CHALMERS
UNIVERSITY OF TECHNOLOGY

Trajectory control with Kalman Filter-based State Estimation for bicycle

Master's thesis in System, Control and Mechatronics

GAIZKA BARRASA PEREZ

DEPARTMENT OF ELECTRICAL ENGINEERING

CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2023
www.chalmers.se

Master's thesis 2023

Trajectory control with Kalman Filter-based State Estimation for bicycle

GAIZKA BARRASA PEREZ

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Electrical Engineering
Division of System and Control
Autobike research group
Chalmers University of Technology
Gothenburg, Sweden 2023

Trajectory control with Kalman Filter-based State Estimation for bicycle
GAIZKA BARRASA PEREZ

© GAIZKA BARRASA PEREZ, 2023.

Supervisor: Jonas Sjöberg
Examiner: Jonas Sjöberg

Master's Thesis 2023
Department of Electrical Engineering
Division of System and control
Autobike research group
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2023

Trajectory control with Kalman Filter-based State Estimation for bicycle
GAIZKA BARRASA PEREZ
Department of Electrical Engineering
Chalmers University of Technology

Abstract

The testing of autonomous vehicles presents a challenge in ensuring safety. This study addresses the challenge by developing an autonomous bicycle capable of following predefined paths. Consequently, it eliminates the requirement for human bike riders in testing scenarios involving bicycles and vehicles. Instead, they are replaced by dummies.

The study involves the development of a Kalman filter and the enhancement of the trajectory controller algorithm. The algorithms are validated in Simulink before being transformed into C-code and inserted in LabVIEW which is run on the real bicycle's myRIO (microprocessor).

Finally, the trajectory generation, parameter acquisition (MATLAB), program execution (LabVIEW), and the initialization protocol from program launch to the achievement of autonomous motion for testing with the real bicycle are explained. Additionally, a test is presented where the bicycle tracks three constant radius circles where the developed functionalities and their repeatability are evaluated in the real bike. The result of the test is that the bike can track the reference with a maximum error of $\pm 0.5m$.

The algorithms are parameterized to be adapted for various bikes, regardless of their dimensions and characteristics.

Keywords: testing, autonomous bicycle, Kalman filter, trajectory controller, parameterized algorithms.

Acknowledgements

I would like to extend my heartfelt gratitude to Professor Jonas Sjöberg for his invaluable guidance and mentorship throughout this journey. He taught me how to structure problems and efficiently identify errors in this project. His expertise and insightful feedback played an essential role in shaping this work.

I am deeply appreciative of my colleagues Levi Stevens, Mario Grieco, Lorenzo Benedetto, and many more for their collaboration stimulating discussions, and support in the tough moments. Their diverse perspectives enriched the development of this work.

To my friends, your encouragement and companionship provided a constant source of motivation. Your enduring belief in my abilities kept me going during challenging times.

I am grateful to my family for their unconditional love and support. Their sacrifices enabled me to pursue my academic aspirations.

Lastly, I want to acknowledge the members of different companies that took part in the project for providing the necessary resources.

Gaizka Barrasa Perez, Gothenburg, August 2023

List of Acronyms

Below is the list of acronyms that are used throughout this thesis listed in alphabetical order:

CPA	Closest point algorithm
ESC	Electronic speed controller
FPGA	Field Programmable Gate Array
GNSS	Global Navigation Satellite System
GPS	Global positioning system
IMU	Inertia measurement unit
KF	Kalman filter
LQR	Linear quadratic regulator
LTI	Linear time-invariant
NI	National Instruments
PDB	Power distribution board
RPM	Revolutions per minute
SBC	System Basis Chip

Nomenclature

Below is the nomenclature of indices, sets, parameters, and variables that are used throughout this thesis.

Indices

G	Global frame
L	Local frame
x	Respect to x
y	Respect to y
z	Respect to z
K	Relates to Kalman filter
T	Relates to trajectory controller
v	Corresponding velocity
x_i	Component i of a vector x

Parameters

General parameters:

g	Gravity constant $[m/s]$

Bike parameters:

h	Height of the bike's center of mass $[m]$
h_{IMU}	Height of the IMU $[m]$
J_x	Moment of inertia of the bike with respect to x-axis
J_{xz}	Moment of inertia of the bike with respect to x-z axes
l_r	Distance of the rear axle to the center of mass $[m]$
l_f	Distance of the front axle to the center of mass $[m]$

m	Total mass of the bike $[kg]$
R	Radius from the bike to center of rotation $[m]$
λ	Angle of the fork axis $[Rad]$
K	Kalman gain

Variables

L_x	Angular momentum of the system with respect to x-axis
t	Time $[s]$
T_δ	Resulting torque from steering $[Kgm/s^2]$
v	Velocity of the bike $[m/s]$
X_G	Bike position on the x-axis in global frame
Y_G	Bike position on the y-axis in global frame
X_L	Bike position on the x-axis in local frame
Y_L	Bike position on the x-axis in local frame
β	Direction of motion of the bike $[Rad]$
δ	Steering angle of the bike $[Rad]$
δ_e	Effective steering angle of the bike $[Rad]$
φ	Roll of the bike $[Rad]$
ψ	Heading of the bike $[Rad]$
O	Center of rotation of the bike

Contents

List of Figures

List of Tables

1 Introduction

1.1 Problem Background

The automotive industry conducts tests to evaluate the performance of vehicle functionalities like detection systems and automatic braking. For instance, in the scenario depicted in Figure 1.1, a biker and a vehicle (yellow rectangles) approach a crossing, posing a potential risk of collision. This is a possible real-world scenario for testing automatic braking, which must detect the bicycle, predict its movements, and apply the brakes.

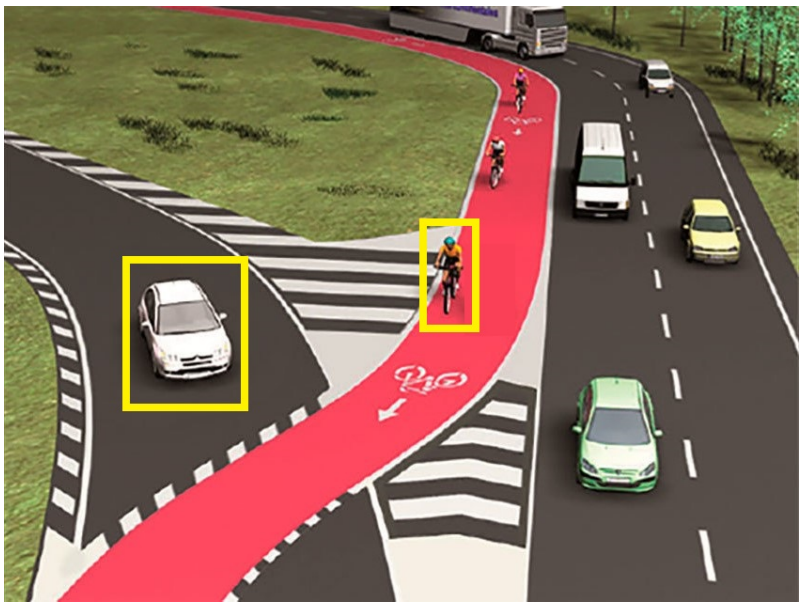


Figure 1.1: Bicycle and car approaching an intersection. Example of a real-world scenario.

Due to safety matters, it is not possible to perform this test in real scenarios. Therefore, the traffic situation is designed and set up on the test site. In the test site, trajectories are designed for both so that the bicycle crosses in front of the vehicle. For that, they must precisely follow the predefined paths and speeds. Considering these potentially hazardous situations, it is possible to use a robot bike that eliminates the need to involve human riders. Thereby, preventing them from any kind of harm. The riders are replaced by dummies to maintain realism. The precise recreation of the scenario ensures accurate testing.

Automatic bicycles can execute predefined maneuvers and follow paths with high precision, eliminating the variability associated with human riders. Therefore, they guarantee a consistent and reproducible execution of tests, minimizing variations from one test to the next. Consequently, it is possible to compare results across numerous test runs, improving the reliability of the data.

This work is about autonomous bicycles to be used in those tests. To follow the predefined paths, the position, orientation, and velocity of the bicycle need to be controlled. Moreover, the bicycle must be balanced to prevent it from falling, so the roll angle is also controlled. These variables describe the state of the bicycle and are illustrated in the Figure 1.2. Thus, the state vector is defined as:

$$x^T = \Big[X_G, Y_G, \psi, \varphi, \dot{\varphi}, \delta, v\Big]. \tag{1.1}$$

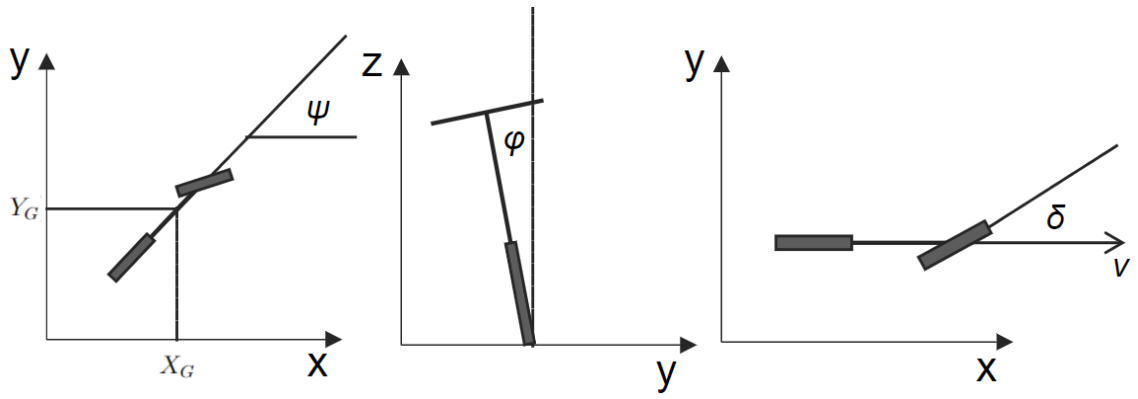


Figure 1.2: : Bike states: Position (X_G,Y_G) in global frame, orientation ψ, roll φ, roll rate $\dot{\varphi}$, steering angle δ, velocity v.

Both trajectory and balance controllers use state estimation as feedback signals. To ensure the precision of them, an accurate state estimator is required.

1.2 Bicycle control diagram

In the next subsections, the different subsystems of the control diagram shown in the Figure 1.3 are briefly introduced to provide a general overview of the system. The trajectory controller and state estimator are the focus of the study. Therefore, their implementation and validation are detailed throughout the report.

1.2.1 Reference path

The reference path is a set of points that contains the position coordinates (X_{ref},Y_{ref}) in the global frame and the orientation (ψ_{ref}) that the bicycle should follow.

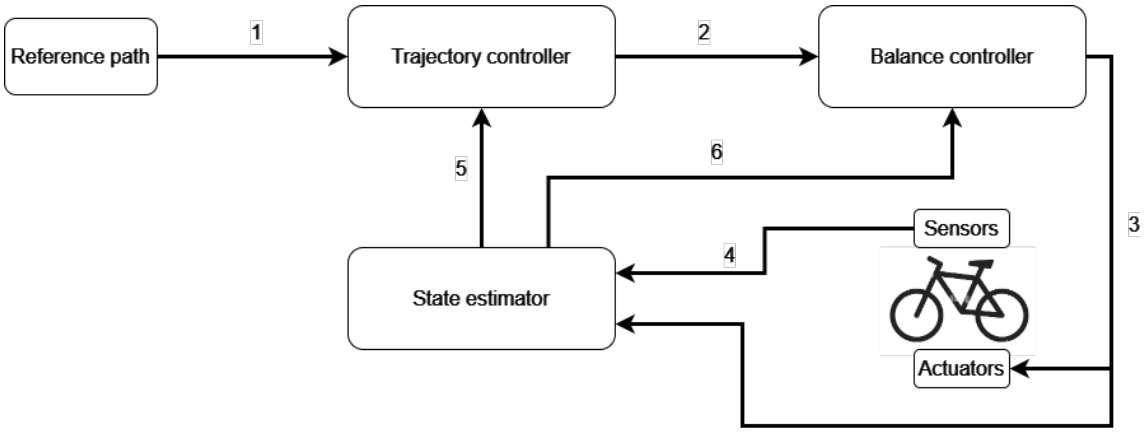


Figure 1.3: Control diagram of the bike.

1.2.2 Trajectory controller

The trajectory controller receives the reference path (arrow 1) and the position and orientation estimations (arrow 5). Given these, the controller calculates the bike's lateral and direction error towards the reference to generate the steering angle (δ_{ref}) that the bike should have to follow the predefined path (see Figure 1.4).

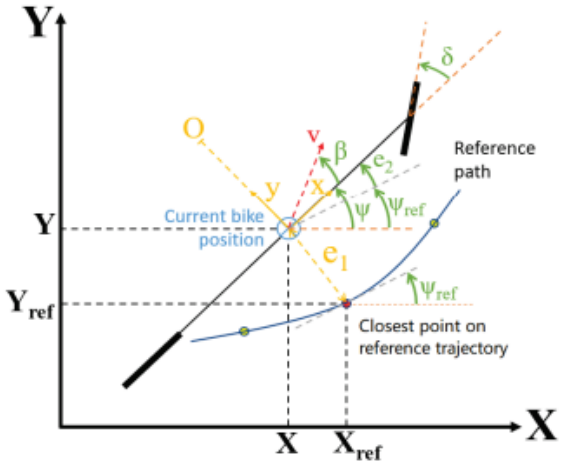


Figure 1.4: Lateral (e_1) and heading (e_2) errors towards the reference path.

The δ_{ref} is transformed into a roll angle (φ_{ref}). This transformation is described in the Appendix A. This controller is explained in detail in Chapter 5.

1.2.3 Balancing controller

The balancing controller receives the φ_{ref} (arrow 2). This PD controller compares the φ_{ref} with the roll ($\hat{\varphi}$) and roll rate ($\dot{\hat{\varphi}}$) estimations (arrow 6) to compute the steering rate ($\dot{\delta}$, input of the steering motor (arrow 3)) so that the bicycle is balanced.

1.2.4 Sensors and actuators

The bikes incorporate two actuators (forward and steering motors) and four sensors (GPS, encoder, IMU, and ESC). The actuators allow the movement and balance of the bicycle. The sensors provide the measurements (arrow 4) that the state estimator uses to estimate its state. In Chapter 2, the hardware of the bike is detailed more.

1.2.5 State estimator

The state estimator receives the measurements from the sensors and the $\dot{\delta}$. Given these inputs, the state estimator calculates an online estimate of the state of the bicycle (see Section 1.1). These state signals are then fed back to the two controllers (arrows 5 and 6). The chosen state estimator is the Kalman filter, which is explained in detail Chapter 4.

1.3 Contributions

This report focuses on the design and validation of the Kalman filter together with the improvement of the trajectory controller algorithm. Besides these main areas of study, there have been many smaller contributions which are listed here:

- Simulation (Simulink):
  - Reorganization of the simulation structure in Simulink to look similar to the Figure 1.3.
  - Implementation and validation of Kalman filter.
  - Improvement of the closest point algorithm in trajectory controller.
  - Very basic trajectory creator interface.
- Real bike (LabVIEW):
  - Transformation of both Kalman filter and trajectory controller algorithms from MATLAB to C-code to be used in LabVIEW.
  - Update LabVIEW structure to include both Kalman filter and trajectory controller blocks.
  - Tuning of both balancing and trajectory controller parameters in the real bike.
- Validation that all the software is also functional on the green (portable) bike. The parameters of the bikes are different such as dimensions, weight, etc. As a result, different values for the tunable parameters will probably be required. However, the structure of the control diagram is the same.
- Solved problems:
  - Separate logging and visualization blocks in LabVIEW. In addition, the visualization loop frequency has been reduced from 100Hz to 10Hz to avoid the delay of the rest of the blocks in the structure.
  - GPS configuration in the red bike (maximum accuracy was obtained but the data was provided with a 4s approx. delay).

2

Hardware description

In this chapter, the hardware of the bicycle is briefly described. Most of the components are located inside a box, but others are attached in different places to the bike.

Figure 2.1: Hardware outside the electrical box.

1. Steering motor DCX 32L (Maxon) with gearhead reduction GPX 32.
2. Encoder HEDS 5540 (Maxon).
3. GPS antenna.
4. Forward motor (Shimano steps).
5. Emergency stop (Schneider Electric).

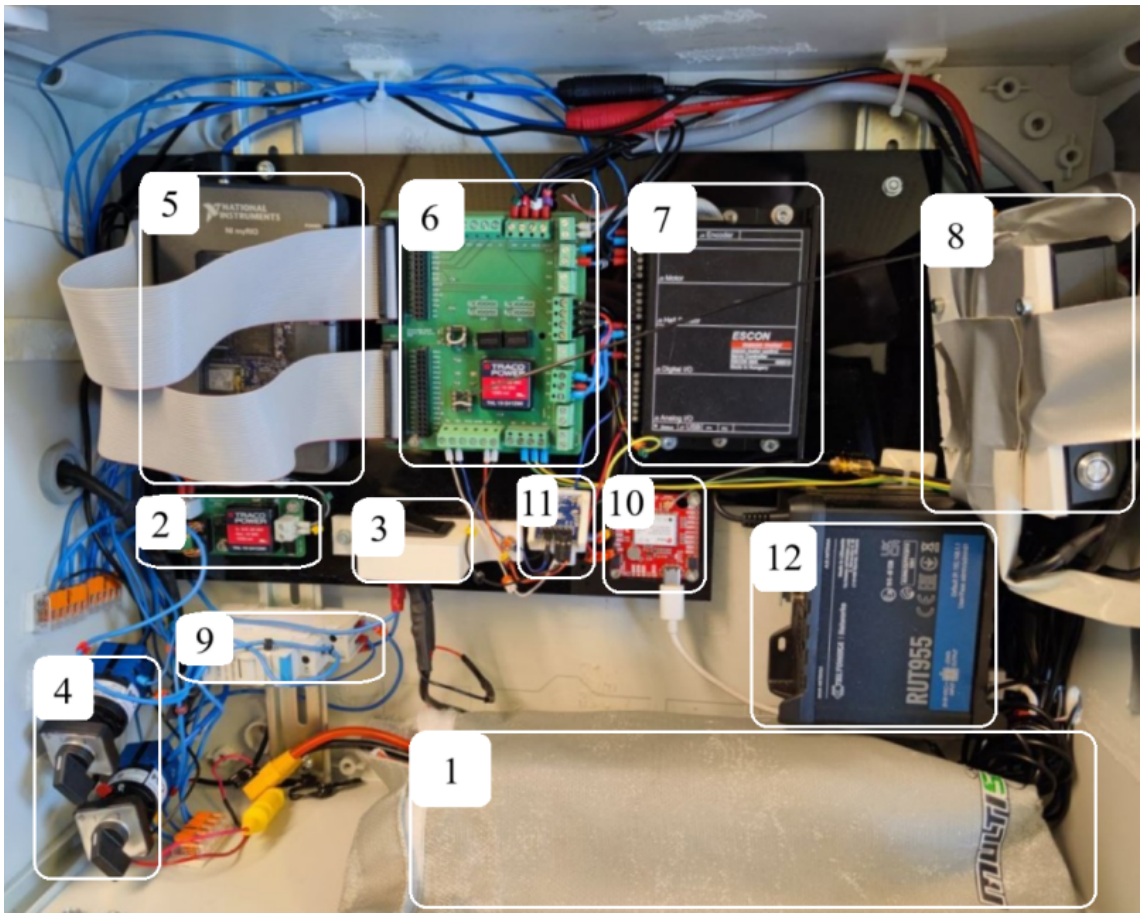


Figure 2.2: Hardware inside the electrical box.

1. Battery: 6-cell lithium battery able to supply voltages of 22.2V DC.
2. PDB: Power distribution board.
3. Power switch.
4. ON-OFF power supply switches for motor drivers, one switch for each.
5. NI myRio: SBC with an embedded FPGA board and a microprocessor.
6. myRio I/O PCB.
7. Escon 50/5: Steering motor controller by Maxon.
8. FSESC 6.7 PRO: Forward motor controller.
9. Relay: Two poles relay from Finder.
10. ZED-F9P module: GNSS receiver by U-blox.
11. IMU: 3-axis accelerometer/gyroscope/magnetometer (PmodNAV LSM9DS1).
12. RUT955: Bike's router.

