UNIVERSITÀ DEGLI STUDI FIRENZE
DINFO
DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

Corso di Laurea Magistrale in
INGEGNERIA ELETTRICA E DELL'AUTOMAZIONE

Corso di
LABORATORIO DI AUTOMATICA

Control of a LEGO Mindstorms EV3 bike model

Controllo di un modello di motocicletta realizzato su piattaforma LEGO Mindstorms EV3

Report written by:

Nicolò D'Amico
Alessandra Lombardi
Pedro Reis
Daniela Straziati

Supervisor:
Prof. Michele Basso

Academic Year 2015/2016

Contents

List of Figures

Introduction

Since its first appearance in 1996, the LEGO© programmable bricks set, originally developed in collaboration with the MIT Media Laboratory, has received considerable attention as an amazing tool for education. Nowadays, the Mindstorms series has just reached its third release with the new smart EV3 brick, replacing the previous RCX and NXT models. This solution provides multiple vantages such as the great flexibility of bricks architectures, vast choice of sensors and many programming techniques. Recently, Mathworks© has granted its simulation software Simulink with native support for Mindstorms NXT/EV3. The integration of Mindstorms within the Simulink environment now allows one to completely describe the LEGO process through a diagram featuring special blocks, which represent sensors and actuators.
The aim of this project is to create a control system for a self-stabilizing LEGO Mindstorms bike. Initially the mathematical model reported in [1] is adapted to the case at hand, properly modifying the moments of inertia and identifying motors and sensors characteristics parameters . The development of a mathematical model allows to simulate and easily analize the present control system response and thus compare it with others types of controllers. In order to obtain a simulated behaviour similar to the real one of the process, actuators nonlinear dynamics must also be modelled. In a second moment the control system that allowes the bike to self-balance and maintain a straight trajectory is designed. This last part requires two steps, first the designed controller is tested through simulation and then applied to the real process to be validated.

Chapter 1

LEGO bike Mindstorms EV3

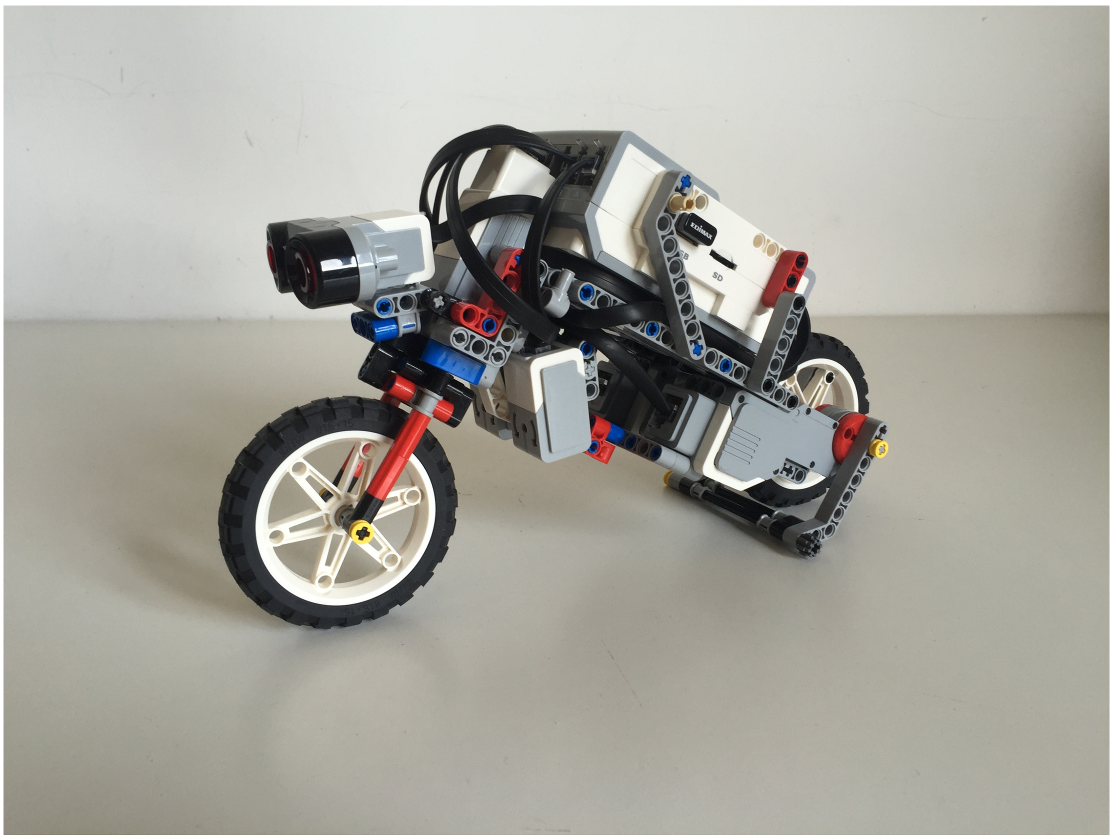

Figure 1.1: The LEGO-bike prototype

The EV3 bike used in this project is depicted in figure (1.1). The EV3 brick, visible in figure (1.2), constitutes nearly a third of the bike total weight and is the robot intelligent unit. This brick contains a small computer and is equipped with a LCD screen and rechargeable lithium battery. The presence of the EV3 brick, together with other factors as negative effects of blocks play, flexibility and elasticity of the plastic bricks and the need to mantain the best simmetry with respect to the

Figure 1.2: EV3 brick

vertical plane, highly influence the bike structure. The resulting bike is a compromise between efficient mechanics and simple multi-body structure and it employs the following electric components: three motors (motion drive, steer and kickstand), one gyro sensor to measure the lean angular velocity and one ultrasonic sensor to detect the obstacles.
The bike is provided with a kickstand, whose motor is mounted opposite to the motion drive motor and it is basically exploited to balance the masses. Nonetheless, it has the ability to keep the bicycle stand-up in a vertical position allowing for an initial calibration of the gyro sensor, resetting its offset and reducing the drift effects on the computation of the bike lean angle. During the calibration the bike must stand still, allowing the noise measurement to be isolated.
Two types of motor are used for the bike:

- Large motor (1.3) (left) as motion drive and kickstand
- Medium motor (1.3) (right) as steering wheel

Figure 1.3: EV3 Large motor (left) and EV3 Medium motor (right)

The encoder inside the motors measures the angular position in degrees with a resolution of 1°.
The gyro sensor (1.4) measures the lean angular velocity in degrees/second with an accuracy of ±3°.
The ultrasonic sensor (1.4) is designed for a wide range of applications including measure the distance, detection, or traffic monitoring. It can measure distances up to $250cm$ with an accuracy of $\pm 1cm$. In our project we used it to stop the EV3 bike when an obstacle is detected.

Figure 1.4: EV3 gyro sensor (left) and EV3 ultrasonic sensor (right)

1.1 Simulink environment

Simulink environment offers a visual way of programming connecting specified blocks from mathematical operations to complex functions (1.5) using lines transporting signals through blocks.

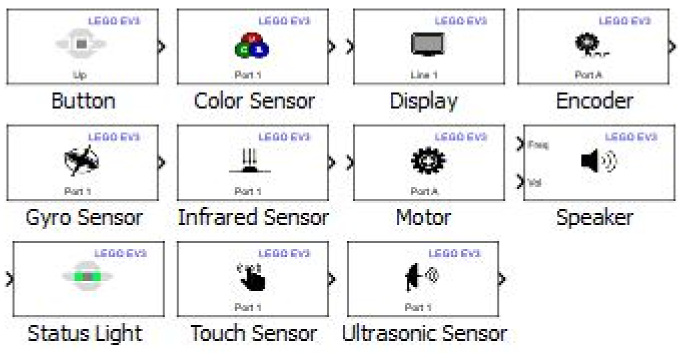

Figure 1.5: Simulink library for LEGO Mindstorms EV3

For a correct operation, some parameters, as the gate on the EV3 where the block is connected and the sample time, can be specified directly in each block. Every Mindstorms block, as motors and sensors, can be used directly to build the control scheme of the robot (1.5).

By using an external Wi-Fi adapter, the control scheme obtained can be uploaded on the EV3 brick. The program can be run in "stand alone mode" or in "external mode", useful to grant a run-time communication between EV3 brick and Simulink, to acquire the experimental data and to modify the system parameters at run-time via WI-FI connection.

