of the motorcycle.

Note that the motorcycle can maintain itself in equilibrium for a smaller time while doing a large turn but

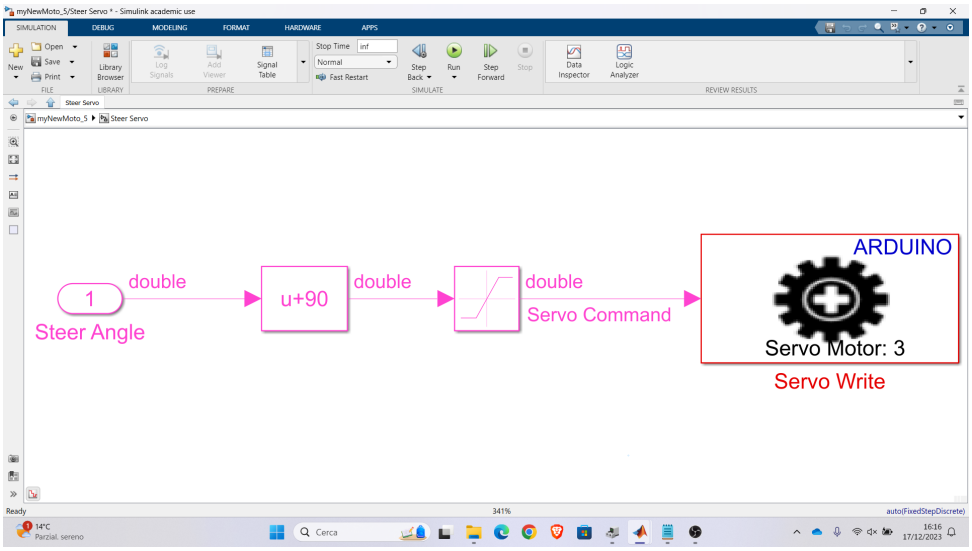

Figure 28: Simulink model of *Steer servo* block.

it cannot continue turning indefinitely because the inertia wheel winds up, leading to a saturation shutdown or the motorcycle simply falling if the inertia wheel cannot generate enough torque to keep it vertical during a sharper turn.

6.2 Final tests

At the end we carried out a lot of tests in order to balance the motorbike while it curves for as long as possible. In this case the motorbike remained in balance and it moves for about 9 seconds but the plot shown only part

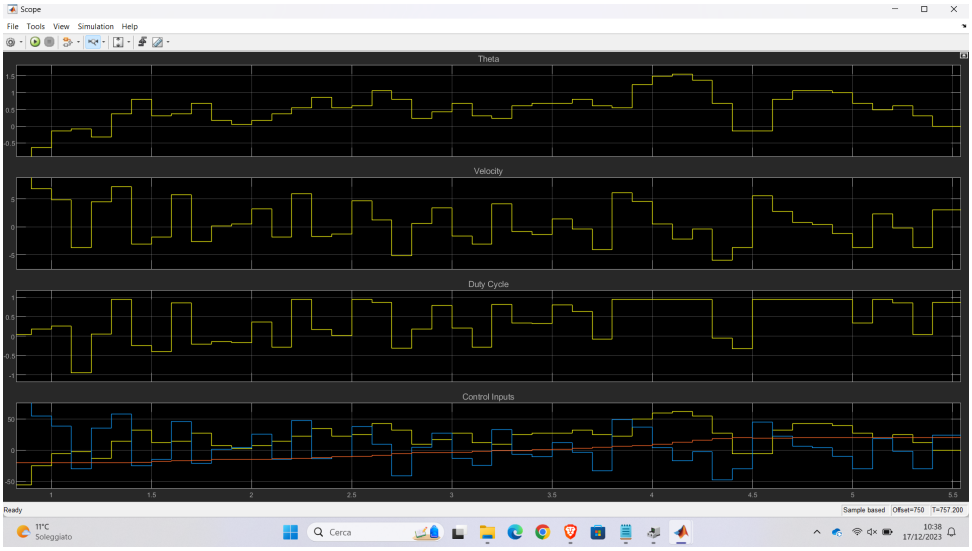

Figure 29: Theta, thetadot, torque command and control inputs plots during the test of the balance of the motorbike while it curves.

of the test. We carried out many tests appropriately modifying the bias in order to obtain better results.