Finally, the connections between the hardware components are shown in Figure 2.3:

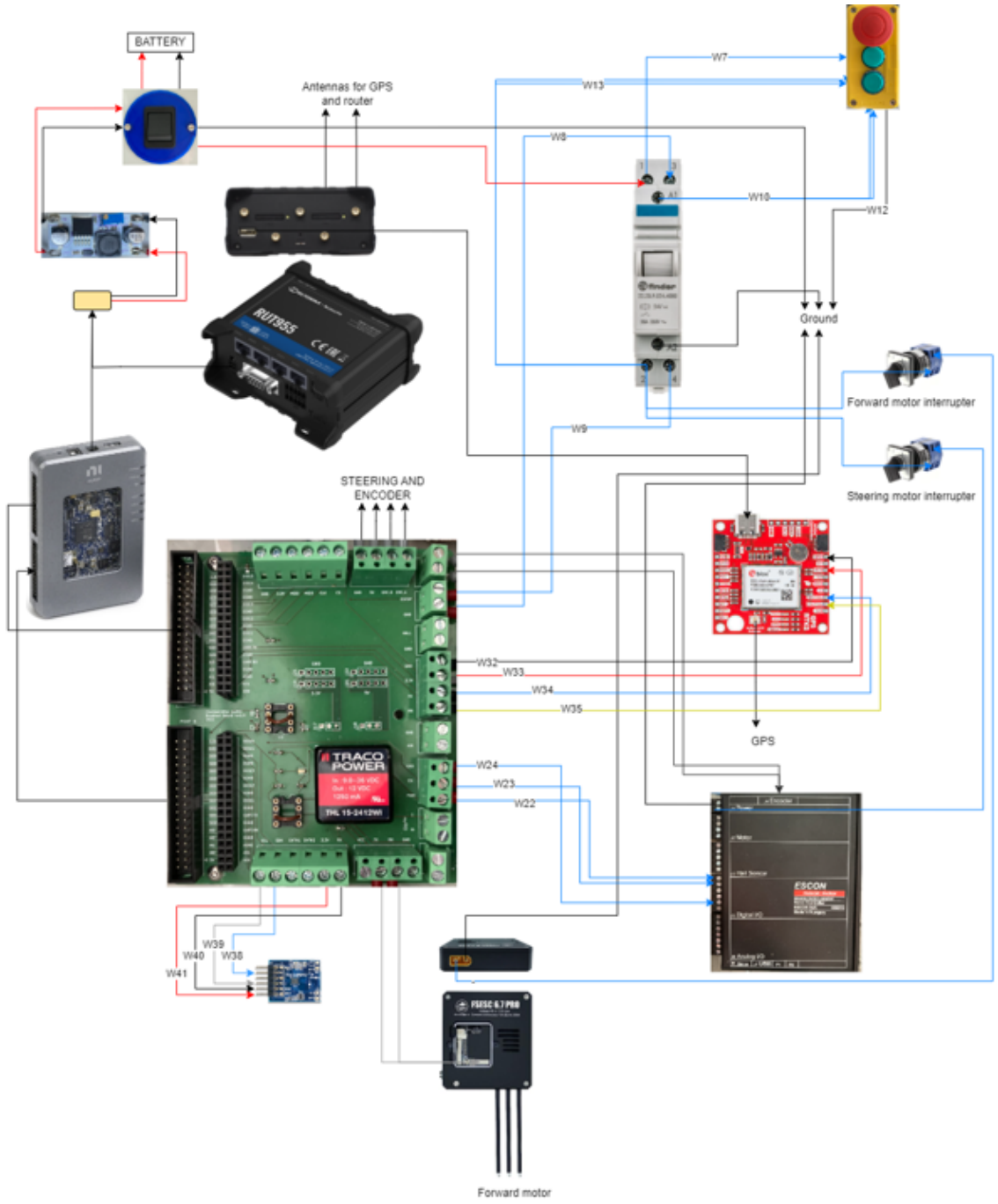


Figure 2.3: The connections between the hardware components of the bike.

2.1 Actuators

The actuators of the bicycle are the following:

1. Steering motor: The front steering wheel of the bike, the DCX32L by Maxon-group. It is used to control the bicycle's handlebar, ensuring a precise direction. It is connected to the motor controller Escon 50/5. The Escon receives

an angle from the myRIO and then it decides the movement of the steering motor. [9] [13]
2. Forward motor: The bicycle is equipped with a forward motor from the Shimano Steps E6000 series. It is a brushless DC motor and is mounted between the pedals of the bicycle. All control inputs to the motor are managed by the motor controller (FSESC 6.7 PRO). It accelerates the bicycle's movement.[12]

2.2 Sensors

The bikes use different sensors to measure some variables or states. These measurements are sent to the Kalman filter to estimate the state of the bike. All the sensors in the bike are listed as:

1. GPS sensor: includes the antenna and the module ZED-F9P GNSS module. It is a highly advanced and precise Global Positioning System with RTK technology. This technology increases the accuracy to the centimeter level. The measurement is given in latitude/longitude coordinates. [11][15]
2. Encoder (HEDS-5540#A11): is an incremental rotary encoder. The maximum velocity is 30000RPM and a maximum resolution of 1024 counts per revolution. The encoder is mounted on the steering motor to measure the steering angle and then transmit it to the myRIO. [10]
3. IMU: is a sensor module designed for measuring various forms of motion, orientation, and environmental data. It measures the acceleration and rotational speed of the bicycle in three axes, combining the accelerometer, gyroscope, and magnetometer. [16]
4. FSESC 6.7 PRO: is a high-performance electronic speed controller (ESC), designed for electric vehicles. This compact and powerful device is engineered to provide precise speed control and efficient power management. It also provides velocity measurement in RPM. [14]

3

Bike model

This chapter describes the bicycle model used within the Kalman filter. It presents the bicycle model, providing the necessary equations for describing the desired state variables in the state space model of the bicycle. This model is used to implement the Kalman filter, which is introduced and elaborated in Chapter 3.

The models are taken from [1] where a more detailed description can be found.

The Figure 3.1 shows some parameters and the direction of the local axes that are necessary for deriving the system models. The Figure 3.2 and 3.3 show the roll and effective steer angles.

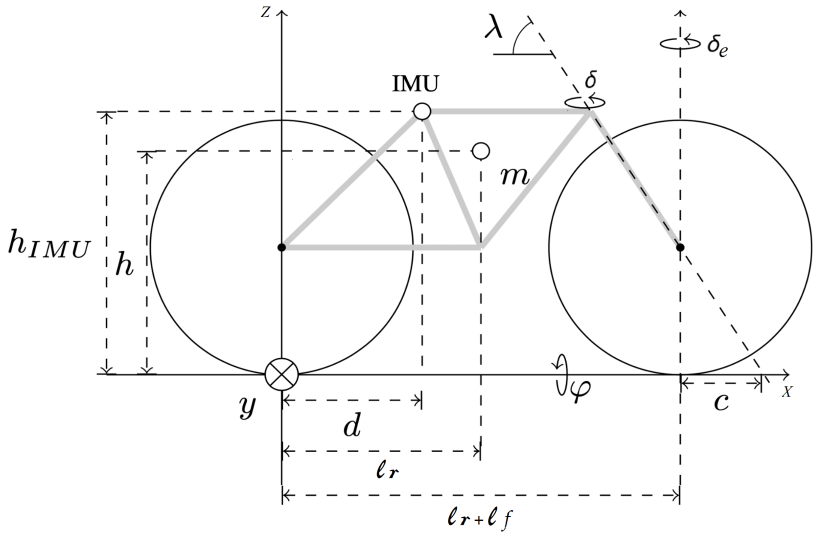


Figure 3.1: Side view of the bike with relevant parameters and direction of the local axes.

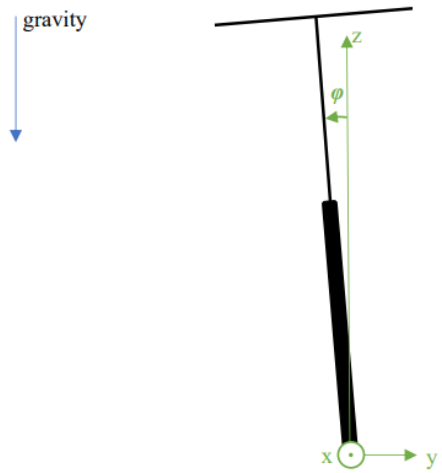


Figure 3.2: Front view of the bike and the roll angle.

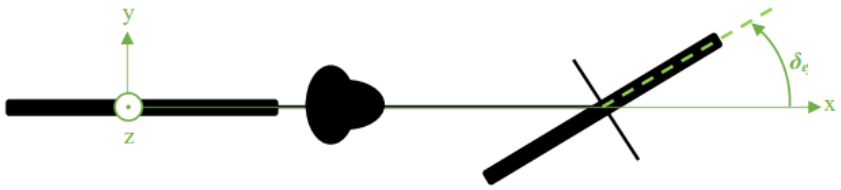


Figure 3.3: Top view of the bike and the effective steer angle.

3.1 Kinematic Model

The kinematic model derived for a bicycle is described by the parameters shown in Figure 3.1.

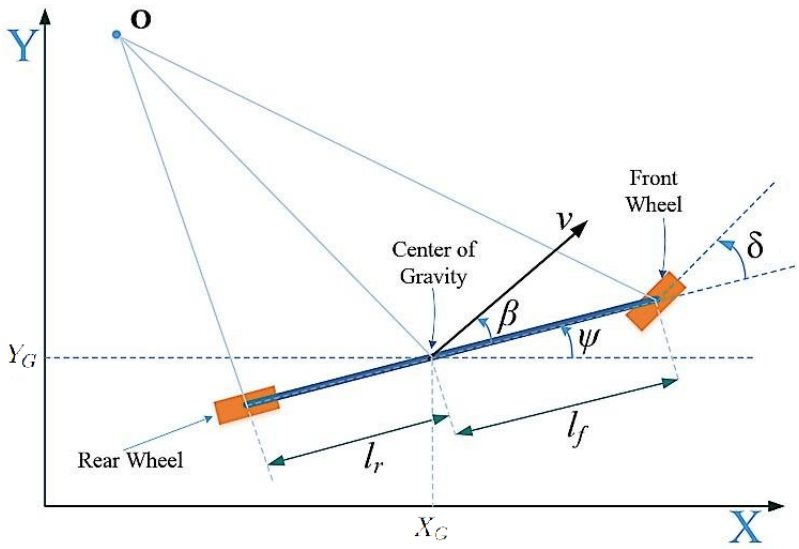


Figure 3.4: Illustration of relevant parameters for the kinematic model of a bicycle.

The model is given by the equations:

$$\begin{cases} \dot{X}_G &= v\cos(\psi+\beta) \\ \dot{Y}_G &= v\sin(\psi+\beta) \\ \dot{\psi} &= \frac{v}{l_r+l_f}\tan(\delta_e)\cos(\beta) \end{cases}. \tag{3.1}$$

Figure 3.4 represents the bicycle's position in a global frame (X_G, Y_G), the heading (ψ), and the velocity (v). The parameters l_r and l_f denote the x-distance from the rear wheel center and the front wheel center to the mass center, respectively. The angular velocity of the center of mass is expressed as a function of the motion direction of the bicycle's body, β. The relationship between β and the effective steering angle, δ_e, can be written as:

$$\beta = \tan^{-1}\left(\frac{l_r}{l_r+l_f}\tan(\delta_e)\right) \tag{3.2}$$

$$\delta_e = \delta\,\sin(\lambda) \tag{3.3}$$

The effective steering angle, δ_e, is measured at the contact surface of the wheel with the ground. It depends on the fork angle (λ), and the observed steering angle of the steering wheel (δ).

3.2 Mathematical model

The model of the bicycle resembles that of an inverted pendulum, where the balancing force is dependent on the steering angle [1].

Using the second Newton's law for rotations, the angular momentum balance of the system around the x-axis is placed on the contact points of the wheels as illustrated in Figure 3.1:

$$\frac{dL_x}{dt} = \underbrace{m\,g\,h\,\sin\varphi}_{T_g} + \underbrace{\frac{m\,v^2\,h}{l_r+l_f}\,\tan\delta_e}_{T_c} + \Big(\underbrace{-\frac{m\,g\,l_r\,c}{l_r+l_f}\,\delta_e}_{T_\delta}\Big) \tag{3.4}$$

where:

- m : bicycle's total mass.
- g : gravitational acceleration.
- h : height of the center of mass.
- φ : roll angle (see Figure 3.2).
- L_x : angular momentum of the system around the x-axis.
- T_g : torque generated by the gravity.
- T_c : torque generated by the centrifugal force.

- T_δ : torque due to the geometry of the front fork. For a non-zero c, the center of mass of the bicycle frame is shifted when the front wheel is turned, which results in an extra torque around the x-axis.

The angular momentum of the system around the x-axis is defined as:

$$L_x = J_{xx}\,\omega_x + J_{xy}\,\omega_y + J_{xz}\,\omega_z \tag{3.5}$$

Considering that the angular rate around the x-axis is the roll rate ($\omega_x = \dot{\varphi}$), the angular rate around the z-axis is the yaw rate ($\omega_z = \dot{\psi}$) and the pitch motion is not allowed since the bicycle frame is rigid and the ground is flat ($\omega_y = 0$), (3.5) becomes:

$$L_x = J_{xx}\,\dot{\varphi} + J_{xz}\,\dot{\psi} \tag{3.6}$$

The components of inertia J_{xx} and J_{xz} are approximated as:

$$\begin{aligned} J_{xx} &= m\,h^2 \\ J_{xz} &= -m\,h\,l_r \end{aligned} \tag{3.7}$$

Considering the contact point of the rear wheel, the $\dot{\psi}$ can be defined as:

$$\dot{\psi} = \frac{v}{l_r + l_f}\,\tan\delta_e \tag{3.8}$$

Inserting (3.7) and (A.3) in (3.6):

$$L_x = m\,h^2\,\dot{\varphi} - m\,h\,l_r\,\frac{v}{l_r + l_f}\,\tan\delta_e \tag{3.9}$$

Assuming small angle approximation for both roll and steering angles ($\tan\delta \approx \delta$ and $\sin\varphi \approx \varphi$) and $\lambda = 90°$ ($\delta_e = \delta$), (3.4) and (3.9) become:

$$\frac{dL_x}{dt} = m\,g\,h\,\varphi + m\,\frac{(v^2\,h - g\,l_r\,c)}{l_r + l_f}\,\delta \tag{3.10}$$

$$L_x = m\,h^2\,\dot{\varphi} - \frac{m\,v\,h\,l_r}{l_r + l_f}\,\delta \tag{3.11}$$

Inserting (3.11) in (3.10) and making some operations, an expression for the derivative of the roll rate is obtained:

$$\ddot{\varphi} - \frac{g}{h}\,\varphi = \frac{l_r\,v}{(l_r + l_f)\,h}\,\dot{\delta} + \left(\frac{v^2}{h} - \frac{l_r\,g\,c}{(l_r + l_f)\,h^2}\right)\delta \tag{3.12}$$

The (3.12) represents the differential equation of $\dot{\varphi}$ (x_5) and completes the set of equations needed for the state space model.

3.3 State Space model

In correspondence with this state vector shown in Section 1.1 , the state space model is derived from (3.12) and substituting (3.2) in (3.1) :

$$\begin{cases} \dot{x}_1 = x_7 \cos\left(x_3 + \tan^{-1}\left(\frac{l_r}{(l_r+l_f)}\tan(x_6)\right)\right) \\ \dot{x}_2 = x_7 \sin\left(x_3 + \tan^{-1}\left(\frac{l_r}{(l_r+l_f)}\tan(x_6)\right)\right) \\ \dot{x}_3 = \frac{x_7}{l_r+l_f}\tan(x_6)\cos\left(\tan^{-1}\left(\frac{l_r}{l_r+l_f}\tan(x_6)\right)\right) \\ \dot{x}_4 = x_5 \\ \dot{x}_5 = \frac{g}{h}x_4 + \left(\frac{x_7^2}{h} - \frac{l_r g\, c}{h^2(l_r+l_f)}\right)x_6 + \frac{l_r x_7}{h(l_r+l_f)}u \\ \dot{x}_6 = u \\ \dot{x}_7 = 0 \end{cases} \tag{3.13}$$

where $u = \dot{\delta}_e$ denotes the input signal. This state space model captures the dynamics of the bicycle, providing a basis for the design of the Kalman filter.

4
Kalman filter

Given the state space model derived in the Section 3.3, a Kalman filter is designed as the state estimator of the system. This filter is capable of effectively estimating the true states of the system with high accuracy by using the model and the measurements provided by the sensors. Therefore, the theoretical foundations, linearization of the bicycle model, development of the measurement model, and step-by-step approach to designing and implementing the Kalman filter are discussed.

4.1 Theory

In this section, the relevant theory behind the Kalman filter is explained since it is the one used in this work. The theory is taken from [4], [5], [6] and [7] where also more deep motivations and descriptions can be found.

4.1.1 Estimation problem

Kalman filtering uses the system's model discretized in the time domain, the known control inputs, and multiple sequential measurements (from sensors) to produce an estimate of the state of the system. In many cases, the number of states in the system is bigger than the number of states that are directly accessible from the sensor data. These states are called hidden states. The objective of the filter is to combine the system's model and the available measurements to provide a reliable and accurate estimate of the current state.

It is also relevant to acknowledge that the model is not perfect, introducing uncertainty to the estimation, and the measurements are subjected to noise. The Kalman filter is capable of using a weighted average between the model's prediction and measurement. In this manner, values with a better (i.e., smaller) estimated uncertainty are trusted more. Consequently, the new estimate falls between the prediction and measurement, offering an improved uncertainty compared to either one alone.

4.1.2 Recursive algorithm

The Kalman filter is a recursive and iterative algorithm that enables the estimation of the state of the system over time.

The algorithm can be summarized in the following steps:

1) Initialization

The algorithm needs an initial condition. The accuracy of the initialization plays a key role in the performance, converging behavior, and robustness of the filter. This means that an accurate initialization will lead to a faster convergence, ensure stability, and better handling of noise or measurement anomalies. Thereby, these initial values for the state variables are usually set by prior knowledge or by an initial measurement.

2) Prediction step

The prediction step is usually known as the time update. The system's model is used to predict how the system evolves through time based on the previous state and control inputs. In matrix form:

$$\begin{aligned} \hat{x}(t|t-1) &= A\,\hat{x}(t-1|t-1) + B\,u(t) \\ \hat{P}(t|t-1) &= A\,\hat{P}(t-1|t-1)\,A^T + Q \end{aligned} \tag{4.1}$$

where:

- A and B are the state space model matrices.
- $\hat{x}(t|t-1)$ is the predicted state at time t.
- $\hat{x}(t-1|t-1)$ is the estimated state at time $t-1$.
- $\hat{P}(t|t-1)$ is the predicted state at time t.
- $\hat{P}(t-1|t-1)$ is the estimated state at time $t-1$.
- $u(t)$ is the current control input.
- Q is the process noise covariance matrix.

3) Update step

Using the predicted state as a starting point, the update step consists of using the measurements to refine the estimation.

To carry out this step, it is necessary to establish the measurement model that will be in charge of estimating the value of the measurements using the state variables. The linear version of this model is:

$$\hat{y}(t|t-1) = C\,\hat{x}(t|t-1) + D\,u(t) \tag{4.2}$$

where:

- C and D are the measurement model in matrix form.
- $\hat{y}(t|t-1)$ is the predicted output vector at time t based on the previous state of the system.

The measurement residual, also known as innovation, shows the discrepancy between the measurements given by the sensors and the predicted values:

$$\tilde{y}(t) = y(t) - \hat{y}(t|t-1) \tag{4.3}$$

The Kalman gain is thereby obtained using the discrete-time algebraic Riccati equation. It determines the weight given to the predicted state and the current measurement.

$$A^T X A - X - (A^T X C)(C^T X C + R)^{-1}(A^T X C)^T + Q = 0 \tag{4.4}$$

where:

- X is the unique stabilizing solution of the discrete-time algebraic Riccati equation.
- Q is the process noise covariance matrix.
- R is the measurement noise covariance matrix.

From which the state-feedback gain (Kalman gain) is computed:

$$K = (C^T X C + R)^{-1} (C^T X A) \tag{4.5}$$

Finally, the following equations are used to estimate the system's state and its covariance.

$$\begin{aligned} \hat{x}(t|t) =& \hat{x}(t|t-1) + K\,\tilde{y}(t) \\ \hat{P}(t|t) =& (I - K\,C)\,\hat{P}(t|t-1) \end{aligned} \tag{4.6}$$

To summarize, the relevant equations used in the design of the Kalman filter have been presented.

4.2 Linearization

The derived model in Chapter 3 is nonlinear. Consequently, the model must be linearized to be integrated into the Kalman filter.

As a first approximation, it is assumed that the velocity of the bicycle changes much more slowly than the other states, which allows it to be considered constant. However, the velocity is also a state x_7. Therefore, it is also estimated.