1.2 3D Model

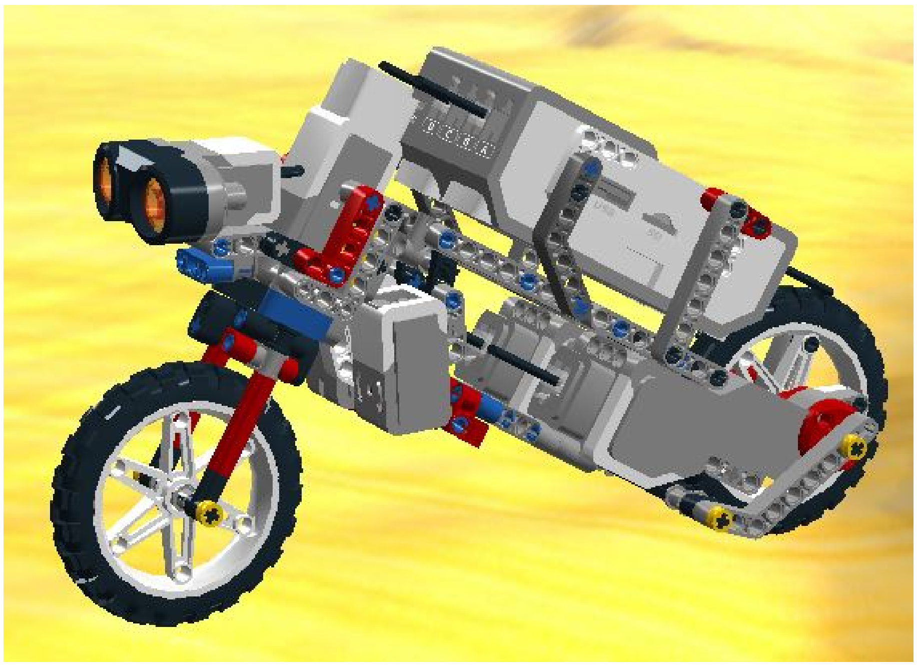

Figure 1.6: 3D model used for virtual reality simulation in the Simulink 3D Animation Toolbox.

To analyze the results and to have a visual comparison during simulation, we created a tridimensional EV3 bike model with "Simulink 3D Animation Toolbox". Using the 3D model we were able to observe the EV3 bike behaviour during simulation. This toolbox allows to interact with the virtual world of EV3 bike and to control its performance. The only format compatible with the trdimensional model is VRML (Virtual Reality Modeling Language). We used three different programs to convert the file and to adapt it to Simulink: "Lego Digital Designer", "LeoCAD" and "Autodesk 3DStudio Max".
We initially created the LEGO model of the bike with the free and easy to use software "Lego Digital Designer", in order to automatically ob-

tained the building instructions of the bike. The result is visible in figure (1.6). From here we exported the file into ".ldr" (LDraw) that allowed us to open it in LeoCAD.
After this we used the program "LeoCAD" and exported the bike model in the format ".3ds" (3D Studio) that could be correctly imported in "Autodesk 3DStudio MAX". Using this latter program we created the blocks with the hierarchy in figure (1.7) in this way if we set an action for the block "bike" the entire model will be enovolved, but if we perform a rotation for the "front frame" it acts also only on the "front wheel". Generally if we set an action for a specified block it will act only on it and its children and doesn't affect the other pieces of the model.
Exporting the file, from "3DStudio Max", in VRML format we make it readable from the Simulink 3D Animation Toolbox.
Finally, using the Matlab integrated sofrware "V-Realm Builder", we created a viewpoint and a background to have a better view of the 3D model.

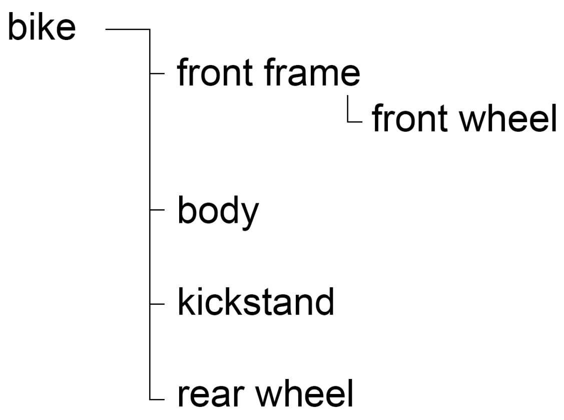

Figure 1.7: Hierarchy of the 3D model

Chapter 2

Model

In this chapter a linearized fourth order mathematical model representing the EV3 bike is described. This multibody model represents a driverless bike divided in four blocks: rear frame (rf), front frame (ff), rear wheel (rw) and front wheel (fw).

2.1 Hypotheses

The model is based on the following hypothesis:

- The bike is divided in four rigid bodies
- The friction between wheels and ground is enough to prevent slipping and slitting
- Only small oscillations around the vertical equilibrium are considered
- Gravity, constraining reactions and steer and roll torques are the only external forces applied to the bike
- The bike is symmetrical with respect to the vertical longitudinale plane when standing in the vertical position of equilibrium
- Velocity is constant

2.2 Coordinate reference system

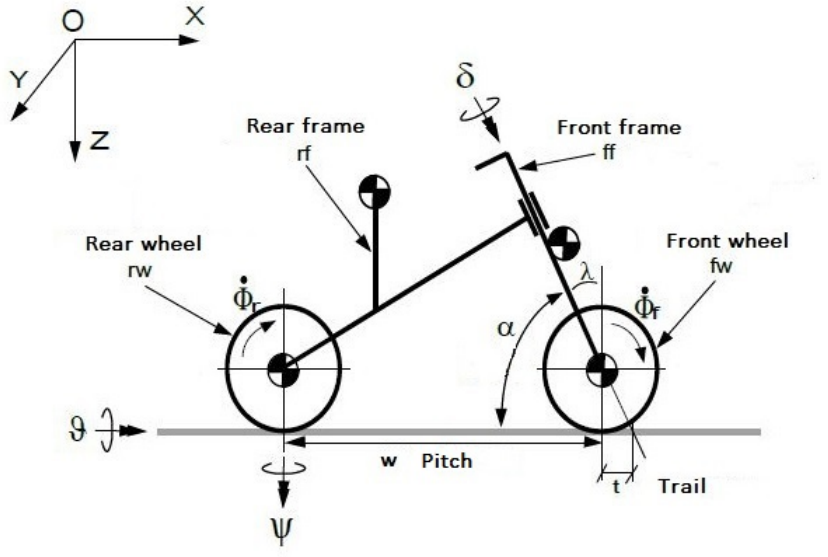

Figure 2.1: Coordinate reference system

In figure (2.1) is showed the coordinates reference system where:

- Ψ is the yaw angle;
- θ is the lean angle from the vertical axis;
- δ is the steer angle around the upright position;
- Φ_r and Φ_f are respectively the angular position of the rear and front wheel;
- λ is the Caster angle.

2.3 Equations of motion

Assuming a constant forward velocity V, the lateral dynamics (balancing, leaning, steering and turning) around the upright position can be described by the second-order Linear Parameter-Varying (LPV) differential equation system:

$$M \begin{bmatrix} \ddot{\theta} \\ \ddot{\delta} \end{bmatrix} + C \begin{bmatrix} \dot{\theta} \\ \dot{\delta} \end{bmatrix} + K \begin{bmatrix} \theta \\ \delta \end{bmatrix} = \begin{bmatrix} T_\theta \\ T_\delta \end{bmatrix} \tag{2.1}$$

where

- T_θ and T_δ are the external moments around the two axes of interest

- M= $\begin{bmatrix} T_{xx} & F_{\lambda x} + \dfrac{c_f}{w} T_{xz} \\ F_{\lambda x} + \dfrac{c_f}{w} T_{xz} & F_{\lambda\lambda} + 2\dfrac{c_f}{w} F_{\lambda z} + \dfrac{c_f^2}{w^2} T_{zz} \end{bmatrix}$

  is the inertial term

- C= $\begin{bmatrix} 0 & H_f cos(\lambda) + \frac{c_f}{w} H_t + V(T_{xz}\frac{cos(\lambda)}{w} - \frac{c_f}{w} m_t h_t) \\ -(H_f cos(\lambda) + \frac{c_f}{w} H_t) & V(\frac{cos(\lambda)}{w}(F_{\lambda z} + \frac{c_f}{w} T_{zz}) + \frac{c_f}{w}\nu) \end{bmatrix}$

  is the damping term

- K= $\begin{bmatrix} g m_t h_t & -g\nu + H_t V \frac{cos(\lambda)}{w} - V^2 \frac{cos(\lambda)}{w} m_t h_t \\ -g\nu & -g\nu sin(\lambda) + V H_f sin(\lambda) \frac{cos(\lambda)}{w} + V^2 \frac{cos(\lambda)}{w} \nu \end{bmatrix}$

  is the stiffness term

In the matrices above:

- $c_f = tcos(\lambda)$;
- H_f is the angular momentum due to wheels rotation;
- H_t is the total angular momentum;
- F is the inertial tensor of the block containig the front frame and the front wheel, computed with respect to the steer axis;
- T is the total inertial tensor of the moto in vertical position;
- $\nu = m_f u + \dfrac{c_f m_t l_t}{w}$;
- $g = 9.81 {}^{m}/_{s^2}$;
- h_t are the coordinates of the bike centre of mass.