To linearize the kinematic equations (3.1), small angle approximation is assumed in δ, leading to the following linear approximations:

$$\tan(\delta_e) \approx \quad \delta_e \tag{4.7}$$

$$\tan^{-1}\Big(\frac{l_r}{(l_r + l_f)}\tan(\delta_e)\Big) \approx \quad \tfrac{l_r}{(l_r+l_f)}\delta_e \tag{4.8}$$

Finally, if $x_3 = \psi$ is small, the model (3.13) could be linearized with respect to x_3. To this end, a local frame is introduced to facilitate the linearization of the model. The local coordinate change is related to the global coordinate frame through a simple rotation of ψ_t around the z-axis. In this frame, the heading of the bike will

always be aligned with the x-axis ($\psi_L = 0$). This transformation is described by the following relation and it is illustrated in Figure 4.1:

$$\begin{bmatrix} X_L \\ Y_L \end{bmatrix} = \begin{bmatrix} \cos(\psi) & \sin(\psi) \\ -\sin(\psi) & \cos(\psi) \end{bmatrix} \begin{bmatrix} X_G \\ Y_G \end{bmatrix} \tag{4.9}$$

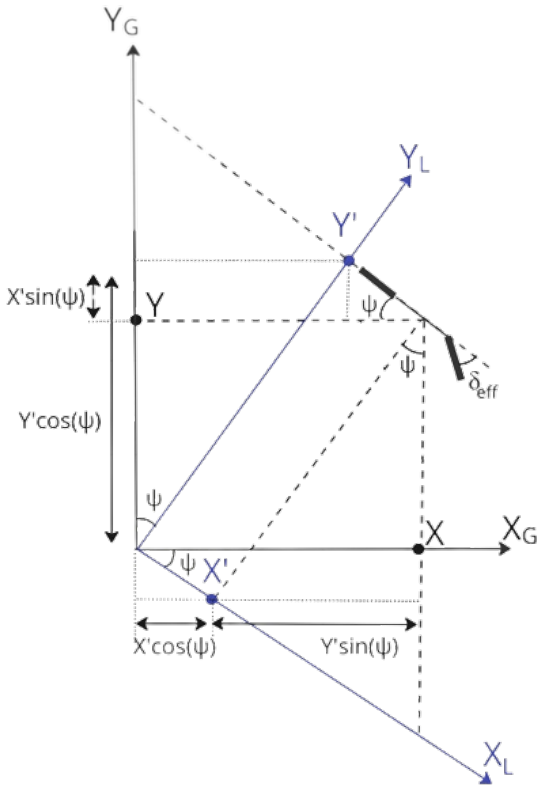


Figure 4.1: The local frame (X_L, Y_L) with respect to the global frame(X_G, Y_G)

Applying these three assumptions to (3.13) yields the linear model:

$$\begin{cases} \dot{x}_1^l = x_7 \\ \dot{x}_2^l = v\,(x_3^l + \frac{l_r}{(l_r+l_f)} x_6) \\ \dot{x}_3^l = \frac{v}{(l_r+l_f)} x_6 \\ \dot{x}_4 = x_5 \\ \dot{x}_5 = \frac{g}{h} x_4 + \left(\frac{v^2}{h} - \frac{l_r g\, c}{h^2(l_r+l_f)}\right) x_6 + \frac{l_r v}{h(l_r+l_f)} u \\ \dot{x}_6 = u \\ \dot{x}_7 = 0 \end{cases} \tag{4.10}$$

where x_1^l, x_2^l, and x_3^l represent the local versions of the respective states, while the other states remain the same.

Finally, the linear state space model is presented in matrix form for use in the state estimator:

$$\dot{\hat{x}}^l(t) = A\,\hat{x}^l(t) + B\,u(t) \tag{4.11}$$

where:

$$A = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & v & 0 & 0 & \frac{v\,l_r \sin\lambda}{(l_f+l_r)} & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{v \sin\lambda}{(l_r+l_f)} & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & \frac{g}{h} & 0 & \left(\frac{v^2}{h} - \frac{l_r\,g\,c}{h^2(l_r+l_f)}\right)\sin\lambda & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$B^T = \begin{bmatrix} 0 & 0 & 0 & 0 & \frac{l_r v}{h(l_r+l_f)} & 1 & 0 \end{bmatrix}$$

When a model is linearized, the approximations deteriorate as one moves farther from the operating point. However, it is anticipated that the steering angle generally remains below 10°, and the bicycle's velocity is expected to be constant. Consequently, it can be deduced that the linearization is precise.

4.3 Time discretization

The Kalman filter is run by a digital microprocessor. Therefore, the state space model (4.11) is discretized. The time update is computed:

$$\hat{x}(t+1) = \hat{x}(t) + (A\,\hat{x}(t) + B\,u(t))\,T_s \tag{4.12}$$

where T_s is the sampling time of the system. In this context, t does not represent actual time but rather signifies the number of discrete time steps or iterations taken. For instance, $t+1$ refers to the next iteration.

Developing (4.12), the discretized matrices (A_d and B_d) are obtained:

$$\begin{aligned} \hat{x}(t+1) &= \hat{x}(t) + A\,T_s\,\hat{x}(t) + B\,T_s\,u(t) \\ \hat{x}(t+1) &= \underbrace{(I + A\,T_s)}_{A_d}\,\hat{x}(t) + \underbrace{B\,T_s}_{B_d}\,u(t) \\ \hat{x}(t+1) &= A_d\,\hat{x}(t) + B_d\,u(t) \end{aligned} \tag{4.13}$$

The T_s is a variable to be selected by the user. In this work, the algorithm is running at 100 Hz. The dynamics of the bicycle do not require such a fast sampling rate. However, to have precise controllers, a highly accurate estimation is aimed. Thus, a faster estimator is more suitable.

4.4 Measurement model

The available measurements are described in the Section 2.2. However, not all measurements from these sensors are used. Instead, a specific set of measurements, defined as a vector, is chosen:

$$y^T = \left[X_{GPS}, Y_{GPS}, a_y, \omega_x, \omega_z, \delta_{ENC}, v\right] \tag{4.14}$$

From the states, a prediction of the measurements is carried out. The measurement equations from [1] used for this prediction are as follows:

$$\begin{cases} X_{GPS}^l = X_L \\ Y_{GPS}^l = Y_L \\ a_y = -h_{IMU}\, G_1 + G_2\, \cos\varphi - g\, \sin\varphi \\ \omega_x = \dot{\varphi} \\ \omega_z = \frac{v}{(l_r + l_f)} \tan\left(\delta\, \sin\lambda\right) \cos\varphi \\ \delta_{ENC} = \delta \\ v = v \end{cases} \tag{4.15}$$

where:

$$\begin{aligned} G_1 &= \frac{g}{h}\varphi + \frac{(h\, v^2 - g\, l_r\, c)\sin\lambda}{(l_r + l_f)\, h^2}\, \delta + \frac{l_r\, v \sin\lambda}{(l_r + l_f)\, h}\, \dot{\delta} \\ G_2 &= \frac{v^2}{(l_r + l_f)^2} \tan^2\left(\delta \sin\lambda\right) \sqrt{(l_r + l_f)^2 \cot^2\left(\delta \sin\lambda\right) + d^2} \end{aligned} \tag{4.16}$$

Assuming small angles for φ, $\dot{\varphi}$, and δ, and neglecting the d^2 term ($\cot^2(\delta\ \sin\ \lambda) >> d^2$) as well as the constant v, we can linearize the expressions in (4.16) obtaining the following:

$$\begin{aligned} G_1 &= \frac{g}{h}\, \varphi + \frac{(h\, v^2 - g\, l_r\, c)\sin\lambda}{(l_r + l_f)\, h^2}\, \delta + \frac{l_r\, v \sin\lambda}{(l_r + l_f)\, h}\, \dot{\delta} \\ G_2 &= \frac{v^2}{(l_r + l_f)}\, \delta \end{aligned} \tag{4.17}$$

Similarly to (4.11), the measurement equations are presented in matrix form:

$$\hat{y}^l(t) = C\, \hat{x}^l(t) + D\, u(t) \tag{4.18}$$

where:

$$C = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -g - \frac{h_{imu}\, g}{h} & 0 & \frac{-h_{imu}(h\, v^2 - g\, l_r\, c)\sin\lambda}{h^2\,(l_r+l_f)} + \frac{v^2\, sin\lambda}{(l_r+l_f)} & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{v\sin\lambda}{(l_r+l_f)} & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.19}$$

$$D^T = \begin{bmatrix} 0 & 0 & \frac{-h_{imu}\, l_r\, v}{(l_r+l_f)\, h} & 0 & 0 & 0 & 0 \end{bmatrix}$$

4.4.1 Different sampling rates of the sensors

The IMU, encoder, and velocity controller operate at a 100Hz sampling rate (same as the Kalman filter), while the GPS sensor operates at 10Hz. Consequently, GPS measurements are available only once every 10 iterations.

To deal with the different sampling rates, two situations arise:

a) Including GPS measurements. This scenario happens in 1/10 iterations. The matrices from (4.19) are used.

b) Excluding GPS measurements. This scenario happens in the rest of the iterations (in 9/10). The GPS measurements are not available. Therefore, the states (x_1^l, x_2^l, x_3^l) are not updated and a subset of measurements is used:

$$y_{sub}^T = \begin{bmatrix} a_y, \omega_x, \omega_z, \delta_{ENC}, v \end{bmatrix}. \tag{4.20}$$

Consequently, submatrices from the (4.19) are obtained by eliminating the first two rows, which are related to the unavailable GPS measurements, and the first three columns, which are associated with the three not-updated states since they are independent of the subset (4.20) :

$$C_{sub} = \begin{bmatrix} -g - \frac{h_{imu}\, g}{h} & 0 & \frac{-h_{imu}(h\, v^2 - g\, l_r\, c)\sin\lambda}{h^2\,(l_r+l_f)} + \frac{v^2\, sin\lambda}{(l_r+l_f)} & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{v\sin\lambda}{(l_r+l_f)} & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.21}$$

$$D_{sub}^T = \begin{bmatrix} \frac{-h_{imu}\, l_r\, v}{(l_r+l_f)\, h} & 0 & 0 & 0 & 0 \end{bmatrix}$$

4.5 Tuning principle of Q_K and R_K

The Q_K and R_K matrices represent the uncertainty of the system's state evolution over time and of the sensor's measurements respectively. Tuning these matrices appropriately is essential for the Kalman filter's performance. They are used to compute the K (see Section 4.1.2).

The matrices Q_K and R_K determine the weight that is assigned to the prediction and measurement, i.e., how much the model and the measurements are trusted. A high value of Q_K means that the low confidence in the model (high uncertainty) will increase the K values to give more weight to the measurement update. On the other hand, a low value of Q_K means that the model is accurate and the prediction will be relied on more (low values of K). In the case of R_K the tuning is equivalent but related to the accuracy of the measurements.

The process of tuning implies identifying the optimal balance point between trusting the prediction and relying on the measurements. This balance is of importance as it directly affects the accuracy and stability of the state estimates produced by the filter. Identifying the appropriate balance point ensures that the filter avoids extremes: it neither reacts excessively to every measurement, leading to noisy estimates, nor sticks too much to its predictions, potentially overlooking significant changes. Instead, the filter achieves a balance combining the advantages of both predictions and measurements to deliver state estimates that are stable and accurate.

4.6 Implementation

This section outlines the steps involved in implementing the bike system's Kalman filter. These steps are divided into two groups: offline and online. The offline steps are computed in MATLAB to obtain the necessary parameters for the algorithm. The algorithm consists of the online steps that are iteratively run in Simulink (simulation) or LabVIEW (real world).

4.6.1 Offline steps

The offline steps consist of time discretization and computation of the Kalman gain.

4.6.1.1 Time discretization

The system is discretized as explained in Section 4.3.

4.6.1.2 Computation of the Kalman gain

The Kalman gain is obtained using (4.5). In the Section 4.4.1 is mentioned that due to the different sampling times of the sensors, two measurement models appear

where (4.21) (C_{sub}) are just submatrices of (4.19) (C).

However, it is not necessary to obtain two Kalman gains. When you perform the same operation with submatrices extracted from larger matrices, the result of the second operation will always be a submatrix of the result of the first operation, provided the dimensions match correctly. Therefore, it is sufficient to obtain the Kalman gain using C.

4.6.2 Online steps

The Figure 4.2 shows the inputs, output, and parameters (obtained from the offline steps) and lists the online steps.

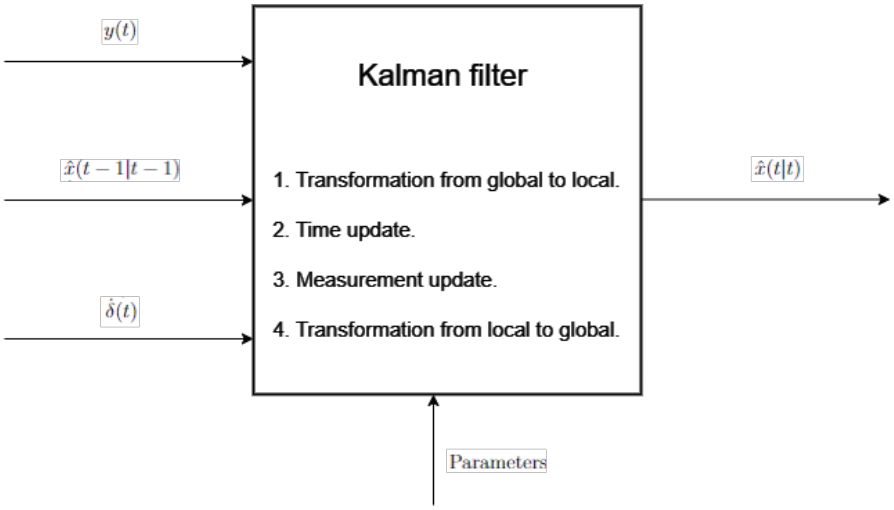


Figure 4.2: Kalman filter overview. The inputs, output, parameters, and the online steps are shown. The parameters are obtained from the offline steps.

- Inputs:
  1. $\hat{x}(t-1|t-1)$ is the estimated states at the previous time.
  2. $\dot{\delta}(t)$ is the control input at time t.
  3. $y(t)$ is the measurement vector of the current time.
- Outputs:
  1. $\hat{x}(t|t)$ is the estimated states at the current time.
- Parameters:
  1. Kalman gain (K).
  2. Model matrices: A_d, B_d, C_d and D_d.

The algorithm that is run iteratively takes the steps:

1. Transformation to the local frame of the $\hat{x}(t-1|t-1)$ and $y(t)$ using (4.9):

$$\begin{cases} \begin{bmatrix} \hat{x_1}^l(t-1|t-1) \\ \hat{x_2}^l(t-1|t-1) \end{bmatrix} = \begin{bmatrix} \cos(\hat{x_3}(t-1|t-1)) & \sin(\hat{x_3}(t-1|t-1)) \\ -\sin(\hat{x_3}(t-1|t-1)) & \cos(\hat{x_3}(t-1|t-1)) \end{bmatrix} \begin{bmatrix} \hat{x_1}(t-1|t-1) \\ \hat{x_2}(t-1|t-1) \end{bmatrix} \\ \hat{x_3}^l(t-1|t-1) = 0 \\ \\ \begin{bmatrix} y_1^l(t) \\ y_2^l(t) \end{bmatrix} = \begin{bmatrix} \cos(\hat{x_3}(t-1|t-1)) & \sin(\hat{x_3}(t-1|t-1)) \\ -\sin(\hat{x_3}(t-1|t-1)) & \cos(\hat{x_3}(t-1|t-1)) \end{bmatrix} \begin{bmatrix} y_1(t) \\ y_2(t) \end{bmatrix} \end{cases}$$

2. Prediction step.

$$\hat{x}^l(t|t-1) = A_d\,\hat{x}^l(t-1|t-1) + B_d\,u(t) \tag{4.22}$$

3. Measurement update. Two situations arise (see Section 4.4.1):

   a) Including GPS measurements. The equations (4.2), (4.3), and (4.6) are used with the matrices (4.19) and (4.5).

   b) Excluding GPS measurements. The same equations (4.2), (4.3) and (4.6) are used but with the submatrices (4.21) and K_{sub} (obtained performing the same eliminations as in Section 4.4.1 to obtain C_{sub}):

$$\begin{aligned} x_1^l(t|t) &= x_1^l(t|t-1) \\ x_2^l(t|t) &= x_2^l(t|t-1) \\ x_3^l(t|t) &= x_3^l(t|t-1) \\ \hat{x}_{sub}^l(t|t) &= \hat{x}_{sub}^l(t|t-1) + K_{sub}\,\tilde{y}_{sub}^l(t) \end{aligned} \tag{4.23}$$

4. Transformation back to the global coordinate system using the inverse transformation of (4.9):

$$\begin{cases} \hat{x_1}(t|t) = \hat{x_1}^l(t|t)\cos(\hat{x_3}(t-1|t-1)) - \hat{x_2}^l(t|t)\sin(\hat{x_3}(t-1|t-1)) \\ \hat{x_2}(t|t) = \hat{x_1}^l(t|t)\sin(\hat{x_3}(t-1|t-1)) + \hat{x_2}^l(t|t)\cos(\hat{x_3}(t-1|t-1)) \\ \hat{x_3}(t|t) = \hat{x_3}(t-1|t-1) + \hat{x_3}^l(t|t) \end{cases}$$

4.7 Validation

The implemented Kalman filter has been integrated into a Simulink-based simulation model, enabling its validation. The Figure 4.3, shows the schematic used for the validation.

The filter is validated in the simulation where estimated states can be compared with the true ones in different scenarios of disturbances. These scenarios are ideal sensors (no disturbances) and noisy measurements. Finally, a comparison between the Simulink and LabVIEW versions of KF to check the transformation to C-code.

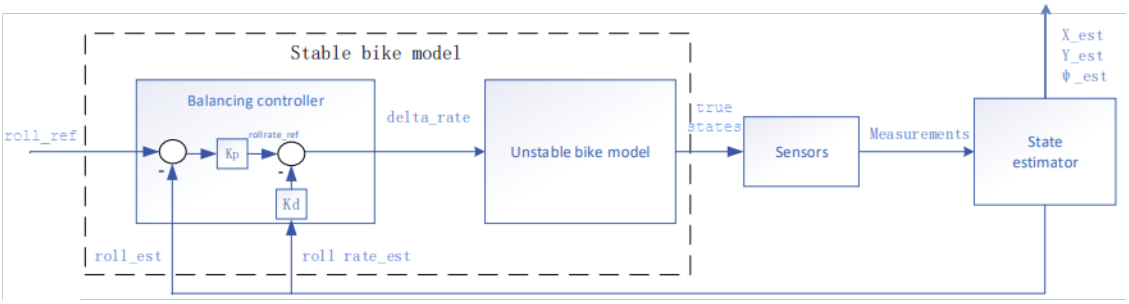


Figure 4.3: Kalman filter validation schematic in Simulink.

The input (roll_ref) used to evaluate the KF is shown in Figure 4.4. This signal is manually generated in Simulink to cover some behaviors of interest such as a step change or a ramp. When a constant value of 0.075 radians is set, the bike will produce a constant radius circle.

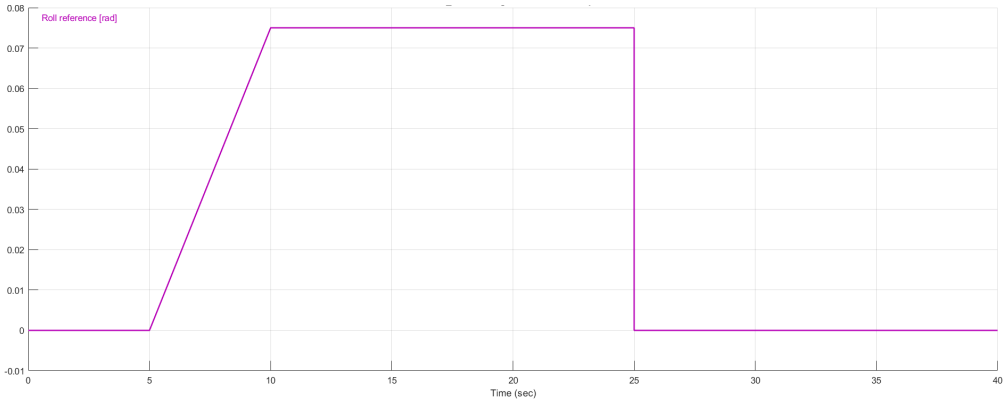


Figure 4.4: Roll reference signal. Input for the validation process.

4.7.1 Ideal sensors

To show the effect of tuning Q_K and R_K, a noise-free simulation is performed. The Table 4.1 presents two different tunings that are used in the ideal sensor scenario. The "standard" tuning is just a reference that consists of two identity matrices. For the "Good" tuning, the principle of Section 4.5 is used by trusting more the measurements since they are noise-free in the ideal scenario. These tunings are far from being optimal but they are chosen on purpose just to show the effect of tuning in the estimations.

	Q_K	R_K
Standard	I_{7x7}	I_{7x7}
Good	I_{7x7}	$0.001 \cdot I_{7x7}$

Table 4.1: Two different tunings of Q_K and R_K for the ideal sensor scenario.

Given the roll_ref (see Figure 4.4), the Figure 4.5 shows the performed trajectory by the bicycle. Three signals are plotted: true (simulated bike trajectory), mea-

surements (simulated GPS), and estimated (Kalman filter's estimation). The three signals cannot be distinguished since they are on top of each other meaning that the differences between them are very small or zero.

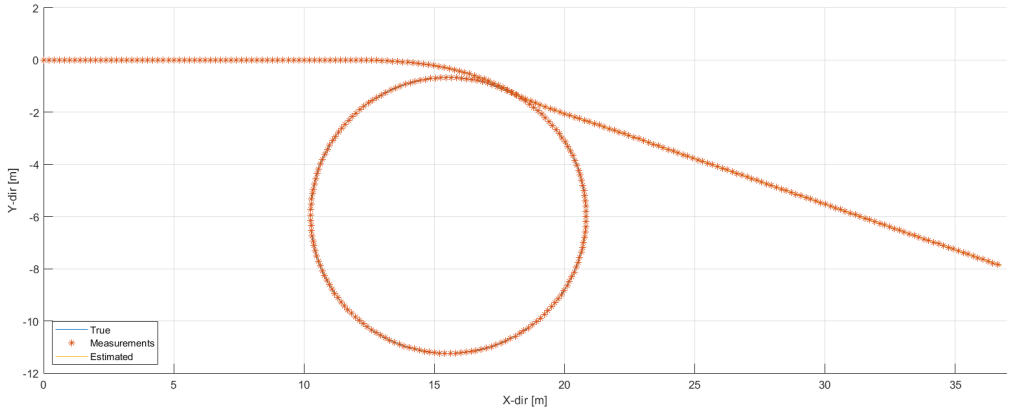


Figure 4.5: True and estimated paths together with the simulated GPS measurements under the standard tuning case in the ideal sensor scenario (Table 4.1). The signals cannot be distinguished since they are on top of each other.

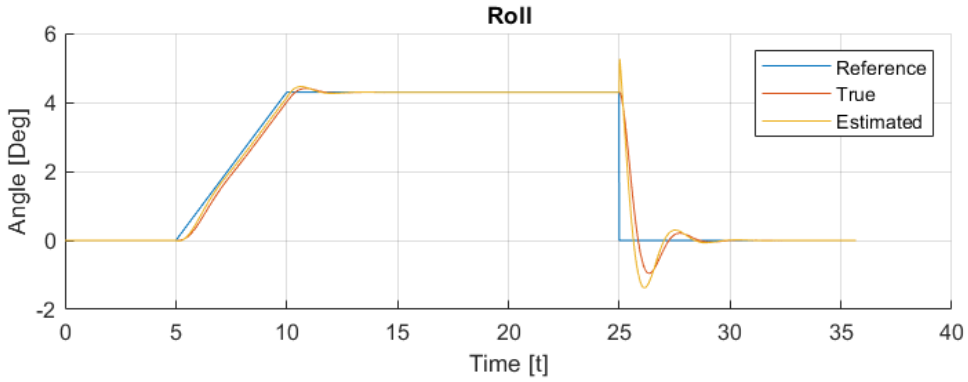
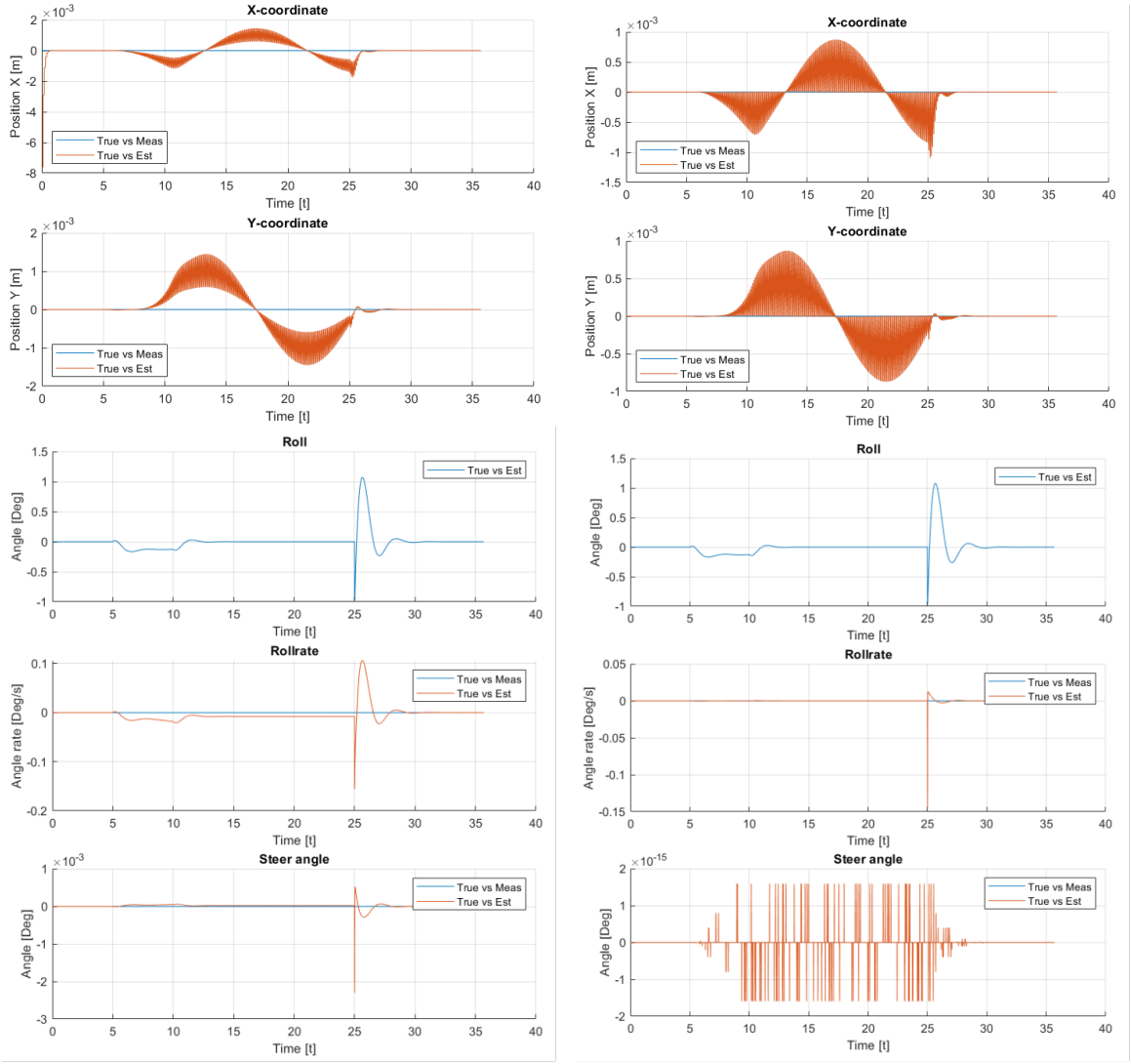


Figure 4.6: Reference, true and estimated roll signals with standard tuning (Table 4.1). The balancing controller (reference vs. true) can track the reference roll angle. Additionally, the Kalman filter (true vs. estimated) can estimate the true roll angle of the bicycle accurately (almost on top of the true signal even with a step change).

The errors between the true-measured states and true-estimated signals are shown in Figure 4.7 where the two tuning cases of Table 4.1 are compared. Some observations:

- The true vs. measurement signal is always at zero since the no-noise sensor scenario is shown.
- In both cases the errors can be considered small. However, from the scale of the plots, it can be observed that the tuned case reduces the errors considerably at this level.

- The spikes in the X and Y signals are due to the 10 Hz sampling. The model is not as accurate as the measurements. Therefore, when the prediction is only used, the error increases. When the measurement is used to correct the prediction (1/10 times), the error is decreased in the standard case and is zero in the tuned case.
- The spikes in the steer angle plot are due to computational errors since the scale is at 10^{-15} meaning that the tuned case is still better than the standard one.



(a) Standard ($Q_K = R_K = I_{7x7}$) **(b)** Tuned ($Q_K = I_{7x7}$, $R_K = 0.001 \cdot I_{7x7}$)

Figure 4.7: Errors between true-meas and true-est signals, using the Table 4.1 tunings, are compared. The true-meas signal is always zero since the measurements are ideal (no noise). The figure illustrates how using the Section 4.5, the quality of the estimations is improved. Since ideal sensors are used, the improvement is not very big but it can be seen from the scale of the plots.

4.7.2 Noisy sensors

In this subsection, the filter's performance under noisy measurements is analyzed. For this analysis, the estimation accuracy is compared under different tunings (see Table 4.2). White noise (independent and identically distributed with a constant variance)is used which is not realistic. However, it can be used to show the effect of tuning. It is only applied to the GPS measurements.

The "standard" tuning is the same as in Section 4.7.1 and it is used as a reference again. The "Good" tuning consists of increasing the diagonal values related to the GPS measurements of the R_K since they are noisy now, following the described principle of the Section 4.5. In the case of "Bad" tuning, the opposite is done by reducing the values instead. Again these tunings are chosen on purpose just to show the effect of tuning in the estimations, i.e., the "Good" tuning is far from being optimal.

	Q_K	R_K
Bad	I_{7x7}	$diag(0.001, 0.001, 1, 1, 1, 1, 1)$
Standard	I_{7x7}	I_{7x7}
Good	I_{7x7}	$diag(1000, 1000, 1, 1, 1, 1, 1)$

Table 4.2: Three different tunings of Q_K and R_K for the noisy sensor scenario.

The Figure 4.8 depicts the performed trajectory by the bicycle, given the roll reference of Figure 4.4. Unlike in Section 4.7.1, the signals are not on top of each other due to the added noise to the GPS measurements. Some observations from the Figure:

- In the case of "Bad" tuning, when the new measurement arrives, it can be observed that the estimation always takes the value of the noisy measurement or a close one. This leads to a noisy and inaccurate estimation with a clear error when compared to the true signal.
- In the "Standard" case, the estimation is still a noisy signal. However, it can be observed how usually the estimation does not overlap with the measurement. The prediction from the model is taking more into account but still, the filter relies too much on the noisy measurement. Thus, the estimation is improved but is far from being enough accurate.
- In the last case, a much smoother estimation is presented. The signal does not jump from one measurement to another. It can be seen that the estimation is much closer to the real trajectory than in the previous cases. The prior knowledge of the noisy GPS measurement allows tuning the filter so that it relies more on the model and does not trust the GPS.

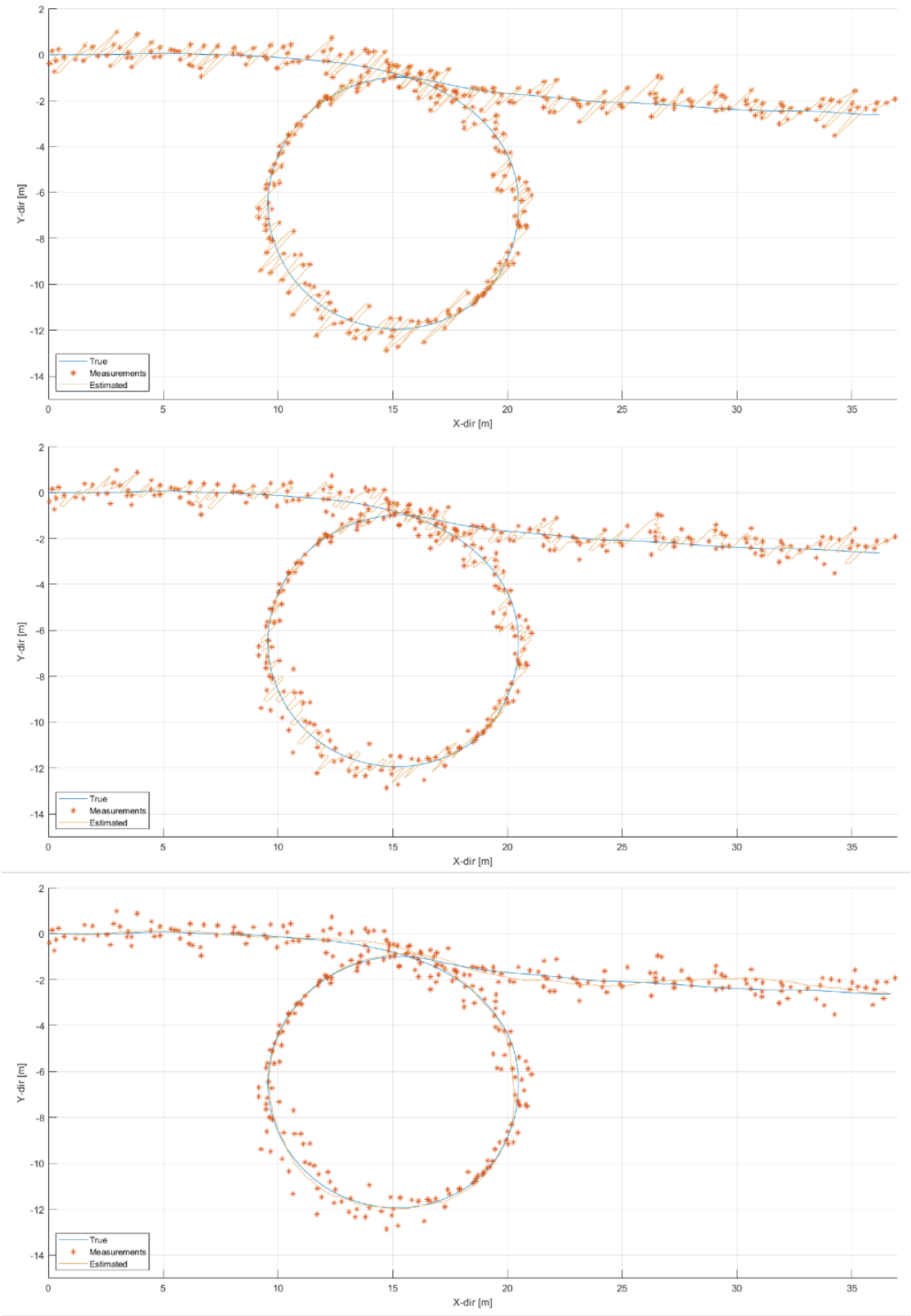


Figure 4.8: True and estimated paths together with the noisy GPS measurements for the three different tuning choices: Bad (1), standard (2), good (3) (see Table 4.2, the plots are in the same order). In (1) the estimation trusts the noisy measurement which leads to being inaccurate. In (3), the tuning choice makes the estimation rely more on the model leading to a smoother and accurate estimation. The (2) is an intermediate performance that is still noisy but more accurate than (1).

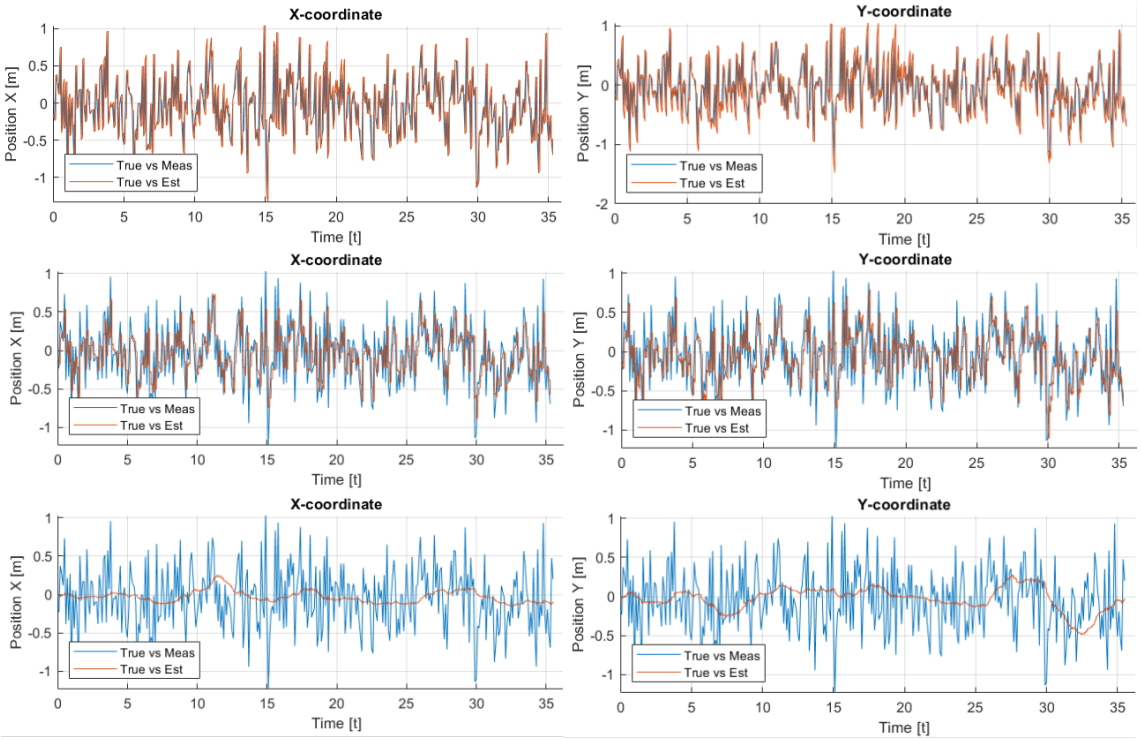


Figure 4.9: Comparison between X_G and Y_G error signals (true-meas and true-est) for the different tunings in Table 4.2: Bad (1), Standard (2), Good (3). The true-meas signals are the same in the three cases since the added noise is the same. In the "Bad" tuning case, the true-meas and true-est signals are almost identical (very noisy) since the filter is tuned so that the estimations rely a lot on the noisy measurements. In the "Good" tuning case, it can be seen that the true-est is a much smoother signal, while it also remains close to zero error since the filter relies more on the model than on the noisy measurements. The "Standard" tuning produces smaller errors than the "Bad" case but still are noisy signals since it relies equally on the model and the noisy measurements. Therefore, this figure supports the observations from Figure 4.8.

4.7.3 Validation of Kalman filter in the real bicycle

The software in the real bicycle is run by LabVIEW. The Kalman filter algorithm is transformed to C-code to be used in LabVIEW. This way, this transformation and the filter's performance in the real bicycle are validated.

A test on the real bicycle is performed. The real test aims to check the performance of the bike trying to follow a constant zero roll angle reference. Therefore, the bike should move straight. The data from that test is logged and saved in a file that is later used in MATLAB. The file is loaded in *Kalman_offline.m* and simulated in the offline KF (*Kalman_offline_sim.slx* in Simulink, see schematic in Figure 4.10). This way, the inputs of the offline KF are the data from the real bike test that are used to generate an offline estimation. The C (online KF) and the Simulink version (offline KF) estimations are plotted together to be compared.

Figure 4.10: *Kalman_offline_sim.slx* file schematic. Given the real bike test data (inputs), it generates an offline estimation. This way, online and offline KF can be compared.

To perform the real bike tests, the filter needs to be tuned correctly. Considering the tuning principle from the Section 4.5, a short explanation is described. To determine the R_K matrix, the real bike is used to evaluate the variances of sensor measurements, estimations, and their discrepancies. The diagonal elements of matrix R_K are set as the variances of each sensor's measurements while inter-sensor covariances are not considered (non-diagonal values are 0). For Q_K, trial and error method is used. It starts with an identity matrix and scale factors are applied based on the model knowledge until an accurate estimation is achieved. The obtained matrices are:

$$Q_K = diag(0.1, 0.1, 0.1, 10^{-9}, 5, 10, 0.5)$$
$$R_K = diag(1.5677, 1.5677, 0.2564, 3.94 \cdot 10^{-12}, 0.0234, 4.15 \cdot 10^{-5}, 0.1)$$

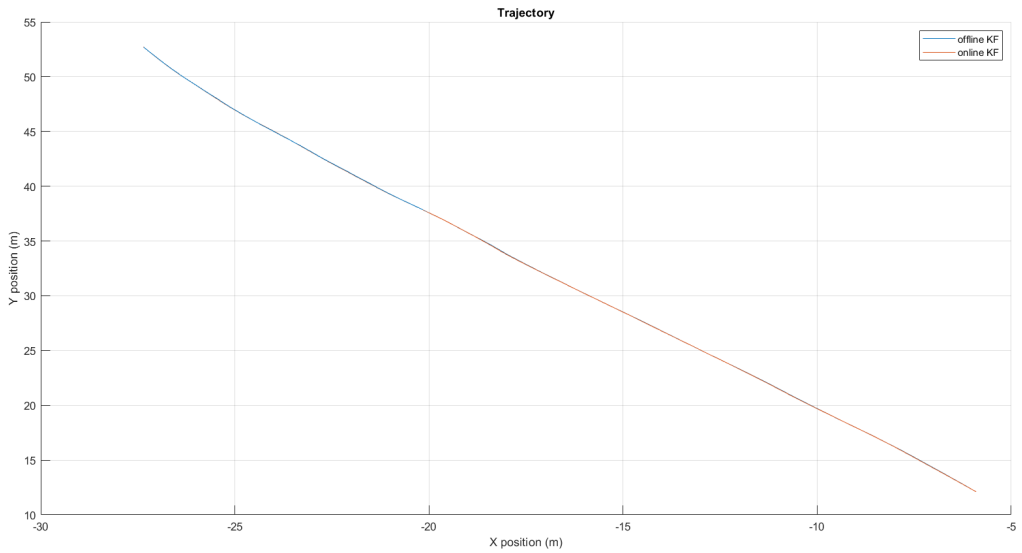


Figure 4.11: The estimated trajectories from online and offline KF. Both estimation signals are on top of each other, meaning that they are almost identical. In addition, it can be seen that the trajectory of the bike is almost straight as expected since it is tracking a constant zero roll angle reference.