The external moment T_θ affecting the lean dynamics in (2.1) accounts only for process noise and/or unmodeled behaviors, while the torque T_δ is provided by the steer servomotor driven by the signal $u \in [-1, 1]$ acting as a stabilizing input and modeled with sufficient accuracy by the equation

$$T_\delta = ku - \beta\dot{\delta}. \tag{2.2}$$

Eventually, by choosing

$$x = \begin{bmatrix} \theta \\ \delta \\ \dot{\theta} \\ \dot{\delta} \end{bmatrix} \tag{2.3}$$

as the state vector, also modelling torque disturbances around the axes θ and δ by the vector signal

$$\xi = \begin{bmatrix} \xi_\theta \\ \xi_\delta \end{bmatrix}, \tag{2.4}$$

and finally denoting sensor noises as the vector ν, we can recast (2.1) and (2.2) into the following LTI state equation form:

$$\begin{cases} \dot{x} = Ax + Bu + G\xi \\ y = Cx + H\nu \end{cases} \tag{2.5}$$

where

- A= $\begin{bmatrix} 0_{2x2} & I_{2x2} \\ -M^{-1}K & -M^{-1}(C + \beta e_2 e_2') \end{bmatrix}$;
- B= $k \begin{bmatrix} 0_{2x1} \\ M^{-1}e_2 \end{bmatrix}$;
- G= $\begin{bmatrix} 0_{2x2} \\ M^{-1} \end{bmatrix}$;

- C=$\begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$.
- H=I_{2x2}.
- $e_2 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$.

2.4 Actuators model

The torque applied to the steer is supplied from the motor using a percent value that is the duty cycle. In order to model the actuators behavior from the input command, represented by the duty cycle, to the output torque, the DC motor equation is used:

$$V_a = R_a i + L_a \frac{d}{dt} i + k_e \omega \tag{2.6}$$

where

- $\omega = \dot{\delta}$;
- V_{max} maximum voltage provided by the battery;
- $V_a = DV_{max}$ relation between duty cycle and voltage;
- L_a is negligible;
- $T = k_t i$ relation between current and torque.

replacing in (2.6)

$$T = \frac{k_t}{R_a}(DV_{max} - k_e \dot{\delta}). \tag{2.7}$$

An experiment has been set up to determine the required missing parameters and to describe the motor model.
Considering the following equation:

$$T_\delta = b\dot{\delta} + I\ddot{\delta} = D(\frac{k_t}{R_a} V_{max}) = Dn \tag{2.8}$$

where I is the load and motor total inertia and b is the damping coefficient.
To estimate the missing coefficients, necessary for motor identification, we followed the ensuing procedure: initially we calculated the ratio value $S = \frac{n}{b}$ as slope of the line obtained by fitting the results of the following experiments. In theese experiments, considering angular acceleration equal to zero, set a duty cycle value, we revealed the corresponding angular velocity value. From equations (2.7) and (2.8)

$$Dn = (b + 4\frac{k_t}{R_a}k_e)\dot{\delta} + I\ddot{\delta} \tag{2.9}$$

Studying the step response with the Areas method the transfer function between D and $\dot{\delta}$ was identified:

$$\dot{\delta} = (\frac{k}{1 + \tau s})D \tag{2.10}$$

where

- $k = V_{max}\frac{k_t}{R_a F_a}$;
- $\tau = \frac{I}{F_a}$.

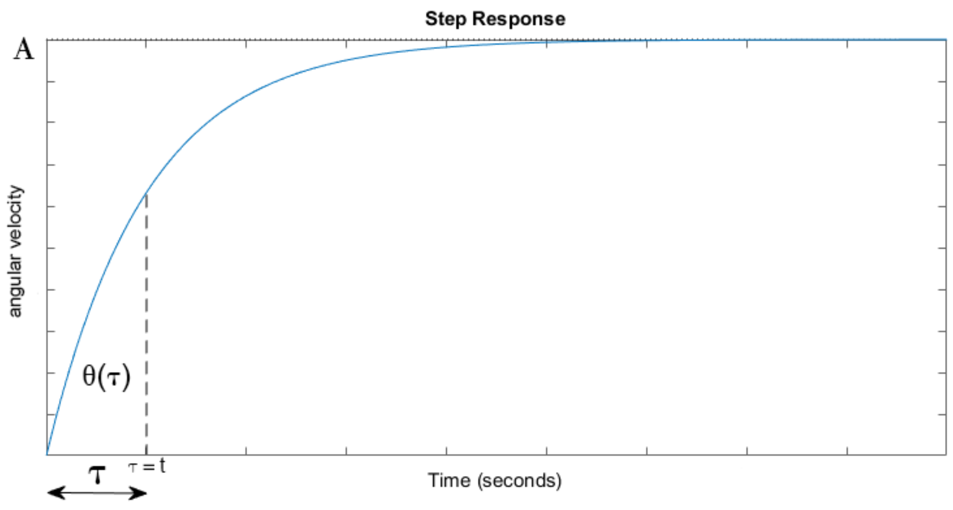

Figure 2.2: Step response

According to this method, τ corresponds to the t value when the ratio $\frac{\theta(t)}{At} = \frac{1}{e}$, where $\theta(t)$ is the area under the step response and At is the total area. The experiment was repeated several times changing the value of duty cycle D. The value of the time constant is obtained as the average of the results is $\tau = 0.0430s$.

D	$A(gradi/s)$	$A(rad/s)$	$tau(s)$
40	750	13.0900	0.0410
50	890	15.5334	0.0485
60	1030	17.9769	0.0390
70	1170	20.4204	0.0433

Table 2.1: τ values.

The obtained data are not enough to identify the parameters of the motor. Due to this we remade the experiment using an inertial load much higher I_c than the wheel in order to neglect the motor inertia. As inertial load we used a thin metallic bar with a lenght of $l_c = 0.62m$, a mass of $m_c = 0.237kg$ and an inertia of $I_c = 0.0076kgm^2$.

D	$A(gradi/s)$	$A(rad/s)$	$tau_c(s)$
40	625	10.9083	1.1300
50	740	12.9154	1.1900
60	850	14.8353	1.1200
70	950	16.5806	1.1000

Table 2.2: τ_c values.

Obtained $\tau_c = 1.1350s$, it's possible to calculate the next coefficients

- $F_a = \frac{I_c}{\tau_c}$;
- $I = \tau F_a$ is the motor inertia;
- $b = \frac{I}{\tau}$ (constant velocity);
- $k = Sb$;
- $\frac{k_t}{R_a} = \frac{k}{V_{max}} F_a$.

2.5 Inertia Matrices

In order to calculate the parameters of matrices A and B, it was necessary to determine the inertia of the four blocks that constitute the model: front frame, back frame, front wheel and back wheel, figure (2.3). Due to the complex geometry of this blocks, in order to calculate the inertia we made the approximations represented in figure (2.4).
Each block was weighed using a scale with the resolution of a gram and measured with a calipers. Applying the known formulae to calculate the inertia of a solid object, we obtained the inertia matrices of each block in figure (2.4). These aproximations imply that the mass of each block is uniformly distributed; this hypothesis is validated by experimental result that confirm that the real mass center is close to the one obtained. After that, by the formula of the center of mass and Huygens-Steiner theorem, we achieved the inertia matrix of the whole EV3-bike with respect to the center of mass and the inertia matrix of the front frame and wheel with respect to the steering axis.
Thus we obtained all the data needed to describe the EV3-bike model.