The plot from the Figures 4.11, 4.12, and 4.13, starts approximately at 24 seconds. Upon this moment the steering motor of the bike is activated. Therefore, the bike starts to ride on its own. Previous data is not useful because human interaction is not desired.

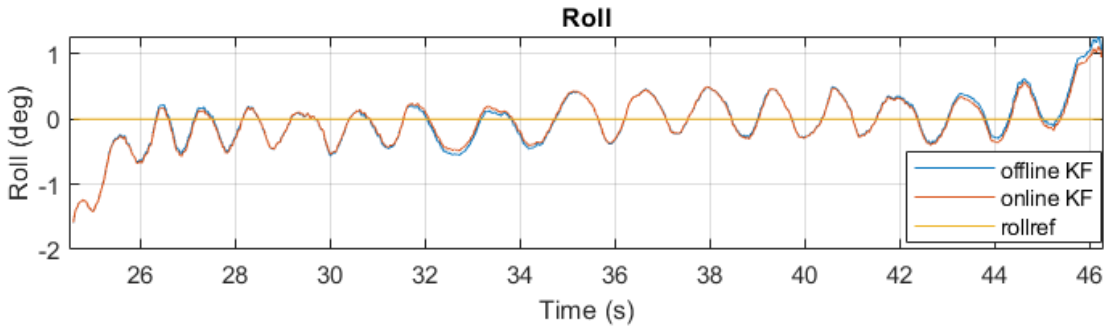


Figure 4.12: Roll reference comparison with online and offline KF roll estimations. It can be seen how the estimations wiggle around the constant zero roll reference. The oscillations are always between ±1 deg which is very small and almost unnoticeable on the bicycle as can be seen in Figure 4.11. These oscillations may be due to the road imperfections and the balancing controllers' small corrections to keep the bike straight.

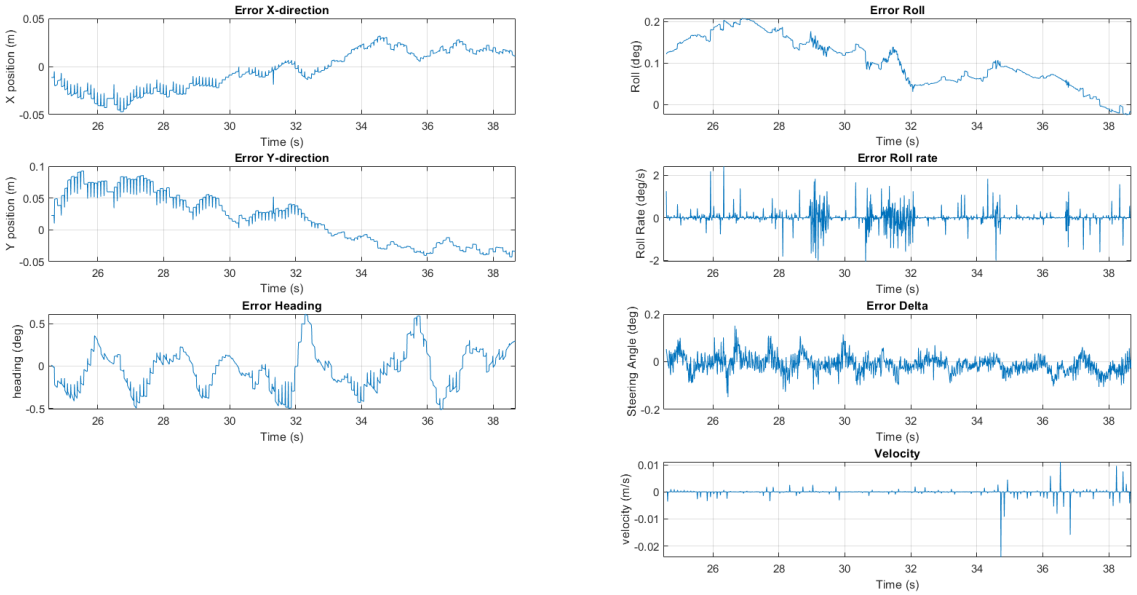


Figure 4.13: Error signals between online and offline KF estimations. This figure completes the validation of the transformation since both estimations are similar given the same data. The difference may be because the online KF operates in real-time while the offline KF works in a simulation environment.

From the Figures 4.11, 4.12, and 4.13, it can be seen that both estimations are sometimes overlapped and always very close. The online and offline KF are the same filter but coded in different environments. Therefore, the estimates, given the

same inputs, should be identical. However, there is a difference in the estimations because the online KF is working in real time. The simulation is an almost perfect environment where all work at the expected frequency. On the other hand, in real-time, many things can happen that delay the system. For example, real sensors are expected to sample measurements at a specific frequency but in the real world, this sampling is not perfect and may cause delays in the KF or even the data is not exactly logged. These examples can be observed in the Figure 4.14 that illustrates the time between each iteration of the KF and logging and between the measurements of the GPS. The differences between both estimations can be attributed to these examples or similar imperfections. Nevertheless, the online estimation is almost identical to the offline given the imperfections. Thus, it can be said that the online KF is validated.

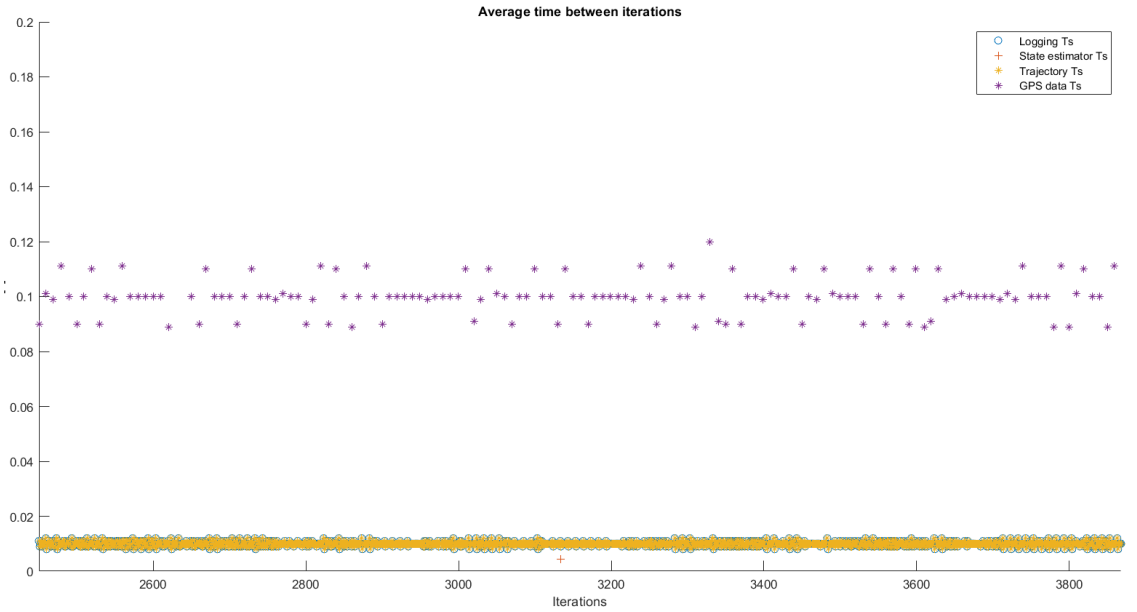


Figure 4.14: Different sample times in the real bike test. The figure shows that in the real bike, all the processes do not operate always at the same expected frequency. It explains the difference between the online and offline KF estimations.

5

Trajectory controller

The trajectory controller is responsible for guiding a system along a predefined path or trajectory. It takes into account the desired motion and the current states of the system to generate control inputs, such as direction commands. Considering the system dynamics and feedback signals, a well-designed trajectory controller enables precise and smooth motion.

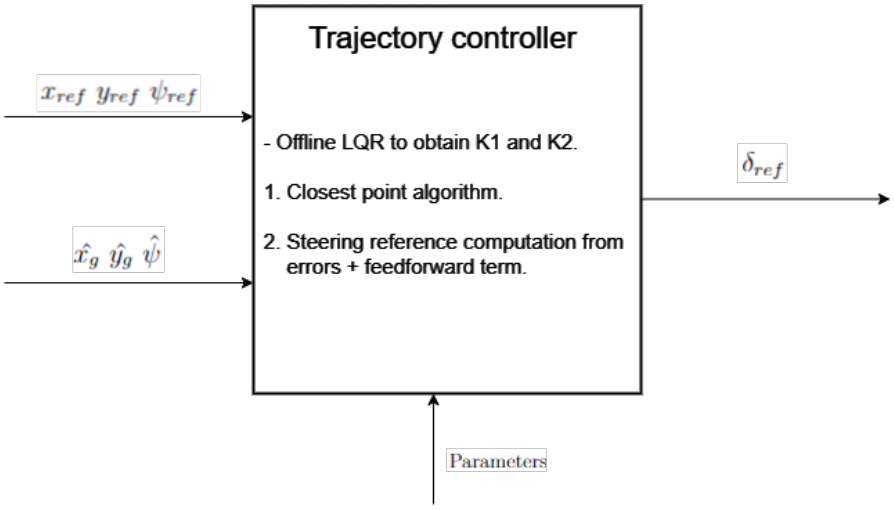


Figure 5.1: Trajectory controller overview.

An overview of the used trajectory controller is presented in Figure 5.1. The inputs and outputs of the algorithm are presented:

- Inputs:
  1. x_{ref}, y_{ref}, and ψ_{ref} are the global position and direction references, i.e., the predefined path to be followed.
  2. $\hat{x}$, $\hat{y}$, and $\hat{\psi}$ are the global position and direction estimations from the Kalman filter.
- Outputs:
  1. δ_{ref} is the steer reference that the bike needs to perform to follow the predefined path.
- Parameters:
  1. Bike parameters.
  2. K_1, K_2 and e_1^{max} are the tunable parameters of the LQR.

3. v is the forward velocity of the bike.

The different parts of the trajectory controller algorithm are explained in more detail throughout this chapter.

5.1 LQR

The Linear Quadratic Regulator (LQR) is a control strategy usually used in control systems and optimization. It is designed to stabilize and control linear time-invariant systems by finding the optimal control inputs that minimize a cost function. The LQR algorithm considers the system's state and control inputs.

This section will summarize the relevant information behind the lateral controller using an LQR from [3]. More deep calculations, motivations, and validations can be found there.

The kinematic model of a balanced bike is presented. It is obtained under the assumption of no wheel slip. In the Figure 5.2, relevant parameters are shown:

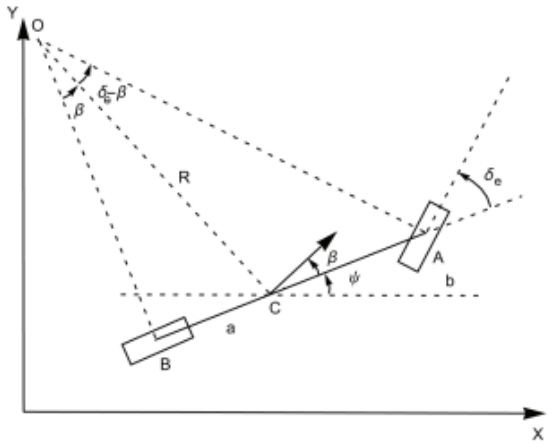


Figure 5.2: Top view of a balanced bike with its physical parameters.

From Figure 5.2, A is the contact point of the front wheel and B of the rear wheel. b $(l_r + l_f)$ is the distance between them. C is the center of mass, of distance a (l_r) from the rear wheel. With an steering angle δ_e, the bike makes a circle of center O and radius R.

Due to only being considered the lateral controller, v is considered as an input signal. Since $(l_r + l_f)$ is much smaller than R, small angles are considered for δ_e and β (body slip angle) to linearize the model. Thus, the model is:

$$
\begin{aligned}
\beta &= \frac{l_r}{l_r + l_f}\,\delta_e \\
x &= \begin{bmatrix} X & Y & \psi \end{bmatrix} \\
u &= \begin{bmatrix} v & \delta_e \end{bmatrix}^T \\
\dot{x} &= \begin{bmatrix} v\,\cos(\psi) \\ v\,\sin(\psi) \\ 0 \end{bmatrix} + \begin{bmatrix} -\frac{l_r}{l_r+l_f}\,v\,\sin(\psi) \\ \frac{l_r}{l_r+l_f}\,v\,\cos(\psi) \\ \frac{v}{l_r+l_f} \end{bmatrix} \delta_e
\end{aligned}
\tag{5.1}
$$

The LQR is used to control the lateral error. Thus, given the kinematics of the bike, the lateral dynamics of the bike can be described by the lateral error e_1 and heading error e_2. These errors are related to the closest point on the reference path to the bike shown in Figure 1.4. The expressions for these errors are:

$$
\begin{aligned}
e_1 &= \sqrt{(X - X_{closest})^2 + (Y - Y_{closest})^2} \\
e_2 &= \psi - \psi_{ref}
\end{aligned}
\tag{5.2}
$$

where X, Y, and ψ are the estimated position/direction of the bike that comes from the Kalman filter.

From the Figure 1.4 and model (5.1), the lateral dynamics are obtained. In state-space form:

$$
\begin{aligned}
e &= \begin{bmatrix} e_1 & e_2 \end{bmatrix}^T \\
\dot{e} &= \underbrace{\begin{bmatrix} 0 & v \\ 0 & 0 \end{bmatrix}}_{A} e + \underbrace{\begin{bmatrix} \frac{l_r\,v}{l_r+l_f} \\ \frac{v}{l_r+l_f} \end{bmatrix}}_{B} \delta_e + \underbrace{\begin{bmatrix} 0 \\ -1 \end{bmatrix}}_{D} \dot{\psi}_{ref}
\end{aligned}
\tag{5.3}
$$

where δ_e is the control input and $\dot{\psi}_{ref}$ is a known variable that is used in section 5.3.

To design the controller, assuming small control errors, a linear state space control law is used:

$$
\delta_e = -K_1\,e_1 - K_2\,e_2 \tag{5.4}
$$

where the parameters K_1 and K_2 are chosen using the LQR. The LQR minimizes a cost function to find the optimal control inputs. The chosen target function is:

$$
J = \int_0^\infty (e^T\,Q_T\,e + \delta^T\,R_T\,\delta)dt \tag{5.5}
$$

where Q_T and R_T are design matrices. The solution is:

$$
K = \begin{bmatrix} K_1 \\ K_2 \end{bmatrix} = R_T^{-1}\,B^T\,P \tag{5.6}
$$

and P is computed from the Riccati equation:

$$A^T P + P A - P B R_T^{-1} B^T P + Q_T = 0 \tag{5.7}$$

The derived linear controller is only valid in a small error range. Out of this range, a simple non-linear controller is defined. It is assumed that the bike starts with relatively small initial control deviations. Specifically, the e_2 is constrained to be no greater than 45º. Additionally, a maximum allowable e_2 value is established to prevent the bike from moving in the opposite direction or entering into circular motions. Thus,

$$e_2^{max} = \frac{\pi}{6}$$

For big lateral errors, i.e., significant magnitudes of e_1, the controller is designed to command $\delta_e = 0$ when the bike is moving toward the reference path at an angle of e_2^{max}. This is achieved by modifying the control law as follows:

$$\delta_e = -K_1 \, sign(e_1) \, min(|e_1|, e_1^{max}) - K_2 \, e_2 \tag{5.8}$$

Here, the value $\pm e_1^{max}$ indicates the range of e_1 within which the control law maintains linearity. Beyond this interval, the control law transitions into a non-linear state. To determine e_1^{max}, the equation (5.4) is employed where $\delta = 0$ and $e_2 = e_2^{max}$:

$$e_1^{max} = \frac{K_2 \, e_2^{max}}{K_1} \tag{5.9}$$

Validations and performance with different design choices for variables are detailed in [3].

5.2 Closest point algorithm

The closest point algorithm (CPA) is a computational technique whose objective is to find the closest point in a given dataset. Given the set of points in the path and the estimated position, the algorithm will output the index of the reference point that the bike should go to.

The designed algorithm has the following steps:

1. Search for the closest point (*cp*). The closest point in the predefined path is obtained by computing the distances from the bike's estimated position and the path's points. The loop starts by comparing the distance of the bike to the *cp* and the next point. It keeps iterating until the distance to the next point is bigger than the previous point. Then, the *cp* will be the previous point before the loop is stopped. In the following Figure 5.3, this step is illustrated:

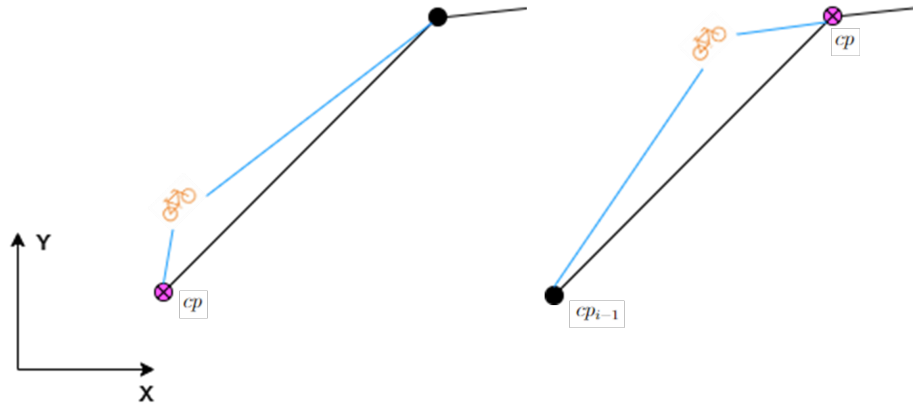


Figure 5.3: Two different situations where the pink point is the selected closest point.

Once the *cp* has been found, the distance in X (d_x) and Y (d_y) from the bike's current location to the selected point is computed. In addition, the heading difference between the *cp* and the next point is also obtained called $\dot{\psi}_{ref}$, and it is always wrapped between $[-\pi, \pi]$. These variables are later used.

2. Interpolation algorithm. This step is used to project the bicycle onto the reference path and detect whether it has passed the closest point. Detecting this is crucial because passing the *cp* does not automatically imply a change in the *cp* closest point itself. However, it does indicate the need to determine the next direction to avoid delays or overshoots in the path-tracking process. This phenomena is illustrated in Figure 5.4.

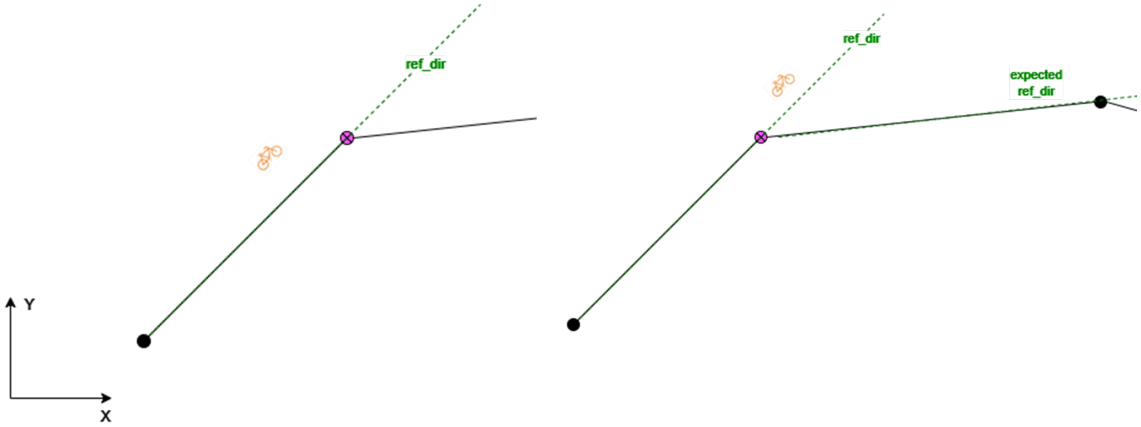


Figure 5.4: Situations when the bike does not surpass the closest point and when it does.