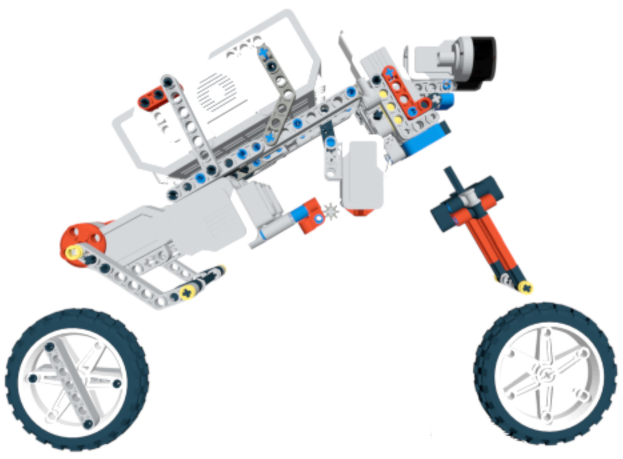

Figure 2.3: Multibody decomposition of the EV3-bike model

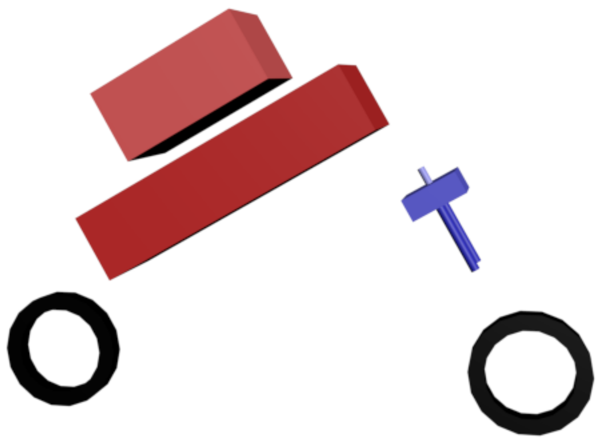

Figure 2.4: Approximation of the four blocks of the EV3-bike model

2.6 Sensors modelling

In this section we model the behaviour of the sensors: the encoder to measure the angular position of the steer and the gyroscope for the lean angular velocity.
The encoder measures with an accuracy of one degree. In order to model this behavior we used a quantizer block. We also needed to convert the signal provided by the sensor from degrees to radians; to this aim we used a gain block of $180/\pi$.
The gyroscope is more complex beacuse, in addition to the quantizer error, it also provides an output signal with a noise of non zero average. This noise is modelled as the sum of white noise and an offset, which parameters are obtained by experimental tests. We also added a gain block in order to convert the angular velocity from degrees per seconds to radians per seconds.

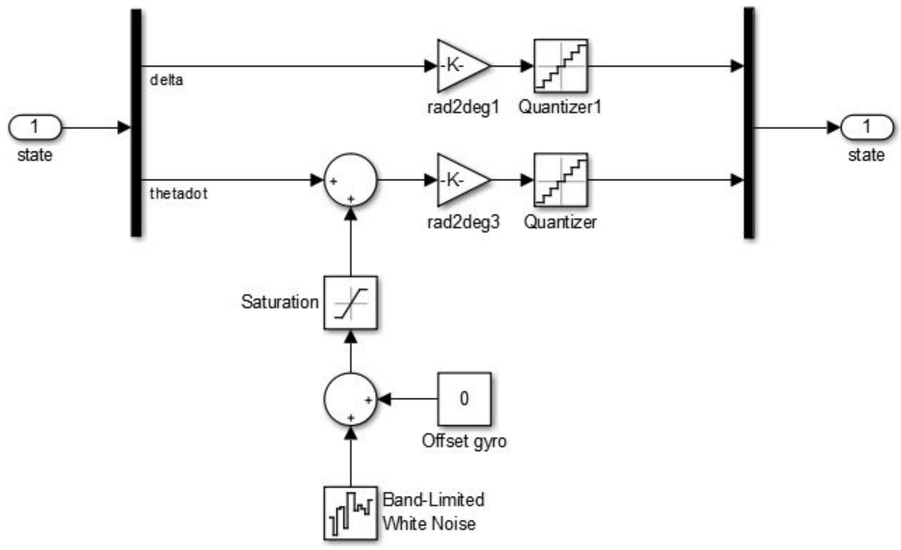

Figure 2.5: Sensors model

Chapter 3

Control system

In a real bike, in order to mantain the equilibrium, the driver can act over the handlebar and also change his position in order to move the center of mass. In the case at hand, the only thing we can act on is the position of the handlebar, that is the steer angle, so the control will just be for this one.
In the followig chapter we analyze the control techniques used to stabilize the EV3 bike. Although a few techniques can be based on the "steer into fall" concept, that consists in steering in the direction the EV3 bike is falling, we design a control algorithm using a model based design approach. First the control system is applyed to the model and then validated with a test on the real EV3 bike.

3.1 Simulation

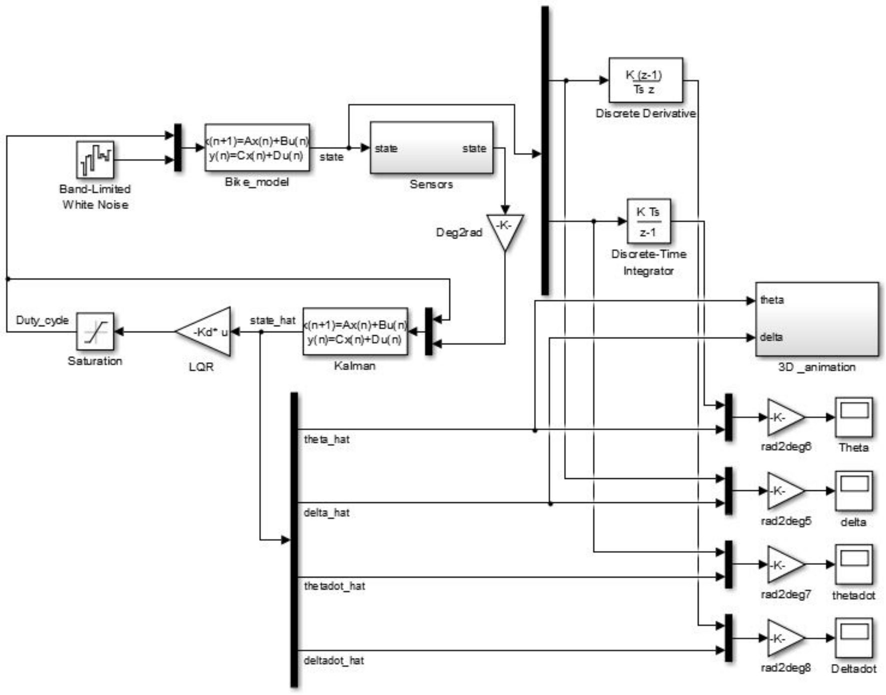

Figure 3.1: Simulink model

In figure (3.1) is depicted the control scheme created in Simulink environment, implementing the motors and sensors model previously described. As controller we used a linear quadratic regulator (LQR) and to reconstruct the state, not entirely accessible by the sensors signals, we used a Kalman filter. In order to improve the real-time view of the simulation we also installed a virtual reality block containing the $3D$ EV3 bike model.

3.1.1 LQR

In the Optimal Control Theory, the Linear Quadratic Regulator (LQR) is a state feedback controller. The control law is:

$$u = -Kx$$

where:

- u is the duty cycle applied to the steering motor

- K is the gain matrix obtained by the Control System Toolbox function
- x is the system state described in the previous chapter

The gain matrix K is obtained using the Control System Toolbox functions with a sample time of $0.01s$ in order to guarantee the data capture from the experiment. This matrix minimizes the following cost functional

$$J = \sum_{k=0}^{\infty} x_k' Q x_k + u_k' R u_k$$

where Q and R are weight matrices based on experience. The Q matrix contains the weight related to each state of the system whereas R matrix have the same dimension of the input, so here is a unit matrix representing the weight related to the input.
To correctly choose the weight we realized several experiments by varying one value each time to understand the effect caused by every single weight. Increasing the importance of the steering angle involve a reduction of the swerve, destabilizing the bike. If we rise the weight on the lean angle, we obtain an efficient control with a slow response with the risk that the EV3 bike won't be stable in case of an abrupt variation. For this reason, it is better to increase also the weight on the lean angular velocity, making the control a bit nervous but clearly more responsive.
As described in the previous chapter the state x consists of the two lean and steering angle, and related angular velocities. Thanks to the sensors we can access to: lean angular velocity and steering angle. Integrating the velocity to obtain the lean angle and deriving the steering angle to get the angular steering velocity, we have all the four components of the state available.