The interpolation algorithm calculates the distance between the point before the actual closest point (cp_{i-1}) and the bike, denominated d_{i-1}. After, the angle to project the bike onto the reference path can be computed:

$$\alpha^*_{proj} = \alpha^* - \psi_{ref}(cp_i) \tag{5.10}$$

where α^* is the heading of the bicycle.

To know if the bike has surpassed the closest point, two distances need to be compared:

$$\begin{aligned} d_{proj} &= |d_{i-1} \cos(\alpha^*_{proj})| \\ d_{cp,cp_{i-1}} &= \sqrt{(x_{ref}(cp_i) - x_{ref}(cp_i - 1))^2 + (y_{ref}(cp_i) - y_{ref}(cp_i - 1))^2} \end{aligned} \tag{5.11}$$

where the second distance is obtained by applying the Euclidean distance.

Therefore,

$$\text{if} \quad \begin{cases} d_{proj} \geq d_{cp,cp_{i-1}} & cp_{heading} = cp_i + 1 \\ d_{proj} < d_{cp,cp_{i-1}} & cp_{heading} = cp_i \end{cases}$$

$cp_{heading}$ indicates the index of the point that is taken as a reference for direction. In the Figure 5.5, all the variables used in the projection are described in the situation where the bike has surpassed the cp:

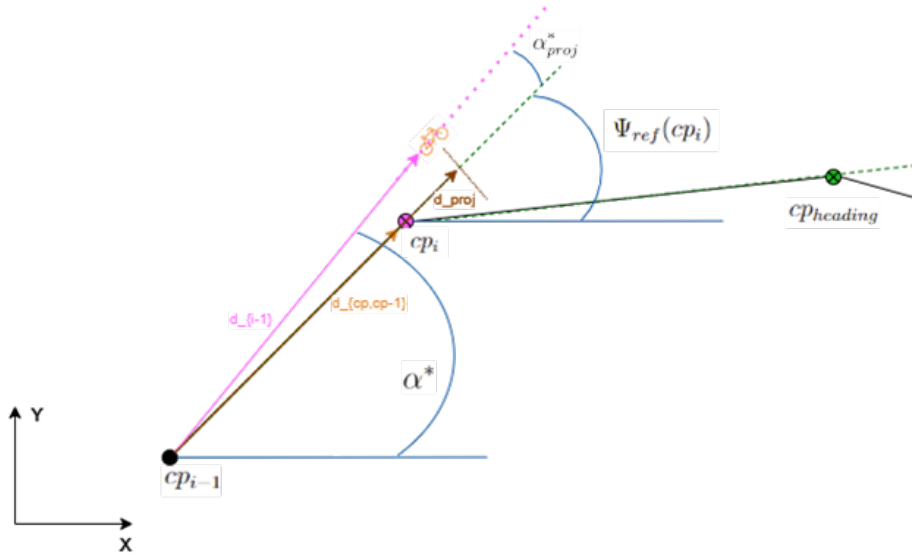


Figure 5.5: Illustration of the variables needed for the interpolation algorithm.

5.3 Steer reference calculation

After calculating the closest point, heading reference, and the values of $K1$ and $K2$, the steering reference for the bike to follow the path can be determined. The steering reference is derived as the sum of three different contributions: lateral error, heading error, and a feed-forward term. The first two terms are illustrated in the Figure 5.6:

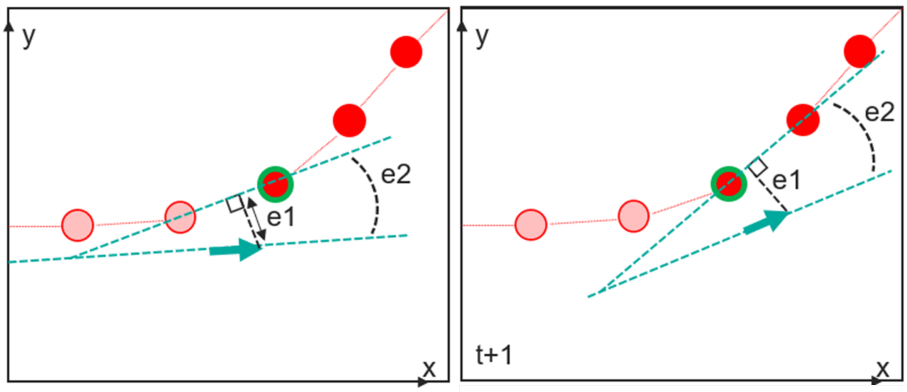


Figure 5.6: Lateral and heading error before and after surpassing the closest point.

Therefore,

$$\begin{aligned} e_1 &= d_y \cos\left(\psi_{ref}(cp_{heading})\right) - d_x \sin\left(\psi_{ref}(cp_{heading})\right) \\ e_2 &= \hat{\psi}_{ref} - \psi_{ref}(cp_{heading}) \end{aligned} \tag{5.12}$$

where e_2 is again wrapped between $[-\pi, \pi]$. The terms d_x and d_y were computed after searching the cp in Section 5.2.

The contribution to the steer reference from these two terms is computed:

$$\begin{aligned} \delta_{e_1} &= -K_1 \, sign(e_1) \, min(|e_1|, e_1^{max}) \\ \delta_{e_2} &= -K_2 \, e_2 \end{aligned} \tag{5.13}$$

Finally, the feed-forward term is computed. This component is introduced to make the bike respond faster to a heading change in the predefined path. Neglecting this term, the bike would only correct its heading after detecting the change through lateral and heading errors relative to the reference path. However, this corrective response could be sluggish, potentially leading to overshooting and significantly degrading path-tracking accuracy.

The feed-forward term $\dot{\psi}_{ref}$ obtained in section 5.2 is used. It is sampled by calculating the time needed to go from cp to the next point:

$$\begin{aligned} T_\psi &= \frac{d_{cp_{i+1},cp}}{v} \\ \dot{\psi}_{ref}^{sampled} &= \frac{\dot{\psi}_{ref}}{T_\psi} \end{aligned} \tag{5.14}$$

where the $d_{cp_{i+1},cp}$ is again obtained by using the Euclidean distance.

Hereinafter, the Laplace transform is applied to (5.3):

$$\begin{aligned} s\, e_1 &= v\, e_2 + \frac{l_r\, v}{l_r + l_f}\delta_e \\ s\, e_2 &= \frac{v}{l_r + l_f}\delta_e - \dot{\psi}_{ref}^{sampled} \end{aligned} \tag{5.15}$$

Eliminating e_2, gives:

$$e_1 = \frac{\frac{v^2}{l_r+l_f} + \frac{l_r\, v\, s}{l_r+l_f}}{s^2}\,\delta_e - \frac{\dot{\psi}_{ref}^{sampled}\, v}{s^2} \tag{5.16}$$

The control law (5.4) is extended by adding the feed-foward term from $\dot{\psi}_{ref}$ and becomes:

$$\delta_e = \delta_{e_1} + \delta_{e_2} + F_\psi\, \dot{\psi}_{ref}^{sampled} \tag{5.17}$$

where F_ψ is a filter chosen so that the effect from $\dot{\psi}_{ref}$ mitigates. Inserting (5.17) in (5.16), the error dynamics is obtained:

$$e_1 = \frac{v^2 + l_r\, v\, s}{(lr_+ l_f)\, s^2}(-K_1\, e_1 - K_2\, e_2 + F_\psi\, \dot{\psi}_{ref}^{sampled}) - \frac{\dot{\psi}_{ref}^{sampled}\, v}{s^2} \tag{5.18}$$

Hence, to get rid of the effect of $\dot{\psi}_{ref}$, F_ψ is choosen so that:

$$\frac{v^2 + l_r\, v\, s}{(l_r + l_f)\, s^2}\, F_\psi\, \dot{\psi}_{ref}^{sampled} = \frac{\dot{\psi}_{ref}^{sampled}\, v}{s^2} \tag{5.19}$$

which gives

$$F_\psi = \frac{1}{\frac{v}{l_r+l_f} + \frac{l_r\, s}{l_r+l_f}} \tag{5.20}$$

This algorithm will be used in LabVIEW using C-code. Thus, this filter is transformed into a state space form. The matrices are discretized the same way as in (4.13) and they are constant. Thus, the contribution from the feed-forward term due to a change in the heading of the predefined path is:

$$\begin{aligned} h(t+1) &= A_{t_d}\, h + B d_t\, d_{\psi_{ref}} \\ \delta_{\dot{\psi}_{ref}} &= C_t\, h + D_t\, d_{\psi_{ref}} \end{aligned} \tag{5.21}$$

The steering reference (δ_{ref}) is obtained by adding all three different contributions together:

$$\delta_{ref} = \delta_{e_1} + \delta_{e_2} + \delta_{\dot{\psi}_{ref}} \tag{5.22}$$

The δ_{ref} to track the predefined path is computed. This reference is transformed into a roll angle reference as explained in Section 1.2.2.

5.4 Validation of the trajectory controller

In this section, the validation of the trajectory controller in Simulink is covered. The validation is focused on the closest point algorithm and the contributions to δ_{ref} from the errors and feed-forward term. The LQR is already validated in the [3].

5.4.1 CPA validation

To validate the closest point algorithm, a specific test has been designed. The designed trajectory is not a "good" one for the bike to follow since the distance between some points is very big. However, it is very useful to check if the algorithm is behaving as expected. In the Figure 5.7, the mentioned reference trajectory is shown together with the true and estimated signals. The last two together and the measurements are on top of each other making it hard to distinguish. On the other hand, Figure 5.8 illustrates the cp and $cp_{heading}$ indexes over time.

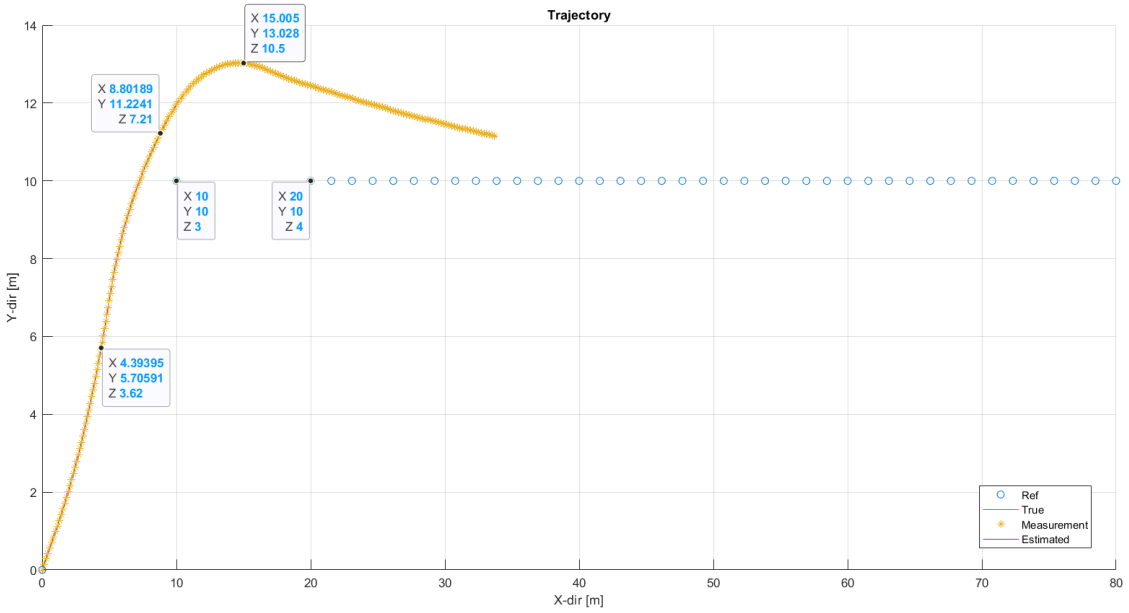


Figure 5.7: Reference, true, estimated, and measured trajectories. The first three points of the reference and times where the cp or $cp_{heading}$ indexes change are indicated. The Z axis in the reference signal represents the index while the others represent the time. Although the tracking is not good, it is expected since the aim of the trajectory is to validate the CPA algorithm.

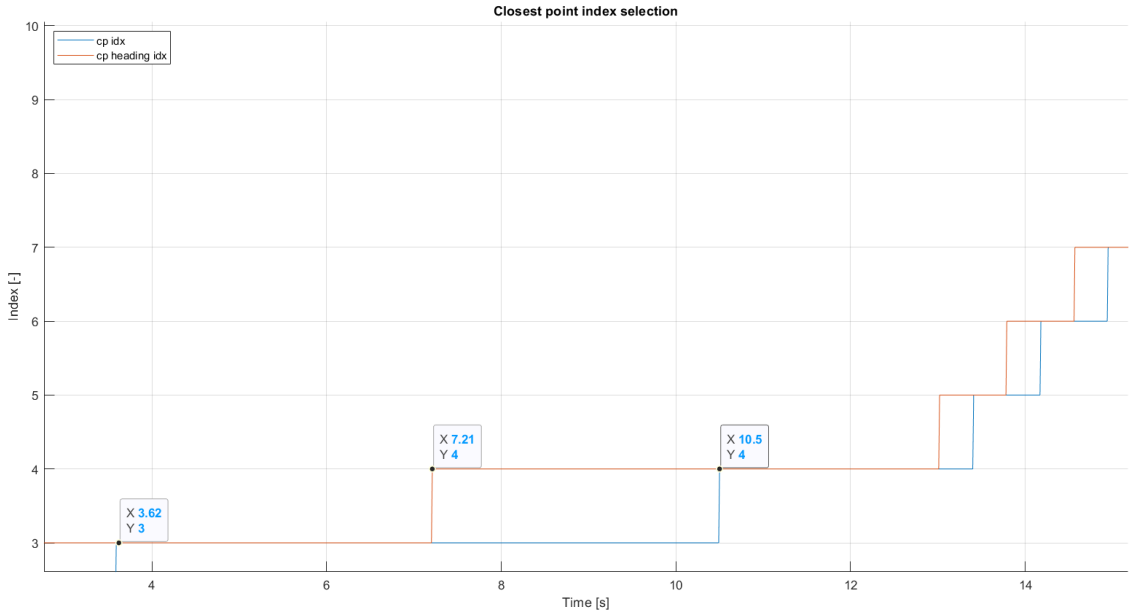


Figure 5.8: The variation of cp and $cp_{heading}$ indexes in time is depicted to validate the closest point algorithm. The highlighted points and times in Figure 5.7 are also indicated here. The figure shows the first part of the test which is the period when the bike is referring to the highlighted points.

Some observations from Figure 5.7 and Figure 5.8:

- The $cp = 2$ in the start since the first point of the reference is duplicated. This is because two points are needed to obtain the heading (ψ_{ref}):

$$\psi_{ref,i} = \arctan\left(\frac{Y_{ref,i} - Y_{ref,i-1}}{X_{ref,i} - X_{ref,i-1}}\right) \tag{5.23}$$

- The search for the closest point can be analyzed in $t = 3.62$ and $t = 10.5$. Before these time instants, the bike has already surpassed the cp but is still closer to it than to the next point. After those instants, the bike is closer to the next point leading to the cp being changed.
- In the beginning, the $cp_{heading} = 3$ since the interpolation algorithm detects that the $cp = 2$ has been already surpassed. Therefore, the next heading is taken to avoid delays and overshoots. This phenomenon is also illustrated in $t = 7.21$, the $cp = 3$ since it is the closest point to the reference but the $cp_{heading} = 4$ because the cp has been surpassed. Thus, it takes the heading of the next point.

5.4.2 Limitations of the closest point algorithm

Two cases where the closest point algorithm fails are presented. In the first case, the trajectory is composed of a horizontal line that suddenly jumps vertically 20m and continues horizontally. There is no change of heading (see Figure 5.9).

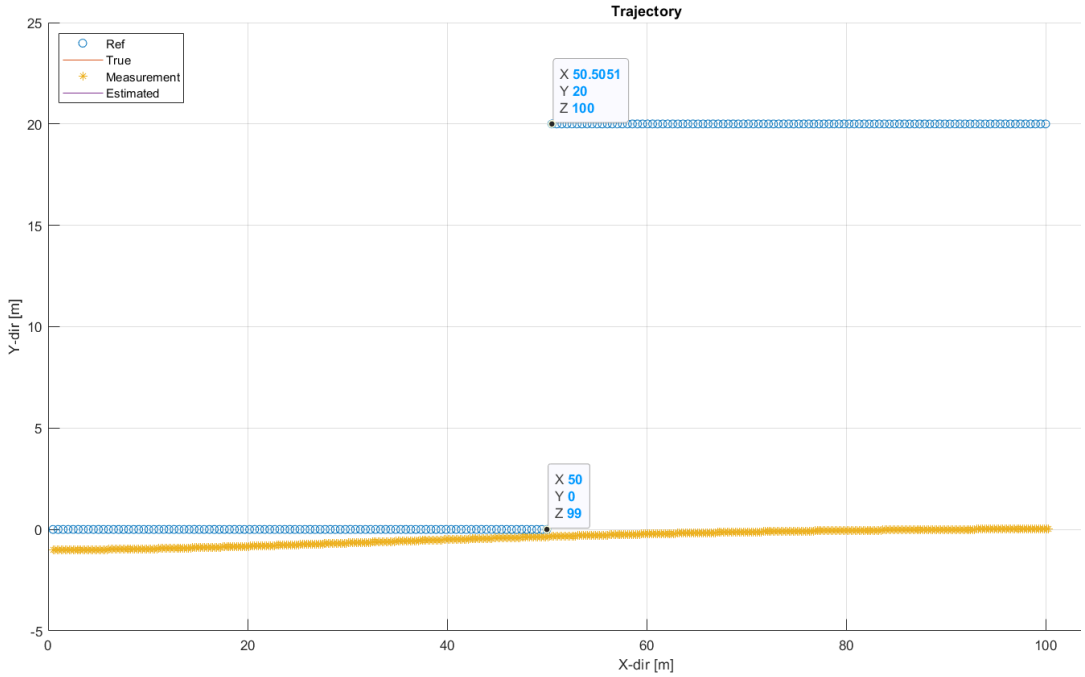


Figure 5.9: Case 1: sudden vertical jump with no heading change in the reference path. The true, measured, and estimated signals are shown. The indexes of the reference points when the jump happens are highlighted (Z axis). The algorithm does not detect the jump and continues moving forward.

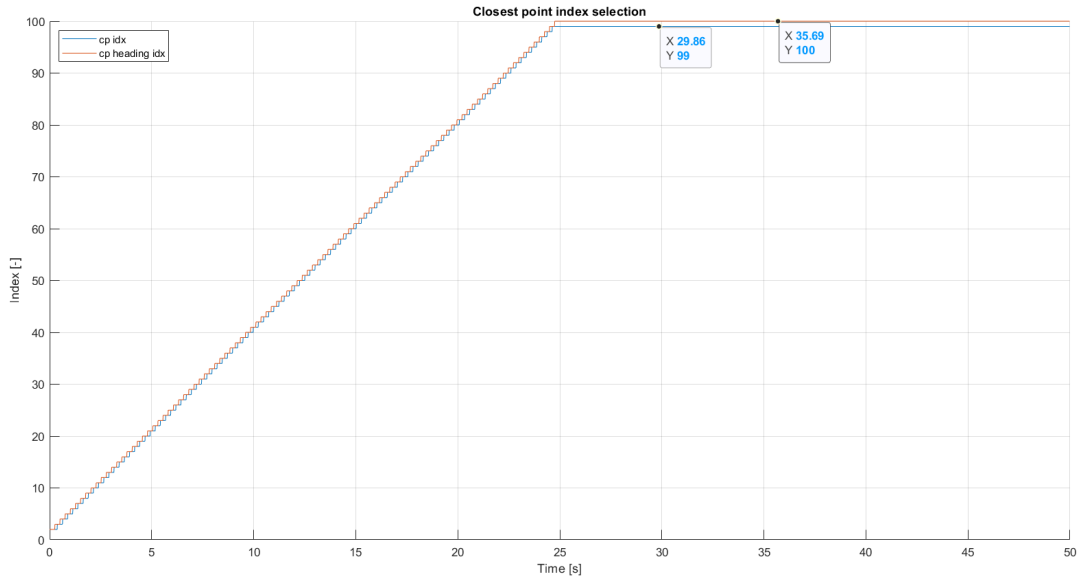


Figure 5.10: Case 1: cp and $cp_{heading}$ index evolution through time. The indexes when the jump happens shown in Figure 5.9 are also pointed out here. After the jump, the indexes do not change anymore since the next point is always farther than cp.