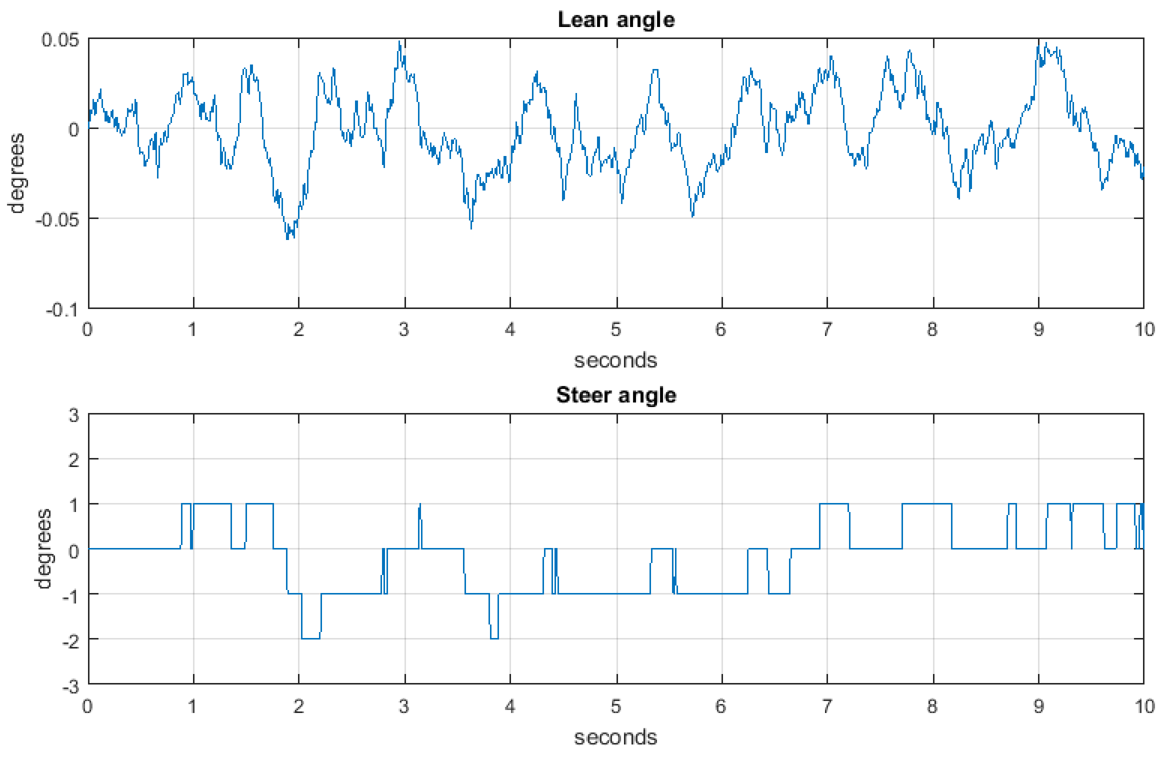

Figure 3.2: Lean and steer angles acquired during simulation. In the second graphic the effect of the quantizer block (modelling the encoder resolution) is visible.

As we can see from the graphs in figure (3.1.1), the control law seems to stabilize the bike but the physical implementation gives a negative result. So we decided to design an observer to rebuild the components of the state with no direct access from the sensors.

3.1.2 State Observer

A linear Kalman filter is implemented as observer. This filter, unlike the Luenberger one, considers the noise measurement and noise process and it is the optimal filter to estimate the state with gaussian noise hypotesis. We estimated mean and variance of noise measurement and noise process of the encoder with empirical test in order to get information about them. The Kalman filter correctly reconstructs the state as we can see in figure (3.3).

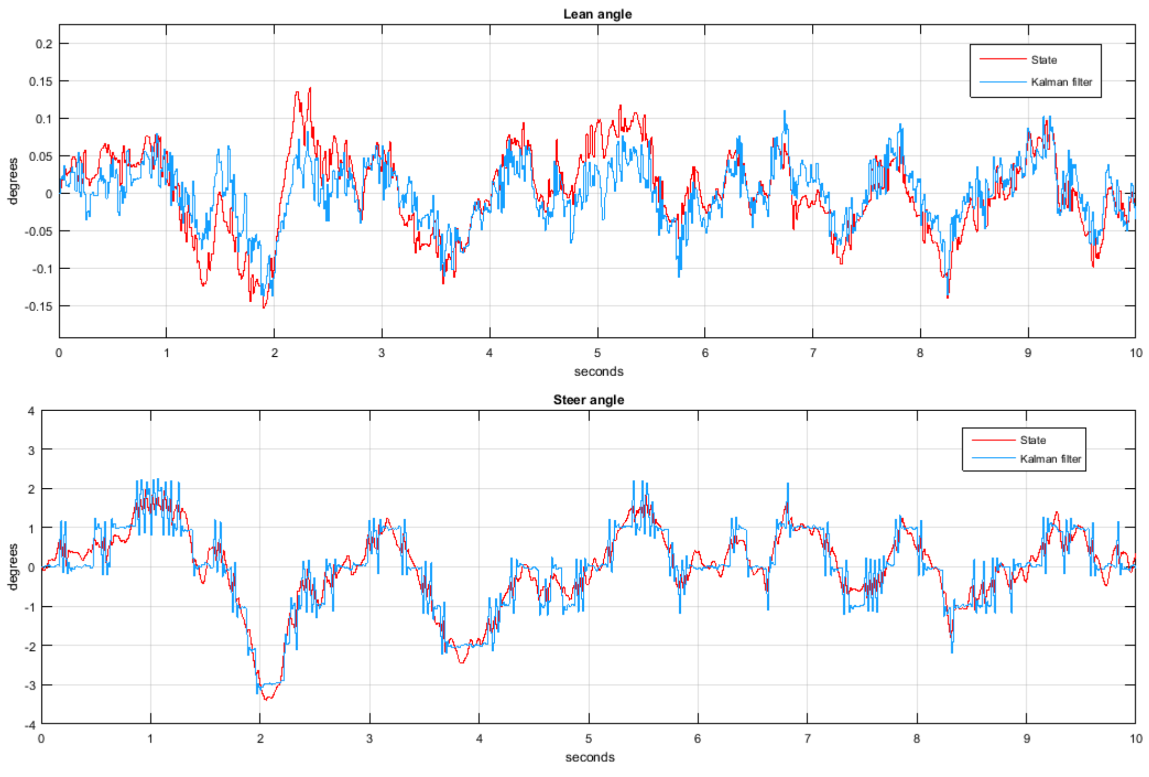

Figure 3.3: Lean and steer angles during simulation

3.2 Application on the EV3 bike

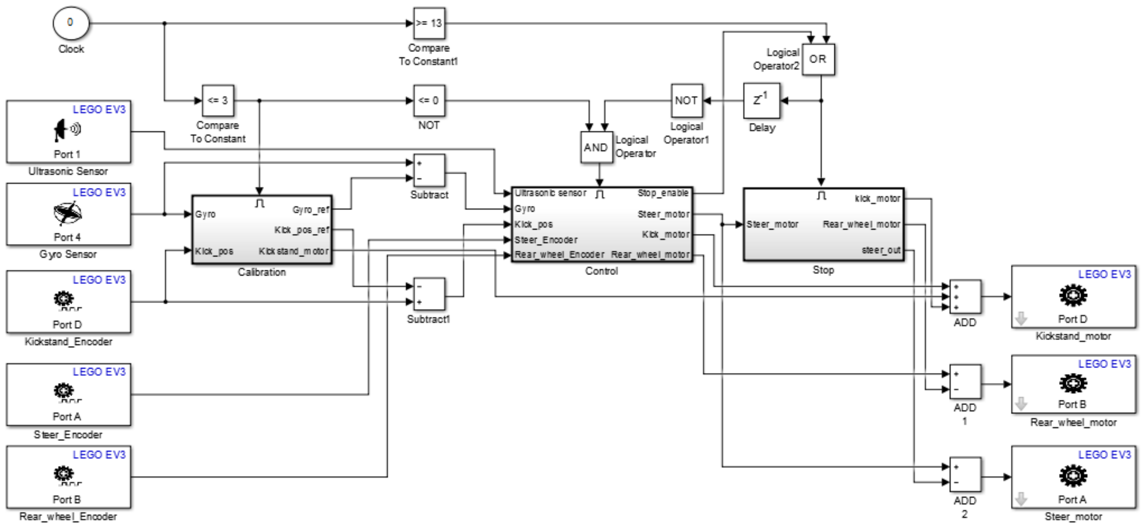

Figure 3.4: Simulink control scheme for EV3 bike

As shown in figure (3.4), the Simulink scheme, used to implement on the EV3 bike the controller previously designed, is composed by three parts: calibration, control and stop.
The "control" subsystem includes the control system that allows the bike

to self-stabilize and mantain a straight trajectory, while the two others include the kickstand and the bike control during start and stop stages. The kickstand allows the bike to start and stop in a vertical position without falling. So when the bike starts the kickstand is down and just goes up after the bike achieves a certain speed. The same for the stop: when the speed is less than a certain value, or the ultrasonic sensor detects an obstacles less than $40cm$ far from the bike, the kickstand goes down to hold the bike in vertical position when it stops. The kickstand is also useful for the calibration of the gyro sensor.

3.2.1 Calibration

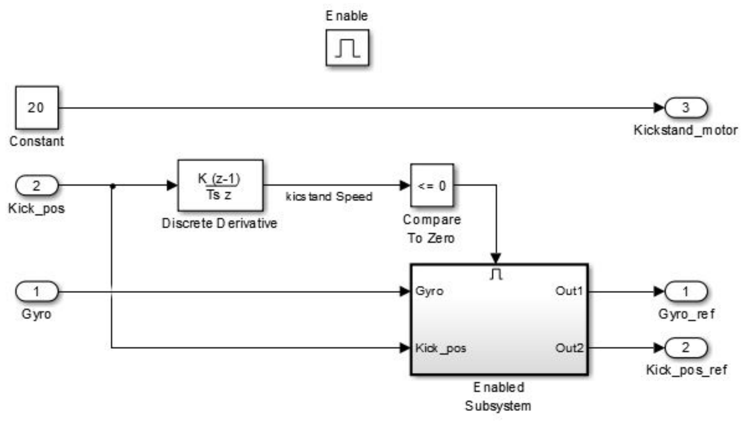

Figure 3.5: Simulink scheme for gyroscope calibration

To calibrate the gyroscope we need to get the offset value of the lean angular velocity provided by the same sensor. For this purpose we put the kickstand down (giving the kickstand motor a duty cycle $D = 20$) and when this stops we assume that the bike is in vertical position, so that the value measured by the gyro sensor is its offset. We also need to memorize the kickstand position as a reference position in order to be able to put the kickstand up while the bike is running.
This subsystem is active for three seconds and after that it holds the output values (gyro offset and kickstand position). These values are then subtracted from the values provided by the sensor and the encoder, thus obtaining a signal for the gyro speed that is zero when the bike is stopped and establishing the kickstand position zero when the kickstand is settled

on the ground.

3.2.2 Control

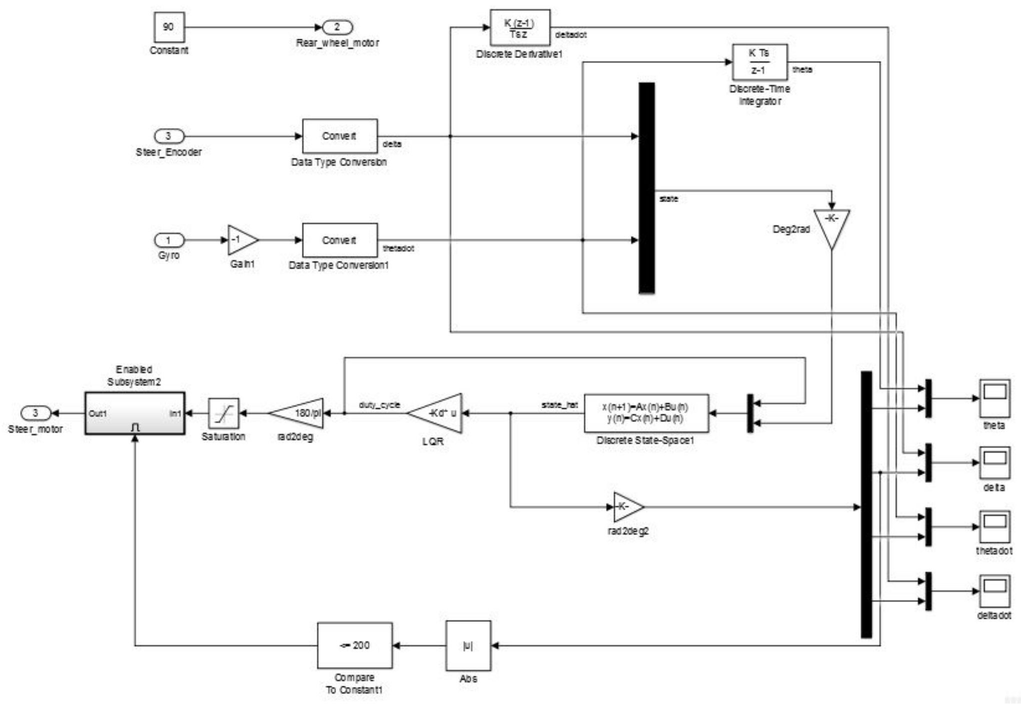

Figure 3.6: Simulink control scheme for EV3 bike

This subsystem is active for ten seconds after the calibration, that is while the bike is moving, and it contains the part for the kickstand (figure (3.7)) and the control system (figure (3.6)). At the beginning, when the bike starts moving, the kickstand is down and goes up when the bike reaches a certain speed. If during this time the bike finds an obstacle that makes it stop or go slower than a certain value, the kickstand goes down again. The signal "kickpos" that is an input for the system is the position of the kickstand already corrected. The subsystem "kickstanddown" just settles the kickstand on the ground and the subsystem "Kickstandup" puts the kickstand up 30 deg. Both the exits of these subsystems return zero when the subsystems are disabled.
The control system is the same one applied during simulations, but the input and the output blocks are substituted with the ones in the Simulink Mindstorms EV3 library. We also added a control that stops the steer motor if the steer angle δ exceeds 200 deg. Figure (3.8) shows the result

of an experiment.