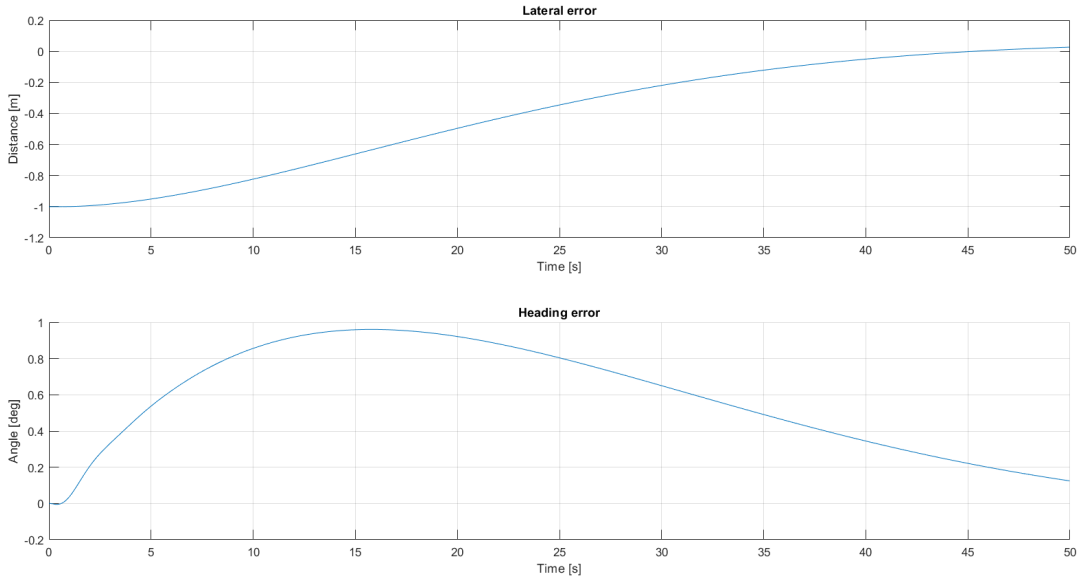


Figure 5.11: Case 1: lateral and heading errors. The lateral error does not increase when the jump in the reference happens since the *cp* is not changing.

From Figure 5.9 (the true, estimated, and measurement signals are on top of each other.), Figure 5.10 and Figure 5.11, it is observed that after $t = 25s$, *cp* and $cp_{heading}$ do not change anymore. The reason is that the next point is always farther than the *cp*, so it does not change. The heading of the points is not changing either. Therefore, no sudden increase in the lateral error is detected and the bike continues going straight.

In the second case, the trajectory has the same shape but the jump is filled with points. Therefore, two 90-degree turns are presented (see Figure 5.12). The true, estimated, and measurement signals are on top of each other.

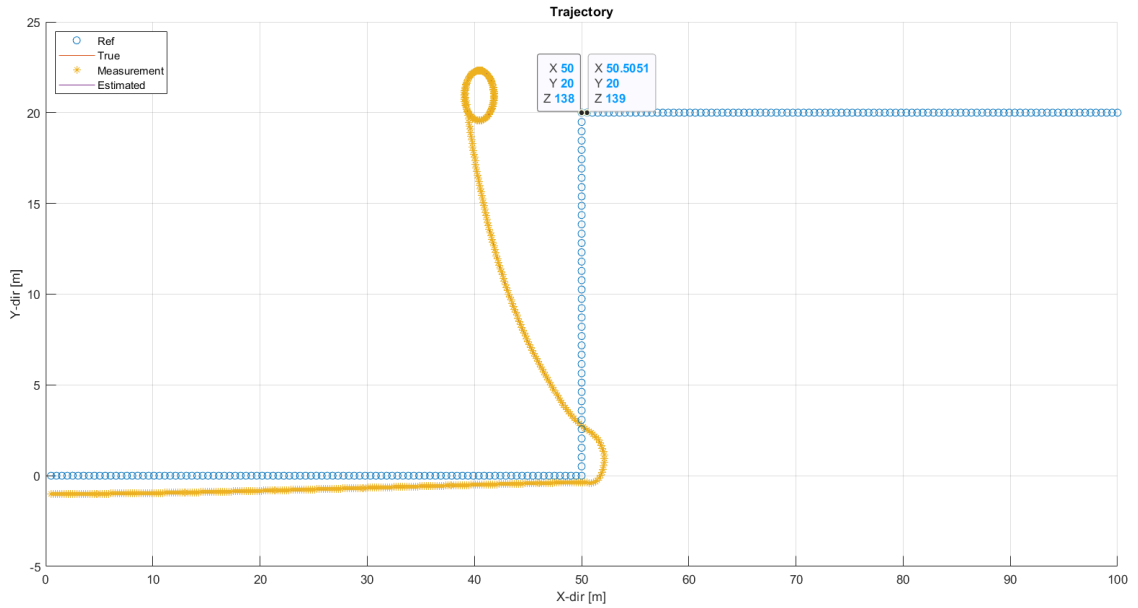


Figure 5.12: Scenario 2: same vertical jump but it is filled with points. Two sharp turns appear which cannot be handled by the controller. The true, measured, and estimated signals are also presented. The main points are emphasized again.

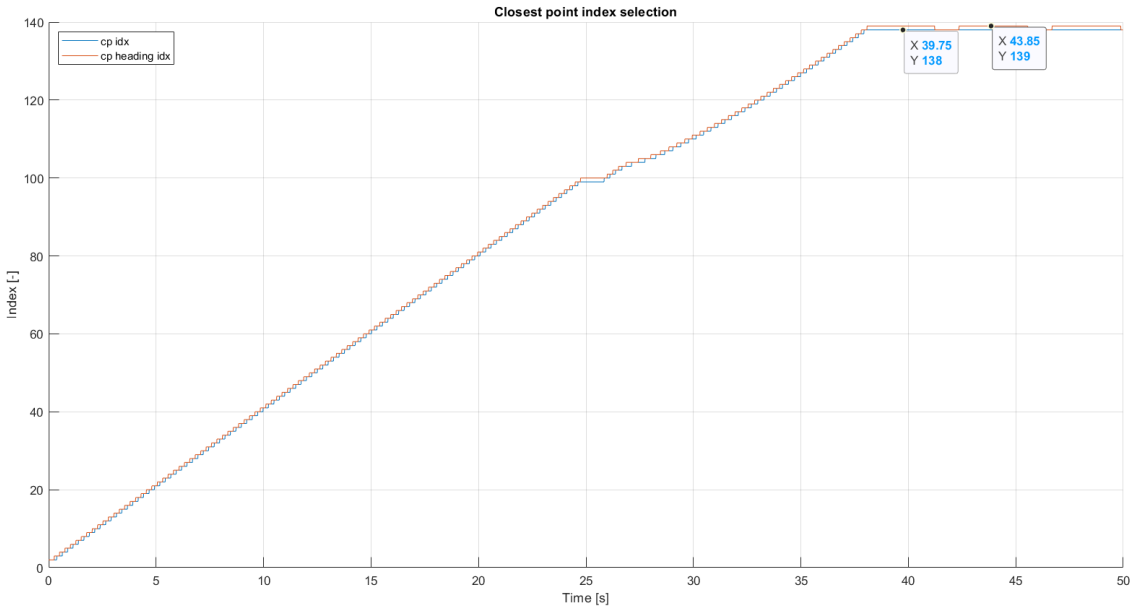


Figure 5.13: Scenario 2: cp and $cp_{heading}$ evolution through time. The moment where the second sharp turn happens is pointed out (see Figure 5.12).

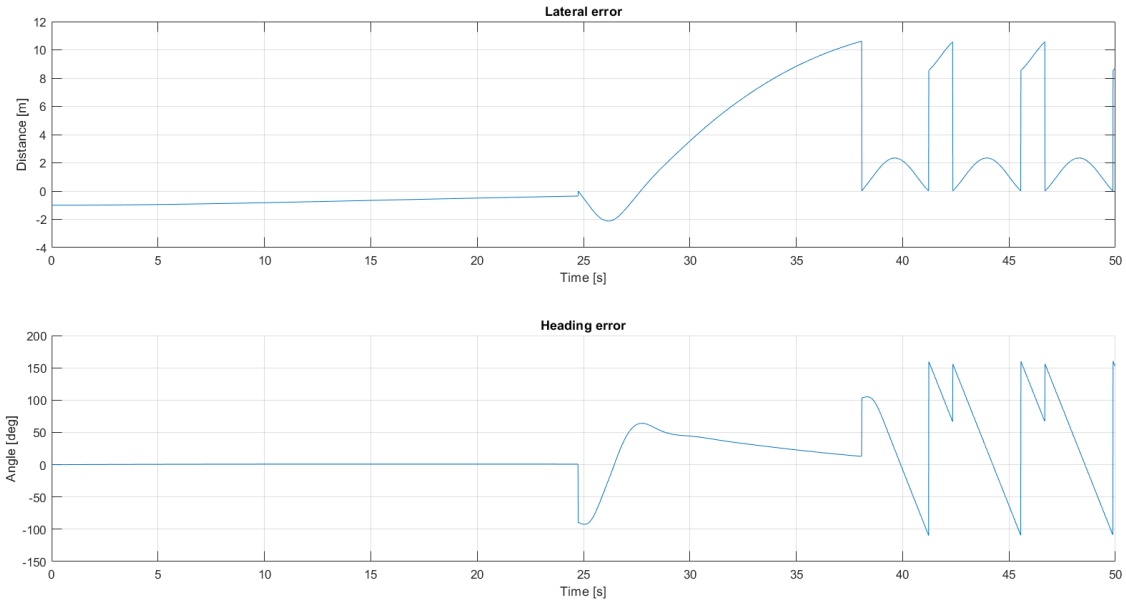


Figure 5.14: Scenario 2: lateral and heading errors. It can be seen that when the sharp turns arrive, a sudden increase in the errors can be noticed.

From Figure 5.12, Figure 5.13, and Figure 5.14, the two sharp turns are analyzed. In the first sharp turn, it can be seen that the closest point gets stuck for a few seconds which makes the feed-forward term have a lot of impact. This makes the bike able to turn fast but it deviates a lot from the reference. Thus, when the second sharp turn arrives, the bike is way off the reference which makes the algorithm unable to go forward. As a result, the bike starts to do circles.

However, both cases have something in common. When the references are generated, a warning pops up. In the first case, the warning is that the distance between the two points was too far away. In the other scenario, it warns that too sharp turns were found.

To summarize, the closest point algorithm has some limitations that need to be found. They are usually associated with the reference generation. Thus, it is very important to introduce warnings when a limitation is found to avoid having them in future tests. The reference check that is used is still very basic and needs to be improved in the future.

5.4.3 Validation of the steering reference

A more realistic trajectory in Figure 5.15 is introduced to validate the derivation of steering reference contributions from the various errors. The reference, true, measure, and estimated paths are displayed. Similarly, as in the previous case, the true, estimation, and measurement signals are very on top of each other. The path consists of a straight line and then a constant radius circle is drawn. In addition, an offset of 1m is introduced at the beginning to evaluate how the lateral controller corrects the lateral error.

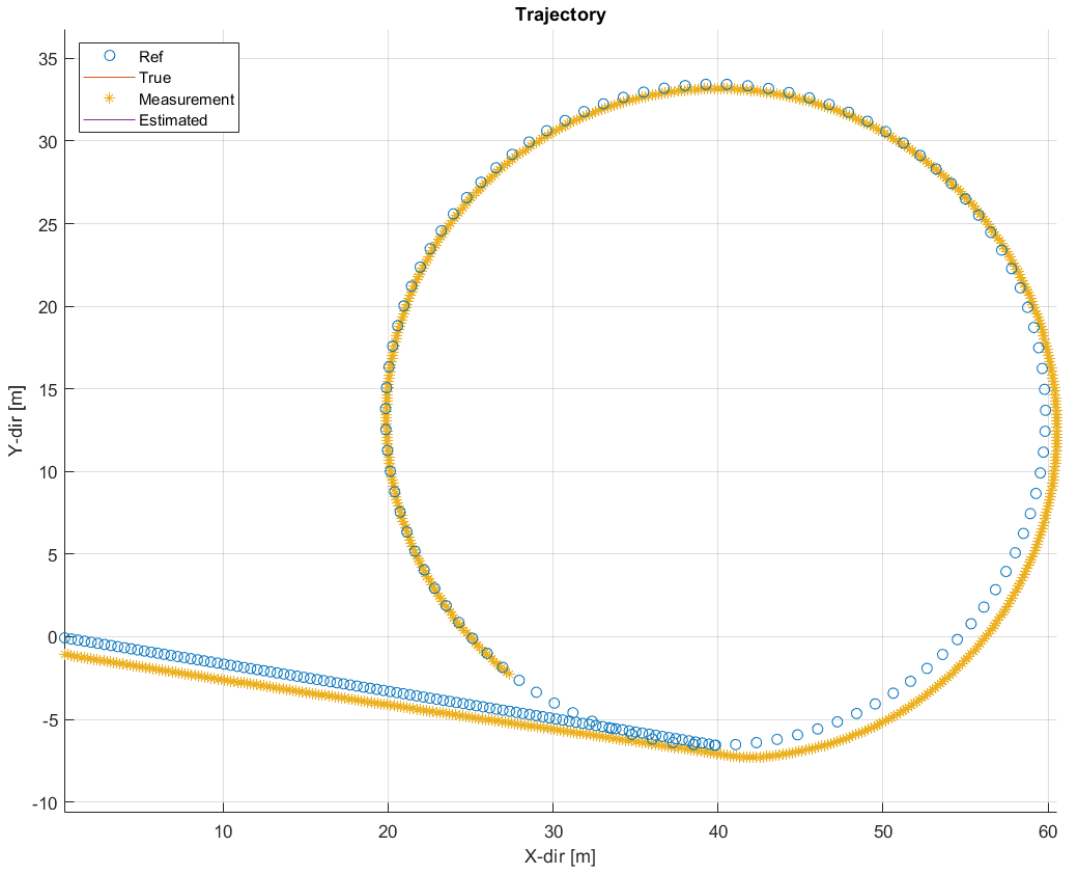


Figure 5.15: Simulation with a more realistic reference path. A 1m offset is introduced in the beginning to evaluate how the lateral controller corrects this error. The constant radius circle allows checking how the lateral controller and the feedforward work simultaneously. The true and estimated trajectories cannot be distinguished since it is under the measurements.

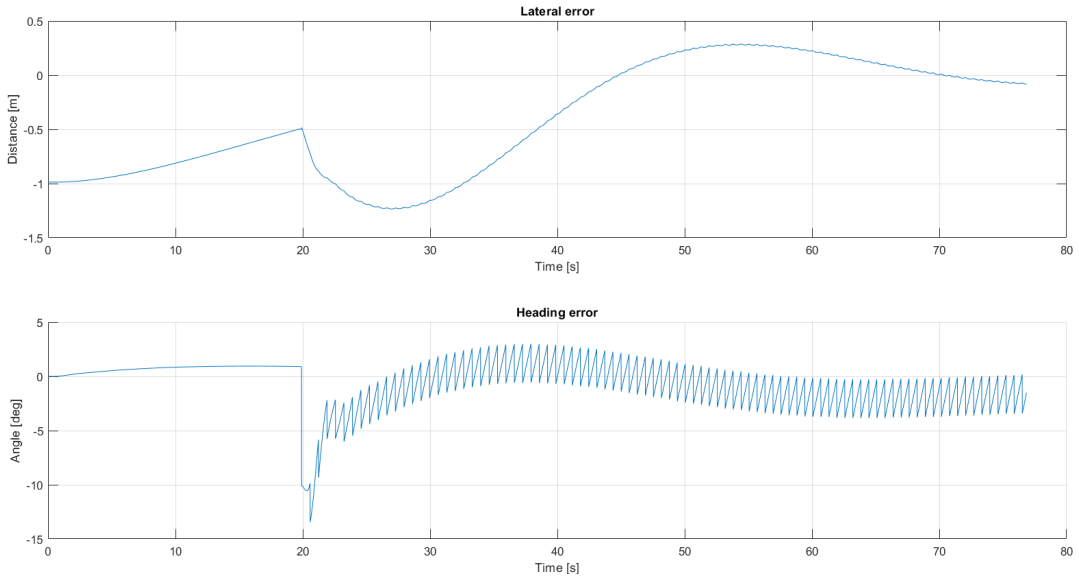


Figure 5.16: Lateral and heading errors with respect to the realistic path. As mentioned before, an offset of 1m is added to the bike with respect to the path at the beginning. It can be seen how the controller is acting on that error and reducing it slowly during the first 20 seconds. After, the circular path is detected with a step heading error together with an increasing lateral error. These errors are slowly reduced. Additionally, it can be observed that the heading error is a very spiky signal. The source of the spikes comes from the closest point algorithm. When the cp or $cp_{heading}$ indexes are changed, there is a fast change in the reference point which triggers a sudden change in the lateral and heading errors.

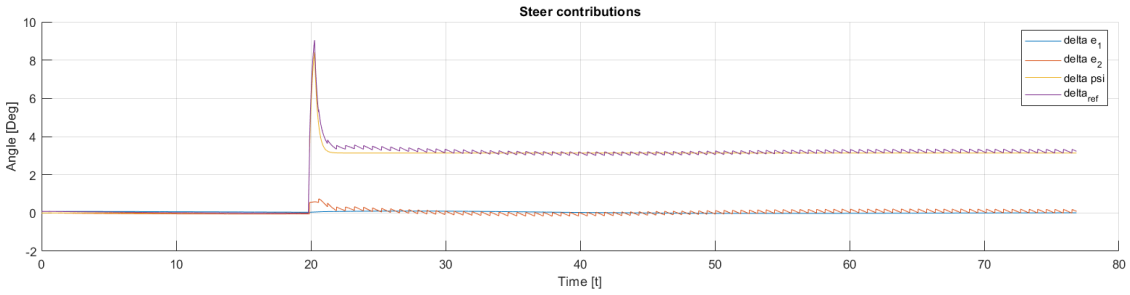


Figure 5.17: Contributions to δ_{ref} from the errors and feed-forward to follow the reference path. After those 20 seconds, the constant radius circle is approached. Therefore, a change in the reference heading appears (constant heading change) which makes the feed-forward act. The feed-forward term acts quickly (sudden step) and it reduces smoothly (filtered) until it stabilizes around 3 deg (constant heading change). Additionally, there are lateral and heading contributions but they are much lower in magnitude.

6

Real testing

In this chapter, an overview of the steps involved in conducting real-world tests is provided. Furthermore, a showcase of one of the most recent tests conducted is presented. The showcase aims to show the available plots to analyze data and offer an understanding of the project's current status.

6.1 Preparation and real test procedure in the bike

The steps taken for real-world tests are:

1. Trajectory generation. To generate the trajectory, points are taken from Google Maps (the place of testing must be known). These points are longitude and latitude coordinates that need to be transformed into global coordinates with:

$$\begin{aligned} X &= R_{earth}\,(long - long_{ref})\,\cos(lat_{ref}) \\ Y &= R_{earth}\,(lat - lat_{ref}) \end{aligned} \tag{6.1}$$

   where $long_{ref}$ and lat_{ref} are the reference coordinates from where the earth is approximated as a plane. It is important to choose them relatively close to the testing place to avoid big approximation errors.

2. The *Startup_labview.m* file is run. In this file, some general settings are chosen. For instance, the constant velocity that the bike will have, which bike (to load its parameters), and the tuning parameters (Q_K, R_K, Q_T, R_T, P or D) among other things.

   Once the values are chosen, the file is run, and the window of Figure 6.1 pops up. In this window, different shapes for the trajectory can be selected by introducing the initial and final coordinates, number of data points, and different parameters depending on the shape of the trajectory. During the process, the trajectory can be visualized before saving it and trajectories can be concatenated.

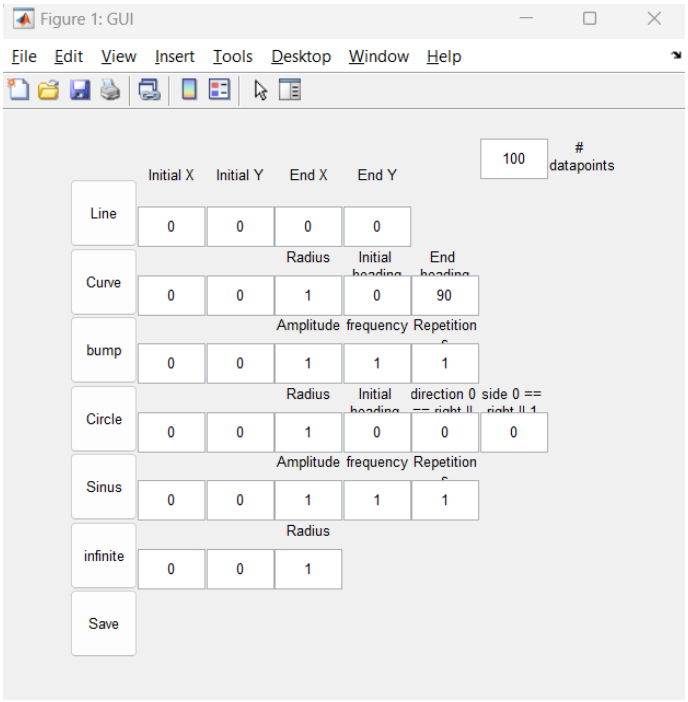


Figure 6.1: Trajectory generator GUI in MATLAB.