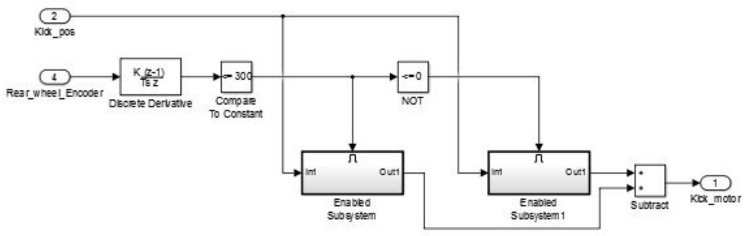

Figure 3.7: Kickstand control while the EV3 is moving

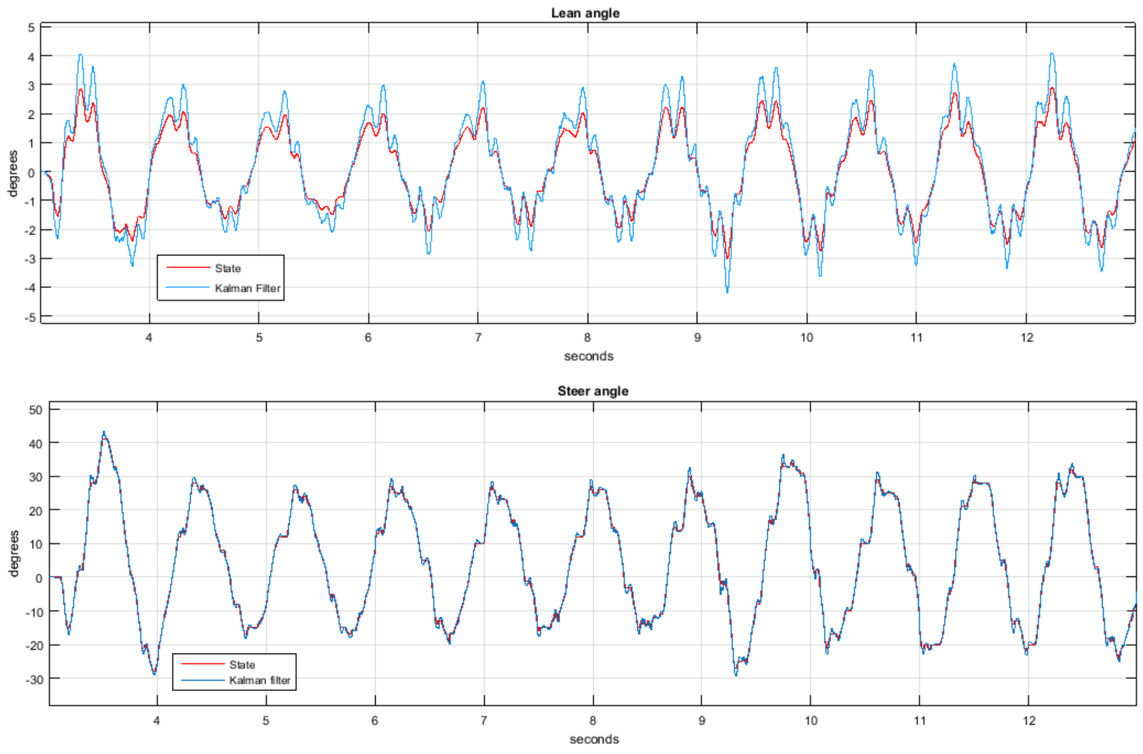

Figure 3.8: Lean and steer angles during an experiment

3.2.3 Stop

In this subsystem (figure (3.9)), that is activated after ten seconds of running mode, the kickstand is just putted down till the program stops running on the bike. Furthermore the speed of the EV3 bike is decreased till the bike stops. This subsystem is also enabled if the ultrasonic sensor detects an obsacle, as previously said. The total time of the experiment, here 13s due to limited space, can be increased. Moreover the condition that stops the bike after a certain time can be deleted, so that the bike stops only if the ultrasonic sensor detects an obstacle.

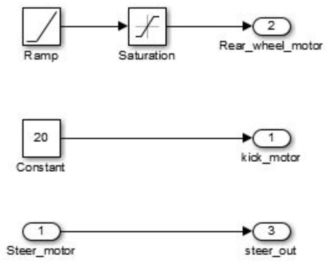

Figure 3.9: Control while the EV3 bike stops

3.3 Alternative configurations

In order to improve the EV3 bike behavior, we tried different quantity and disposition of gyro sensors. If two sensors are used, the lean angular velocity value is obtained as the average of the signals provided by the two gyro sensors.
Initially, we introduced the second gyro sensor higher than the first one, putting it above the EV3 block. In this different position the up gyro sensor reveals greater oscillations of lean angular velocity.

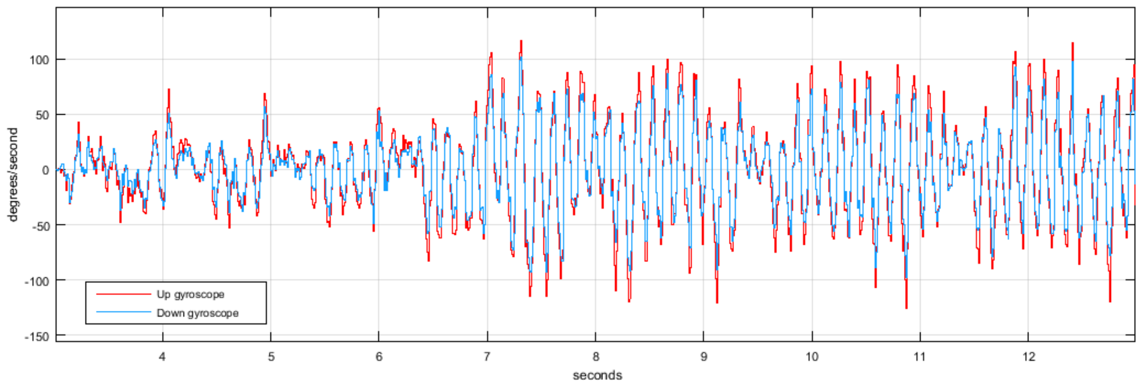

Figure 3.10: Comparison between lean angular velocities provided by up and down gyroscopes.

Even though we used the average between the values of the two gyrosensors, this configuration doesn't introduce improvements.
In the other configuration tried, the second gyrosensor was placed close to the first one. In this way, using the average, the spikes of the final signal are reduced.

Nevertheless, we prefer to use one gyrosensor in order to build the EV3 bike with only one Lego Mindstorms kit and also because the improvements obtained using more than one sensor are not so significant.
The last configuration tried considers the motors in reverse position. This way the pitch increases and the main block has a lower inclination from the ground but the mass distribution is not uniform. Applying the control system previously designed to the case at hand (after changing the mathematical model) we obtained worse results and the bike was unstable.

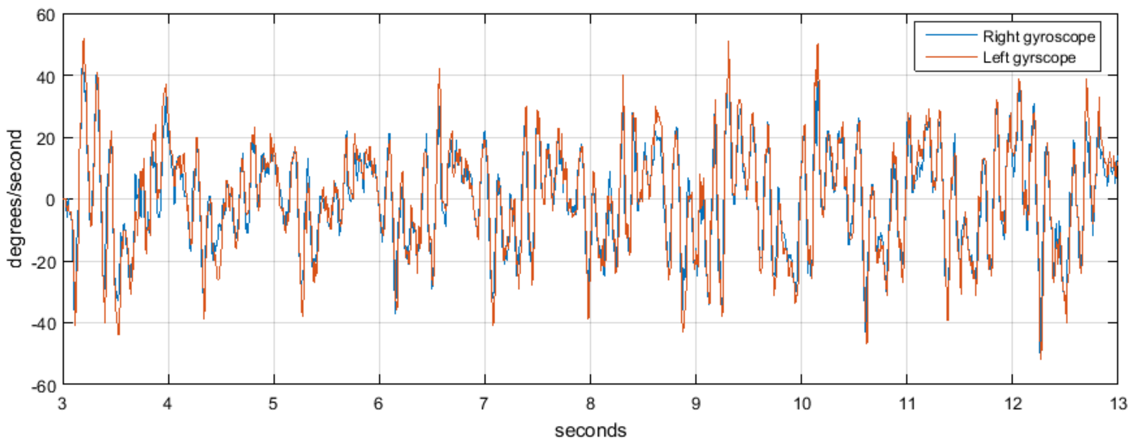

Figure 3.11: Comparison between lean angular velocities provided by the right and the left gyroscopes.

Chapter 4

Conclusion

In this work, we successfully developed the EV3 Bike project, based on a self-stibilizing bike model realized with LEGO Mindstorms platform.
Starting from the mathematical model reported in [2], we designed the control system that allows the bike to follow a straight trajectory.
During the project we had to manage several problems. The first one is due to the high flexibility of plastic LEGO bricks, like the steer play, however improved in comparison with the previous NXT-Bike version. The greater problem is the resolution and the reliability of the gyrosensor, moreover there is a high noise ratio in the signal acquired by this sensor. Also the software makes some troubles, in fact we needed to install the firmware many times. The impossibility to establish a reference initial position of the front wheel is an other disadvantage for the control. Finally, the battery charge affects the EV3 Bike behavior, because the performance quickly decreases. The trajectory tracking and the use of an infrared joystick could be a possible future developments.

Appendix A

Matlab script

```
g=9.81;
V=0.9*0.59;   %60->0.355, 70->0.42, 80->0.46, 90->0.525, 100->0.59
Vmax=10;

% DC motor
ktra=0.007806;  %kt/Ra
ba=0.006689;
J=0.0002873;    %0.0013

%bike parameters
w=0.221;    %pitch
t=0.0237;   %trail
alfa=60;
lambda=30;  %caster angle

%rotation matrix(30\deg around y)
        R=[cosd(lambda) 0 sind(lambda);
            0 1 0;
            -sind(lambda) 0 cosd(lambda)];

%rear wheel (hollow cylinder)
s=0.015;                %thickness
R_rw=0.041;             %radius
m_rw=0.0346;            %mass
A_xx=1/12*m_rw*(3*(R_rw^2+0.026^2)+s^2);
A_yy=1/2*m_rw*(R_rw^2+0.026^2);
A_zz=A_xx;

%front wheel

    %R block (hollow cylinder)
        m_fw=0.0297;
        R_fw=0.041;
        IxxR=1/12*m_fw*(3*(R_fw^2+0.026^2)+s^2);
        IyyR=1/2*m_fw*(R_fw^2+0.026^2);
        IzzR=IxxR;
    %A block(cylinder)
        mA=0.0013;
        lA=0.047;   %lenght
        dA=0.005;   %diameter
        IxxA=1/12*mA*(3*dA/2+lA^2);
        IyyA=mA*dA/4;
        IzzA=IxxA;

     %composition
        D_xx=IxxA+IxxR;
        D_yy=IyyA+IyyR;
        D_zz=IzzA+IzzR;
%rear frame
    %F block (parallelepiped)
        m1=0.3924;
        l1=0.206;   %lenght
        d1=0.087;   %width
        h1=0.048;   %height
        r1=0.02;
        r1z=0.015;

        IxxF=1/12*m1*(d1^2+h1^2)+m1*r1^2;
        IyyF=1/12*m1*(l1^2+h1^2)+m1*r1^2;
        IzzF=1/12*m1*(l1^2+d1^2)+m1*r1z^2;
        TF=R*[IxxF 0 0; 0 IyyF 0; 0 0 IzzF]*R';

    %G block(parallelepiped)
        m2=0.2930;
        l2=0.110;
        d2=0.087;
        h2=0.052;
        r2=0.02;
        r2x=0.017;

        IxxG=1/12*m2*(d2^2+h2^2)+m2*r2x^2;
        IyyG=1/12*m2*(l2^2+h2^2)+m2*r2^2;
        IzzG=1/12*m2*(l2^2+d2^2)+m2*r2^2;
        TG=R*[IxxG 0 0; 0 IyyG 0; 0 0 IzzG]*R';

    %F block and G block composition
        x_rf=0.0768;
        y_rf=0;
        z_rf=-0.1005;
        m_rf=0.6824;
        Tretro=TF+TG;
```

```
B_xx=Tretro(1,1);
B_xz=Tretro(1,3);
B_yy=Tretro(2,2);
B_zx=B_xz;
B_zz=Tretro(3,3);

%front frame
    %C,B blocks (cylinders)
        mB=0.0016;
        lB=0.07;
        dB=0.005;
        rB=0.028;
        rBz=0.016;

        IxxBC=2*(1/12*mB*(3*dB/2+lB^2)+mB*rB^2);
        IyyBC=IxxBC;
        IzzBC=mB*dB/4+mB*rBz^2;

    %D block (parallelepiped)
        mD=0.0084;
        lD=0.04;
        dD=0.024;
        hD=0.014;

        IxxD=1/12*mD*(lD^2+hD^2);
        IyyD=1/12*mD*(dD^2+hD^2);
        IzzD=1/12*mD*(lD^2+dD^2);

    %E block (cylinder)
        mE=0.0016;
        lE=0.07;
        dE=0.005;

        IxxE=1/12*mE*(3*dE/2+lE^2);
        IyyE=IxxE;
        IzzE=mE*dE/4;

    %rear frame composition
        x_ff=0.207;
        y_ff=0;
        z_ff=-0.09;
        m_ff=0.0160;

        IxxAva=IxxBC+IxxD+(mB*2+mD)*0.023^2+IxxE;
        IyyAva=IyyBC+IyyD+(mB*2+mD)*0.023^2+IyyE;
        IzzAva=IzzBC+IzzD+IzzE;
        TA=R*[IxxAva 0 0; 0 IyyAva 0; 0 0 IzzAva]*R';
```

```
C_xx=TA(1,1);
C_xz=TA(1,3);
C_yy=TA(2,2);
C_zx=C_xz;
C_zz=TA(3,3);

%masses, center of mass, inertiae
m_t = m_rw + m_rf + m_ff + m_fw;
x_t = (m_rf*x_rf+m_ff*x_ff+m_fw*w)/m_t;
z_t=(-R_rw*m_rw+m_rf*z_rf+m_ff*z_ff-m_fw*R_fw)/m_t;

T_xx=A_xx+B_xx+C_xx+D_xx+m_rw*R_rw^2+m_rf*z_rf^2+m_ff*z_ff^2+m_fw*R_fw^2;
T_xz=B_xz+C_xz-m_rf*x_rf*z_rf-m_ff*x_ff*z_ff+m_fw*w*R_fw;
T_zz=A_zz+B_zz+C_zz+D_zz+m_rf*x_rf^2+m_ff*x_ff^2+m_fw*w^2;

m_f=m_ff+m_fw;
x_f=(m_ff*x_ff+m_fw*w)/m_f;
z_f=(m_ff*z_ff-m_fw*R_fw)/m_f;

F_xx=C_xx+D_xx+m_ff*(z_ff-z_f)^2+m_fw*(R_fw+z_f)^2;
F_xz=C_xz-m_ff*(x_ff-x_f)*(z_ff-z_f)-m_fw*(w-x_f)*(R_fw+z_f);
F_zz=C_zz+D_zz+m_ff*(x_ff-x_f)^2+m_fw*(w-x_f)^2;

cf=t*cosd(lambda);
u=-z_f*sind(lambda)+(x_f-w)*cosd(lambda)-cf;

F_ll=m_f*u^2+F_xx*sind(lambda)^2+
2*F_xz*sind(lambda)*cosd(lambda)+F_zz*cosd(lambda)^2;
F_lx=-m_f*u*z_f+F_xx*sind(lambda)+F_xz*cosd(lambda);
F_lz=m_f*u*x_f+F_xz*sind(lambda)+F_zz*cosd(lambda);

H_r=A_yy/R_rw*V;
H_f=D_yy/R_fw*V;
H_t=H_r+H_f;

ni=m_f*u+(cf*m_t*x_t)/w;

M11=T_xx;
M12=F_lx+cf/w*T_xz;
M21=M12;
M22=F_ll+2*cf/w*F_lz+(cf/w)^2*T_zz+J;

M_m=[M11 M12; M21 M22];

C11=0;
C12=cf/w*H_t+H_f*cosd(lambda)+V*(T_xz*cosd(lambda)/w-cf/w*m_t*z_t);
C21=-cf/w*H_t-H_f*cosd(lambda);
C22=V*(cosd(lambda)/w*(F_lz+cf/w*T_zz)+cf/w*ni);
```

```

C_m=[C11 C12; C21 C22];

K11=g*m_t*z_t;
K12=-g*ni+(H_t*V-V^2*m_t*z_t)*cosd(lambda)/w;
K21=-g*ni;
K22=-g*ni*sind(lambda)+(V^2*ni+V*H_f*sind(lambda))*cosd(lambda)/w;

K_m=[K11 K12; K21 K22];

%state equations
Mm=M_m^-1;
att=[zeros(2,1) -Mm(1:2,2)]*ba;

A=[zeros(2) eye(2);
   -Mm*K_m -Mm*C_m+att];

B=[zeros(2,1);
   Mm(1:2,2)]*Vmax*ktra;

G=[zeros(2);
   Mm];


C=[1 0 0 0;0 1 0 0;0 0 0.985 0;0 0 0 1];

D=[0;
   0;
   0;
   0];

C1=[0 1 0 0;
    0 0 0.985 0];

H=zeros(4,2);

%sensors parameters
Ts=0.01;
quantiz=1;         %quantization error
WN_power=1.5e-6;   %gyroscope power noise
WN_Ts=0.03;        %gyroscope band noise
offset_gyro=0;%0.0054;   %gyroscope offset

%process noise
pr_delta=1e-7;      %steer torque noise power
pr_theta=1e-7;      %lean torque noise power

```

```
varGyro=WN_power/WN_Ts;     %gyroscope noise power variance
varEnc=1e-7;                %steer encoder mesasure noise variance
varTdelta=pr_delta/Ts;      %steer torque process noise variance
varTtheta=pr_theta/Ts;      %lean torque process noise variance
Qn=[varTtheta 0;0 varTdelta];        %process noise covariance
Rn=[varEnc 0;0 varGyro];             %measure noise covariance

%lqr
Q=[10 0 0 0; 0 5 0 0; 0 0 10 0; 0 0 0 1];
Rc=5;                              %input weight
[K,S,E]=lqr(A,B,Q,Rc);
[Kd,Sd,Ed]=lqrd(A,B,Q,Rc,0.01);
%kalman time continuos
sys=ss(A,[B G],C1,zeros(2,3));
[KEST,L,P]=kalman(sys,Qn,Rn);
[Ac,Bc,Cc,Dc]=ssdata(KEST);

%kalman time discrete
sysd=c2d(sys,Ts);
[KESTD,Lk,Pk]=kalman(sysd,Qn,Rn,'delayed');
[Ad,Bd,Cd,Dd]=ssdata(sysd);
[Ak,Bk,Ck,Dk]=ssdata(KESTD);
```

Bibliography

[1] A. Rosa. *Analisi e controllo di un modello di bici realizzato su piattaforma Lego Mindstorms.* Master's thesis, Scuola di Ingegneria, Università di Firenze, 2013.

[2] Basso, Michele, and Giacomo Innocenti. *"Lego bike: A challenging robotic lab project to illustrate rapid prototyping in the mindstorms/simulink integrated platform."* Computer Applications in Engineering Education 23.6 (2015): 947-958.