When the reference is generated, it is also checked automatically. The program checks the reference in terms of the density of points, the sharpness of the turns, and more. A warning will be sent in case any of these criteria are not fulfilled. However, this automatic check is still very basic and it should be highly improved in the future.

This MATLAB file generates two additional .csv files: "*trajectorymat.csv*" and "*matrixmat.csv*". The first file collects $[X_{ref}, Y_{ref}, \psi_{ref}]$. The second saves all the chosen settings, parameters, and matrices that are needed for LabVIEW.

3. An optional step is to simulate the generated reference. It is recommendable since it is useful to detect undesired behaviors or issues before applying it to the real bike. This step can prevent undesired motions that may lead to falling and breaking the bike, as well as saving a lot of time.

4. Before running the test on the real bike, a few things need to be done in LabVIEW. First, the generated files are uploaded to the myRIO (microprocessor) of the bike. In addition, inside the project of the bike in LabVIEW, two VI's are run before the "*Main.vi*". The "*Calibration.vi*" collects and saves the gyroscope bias by leaving the bike steady and untouched for five seconds. Secondly, the "*Configuration.vi*" collects all the settings and the reference trajectory from the uploaded files from MATLAB.

5. The test execution takes place within the "*Main.vi*" program. Upon initiating the file, the test procedure starts. The trajectory is initialized followed by a brief pause to let the GPS settle. Afterward, the steering of the bike is aligned

to reset the steering encoder. The desired riding velocity is then inputted, prompting the bike to begin its motion. A team member runs with the bike, aiding its balance and ensuring straight movement. Once the straightness is achieved, a signal is given to the other member on the computer to enable the steering motor. In response, the team member alongside the bike releases it, and the bike should start following the path. In the Figure 6.2, the main front panel buttons and inputs needed for the explained process are shown.

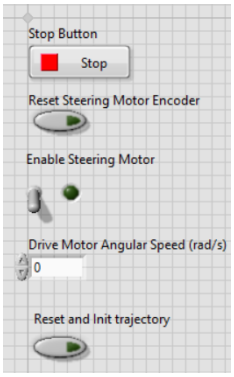


Figure 6.2: Main front panel inputs in LabVIEW.

The test finishes when the stop button is pushed in the front panel or the emergency button in the bike is pressed. At that moment, all the motors are turned off and the member next to the bike holds it again.

6.2 Showcase of a real test

After the test is finished, all the data is logged and saved automatically in the myRIO. The logged data can be taken and saved on the computer. To analyze this data, the same MATLAB/Simulink files are used as in Section 4.7.3.

The presented test was performed at the test track of AstaZero. The red bike was used. The velocity of the bike was 2.4 m/s. The values of the tuned parameters are:

$$\begin{aligned} P &= 3.75 \\ D &= 3.5 \\ Q_K &= diag(0.1, 0.1, 0.1, 10^{-9}, 5, 10, 0.5) \\ R_K &= diag(1.5677, 1.5677, 0.2564, 3.94 \cdot 10^{-12}, 0.0234, 4.15 \cdot 10^{-5}, 0.1) \\ Q_T &= 6 \cdot 10^{-7} \begin{bmatrix} 1000 & 0 \\ 0 & 100 \end{bmatrix} \\ R_K &= 0.2 \end{aligned} \tag{6.2}$$

The Q_K and R_K are the same as in Section 4.7.3. In the case of Q_T and R_T, the study [3] proposes some values after some testing in simulation. However, these values did not work in the real bicycle, as well as in the simulation. Therefore, these new values in (6.2) were obtained by trial and error method testing on the red bike.

The test aims to see the filter's performance in tracking a path that consists of a straight line and three constant and equal circles. The straight line is used to have time to throw the bike but the test aimed to check the tracking of a circle. Having three circles is to test the repeatability. It is important to mention that all the data of this test is plotted from the moment that the steering is enabled. The previous data is of no interest because it is manipulated by the member holding the bike. The online and offline KF estimations are plotted together to detect any possible error.

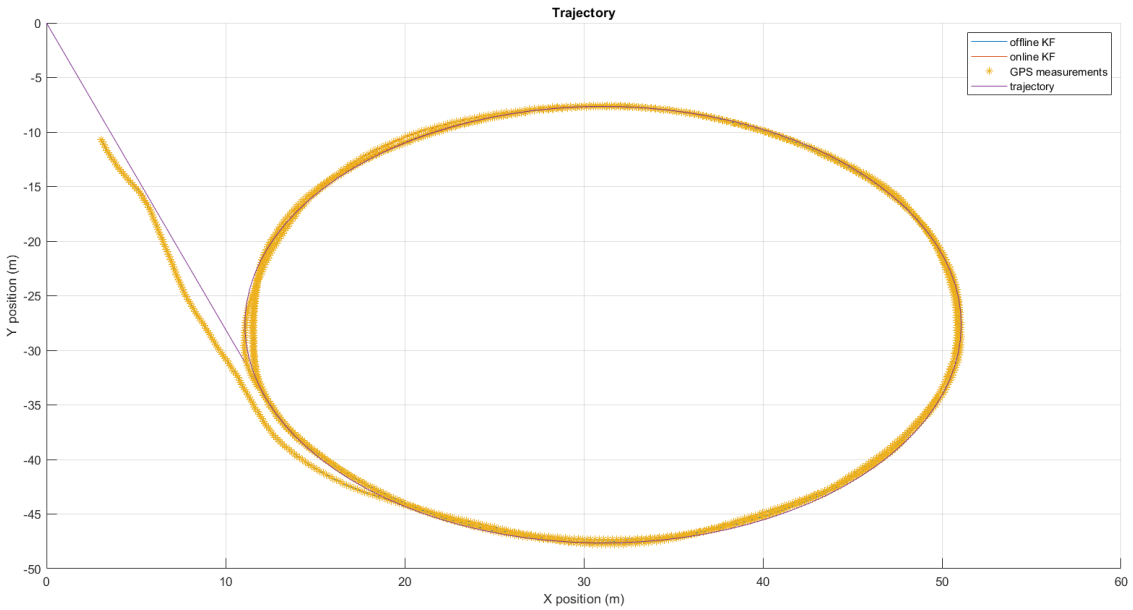


Figure 6.3: Reference and estimated path in a real bike test. The reference consists of a straight line followed by three circles of the same radius to check if the bicycle can track a circular reference and its repeatability. The estimations and measurements are shown from the moment where the steering motor is enabled.

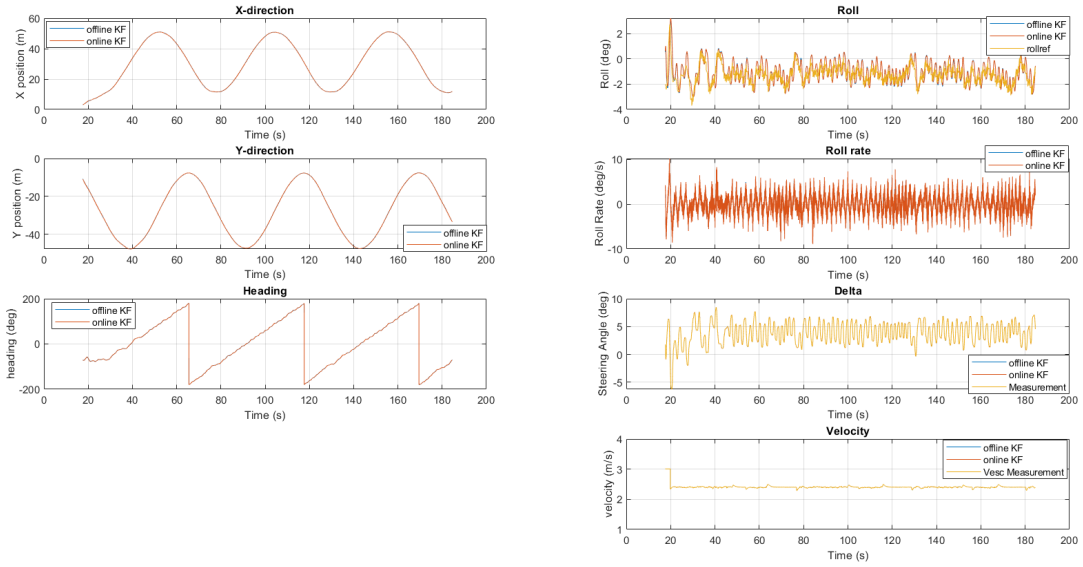


Figure 6.4: State variable estimations in a real bike test. In the case of the roll, the reference is shown for comparison. It can be seen that the roll tracking is quite accurate since the reference and true signals are similar. Furthermore, the steering angle estimation is identical to the measurements since the filter is tuned to rely a lot on the measurement (very accurate encoder). In general, all the estimations are accurate. The signals that are not distinguished are because they are on top of each other.

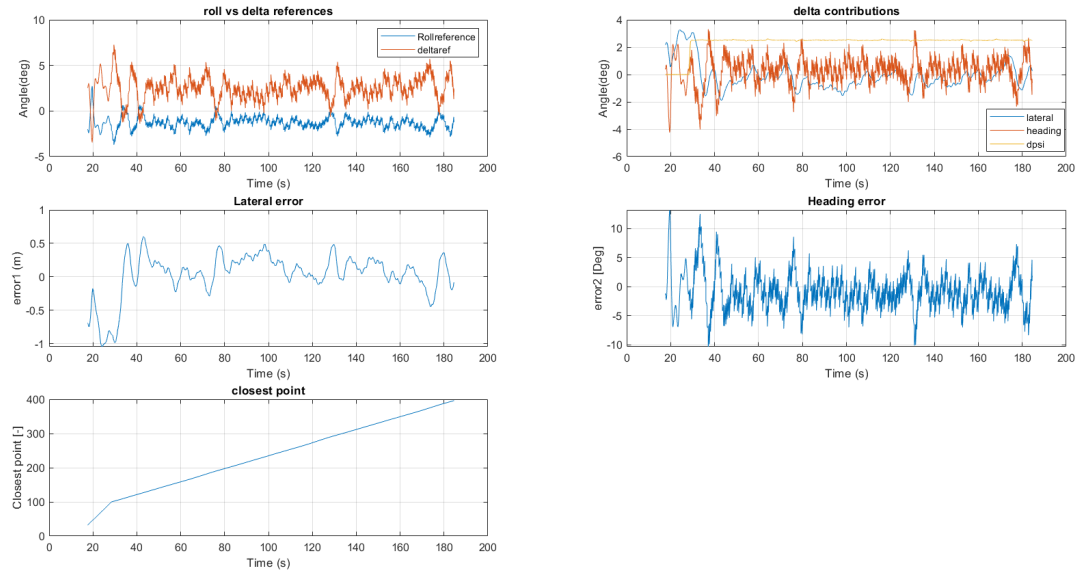


Figure 6.5: Transformation from δ_{ref} to φ_{ref}, lateral and heading errors and their contributions to the δ_{ref}, and the closest point index. It can be seen from the δ contributions that the feed-forward is the most dominant term (expected since a circular path is tracked). The feed-forward is responsible for the bike being able to follow the circular trajectory while the lateral and heading contributions are responsible for correcting the trajectory on the circle for a higher accuracy. The *cp* plot is just to check possible failures.

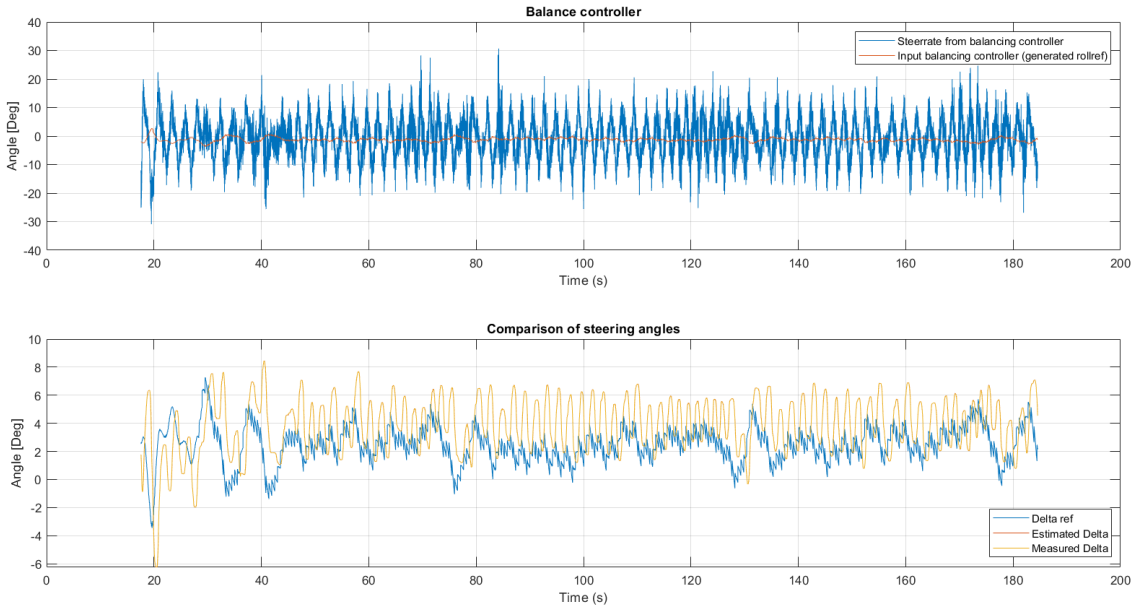


Figure 6.6: Input-output of balance (just to check possible errors) and comparison between steering angles. For the second plot, the estimated and measured are the same signal and are referred to as one now. From Figure 6.3 and Figure 6.5, it can be seen that the bike is following the reference quite accurately ($\pm 0.5m$) but still is not perfect. This is because the δ_{ref} and the estimation have the same tendencies but they are still not identical.

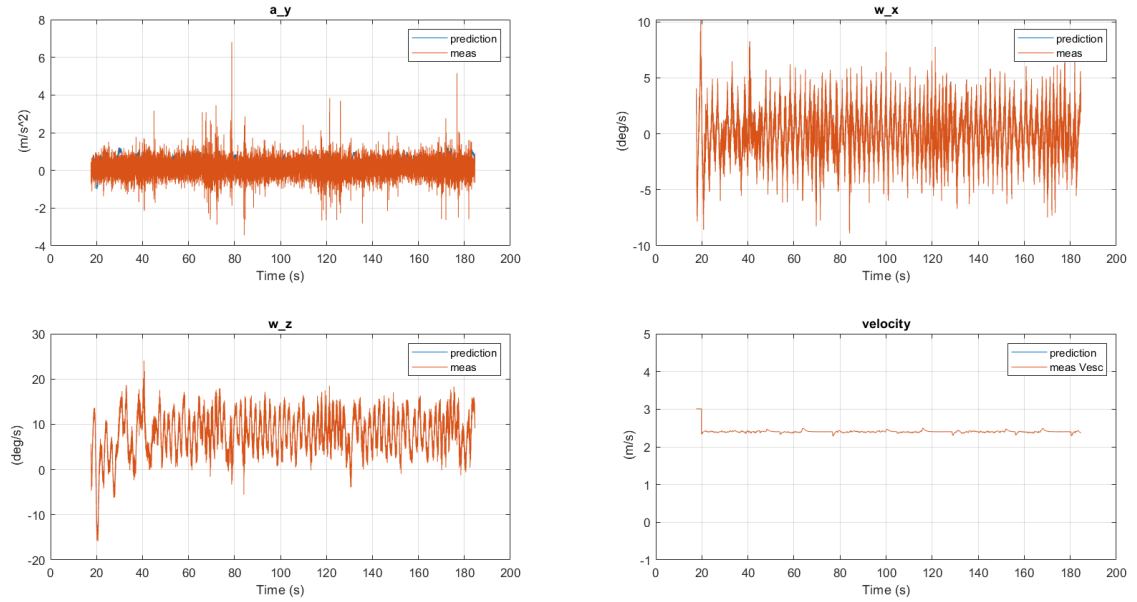


Figure 6.7: Comparison between the measurements and predicted measurements in the Kalman filter. In all the plots, both signals are difficult to distinguish meaning that the prediction is accurate. Used to check possible errors.

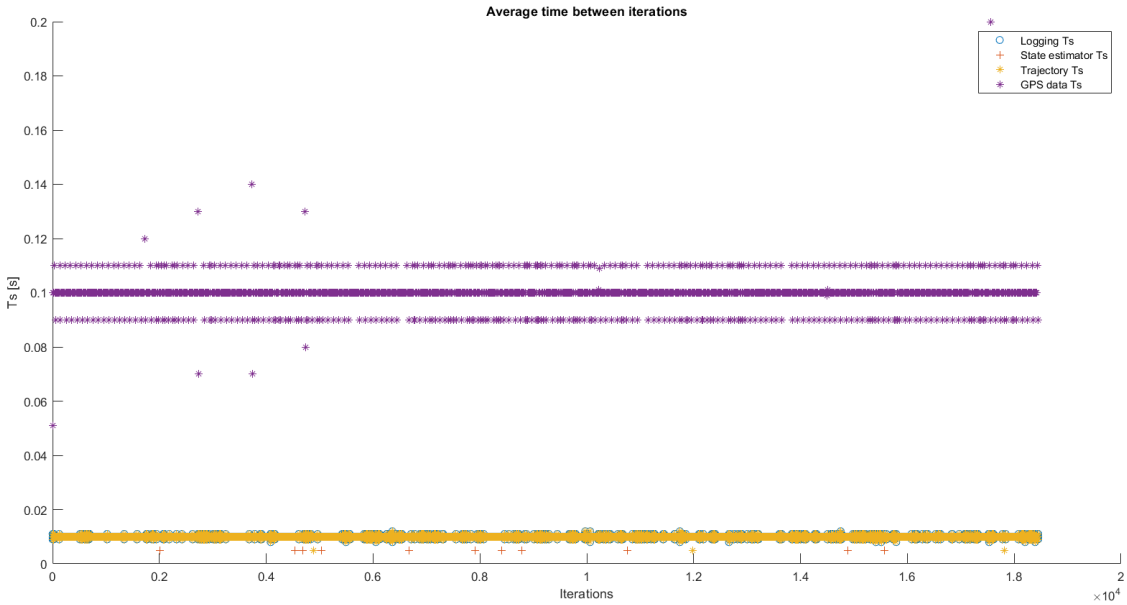


Figure 6.8: The average time between iterations of the different systems. It is used to check that no errors have occurred.

From this test, it can be said that the bicycle can track simple trajectories accurately at this stage of the work. All the developed functionalities that were validated in the simulation are working in the real bicycle. In addition, it is likely that with better tuning, the tracking performance could be improved.

7
Future Work

In this chapter, specific areas for improvement and potential future directions include:

- Redesign the trajectory generation algorithm. The way it works now, every time the place of testing is changed, all the shapes need to be created again since it is not possible to change the direction of the references. Find a way of creating and saving a set of shapes in which it is possible to change their direction when it is necessary.
- Check the communication with the VESC module. Usually, the velocity gets stuck for some seconds to the values of the previous tests. It is necessary to wait until is reset. Thus, it is a waste of time.
- The way the warning messages or signals are obtained is still elementary, e.g., the sampling times are checked with a plot. In addition, include warning messages when unexpected GPS jumps appear or VESC communication gets stuck.
- Improve the tuning of the parameters in the red bike to progress in the path tracking. Easy references are successfully followed but still, it can be improved. Also, check the tracking performance on more complex paths. In the case of the other vehicles, such as the green bike or the scooter, the parameters need to be tuned since probably they will differ from the values of the red bike.
- Design speed and position control. The speed control by controlling the torque and position control to follow the trajectory. Moreover, introduce these controllers to the system both in simulation and in LabVIEW.

Bibliography

[1] E. Umur (2019). "Modelling, validation and control of an autonomous bicycle". Chalmers University of Technology / Department of Electrical Engineering.

[2] Aleix Ricou Pujal (2023). "Trajectory control implementation in autonomous bicycles". PhD thesis, UPC, Escola Tecnica Superior d'Enginyeria Industrial de Barcelona, Departament d'Enginyeria Quimica.

[3] Guanzheng Wen and Jonas Sjoberg (2022). "Lateral control of a self-driving bike" in IEEE International Conference on Vehicular Electronics and Safety (ICVES).

[4] "Kalman filter", wikipedia.org.
Available: https://en.wikipedia.org/wiki/Kalman_filter#Overview_of_the_calculation

[5] G. Welch and G. Bishop (2006). "An Introduction to the Kalman Filter," University of North Carolina at Chapel Hill. (Online resource: https://www.cs.unc.edu/ welch/media/pdf/kalman_intro.pdf).

[6] Lars Hammarstrand notes for the course Sensor fusion & nonlinear filtering at Chalmers University.

[7] Research Institute for advanced study, Baltimore, Md. R.E. Kalman, "A New Approach to Linear Filtering and Prediction Problems," Journal of Basic Engineering, Vol. 82, No. D, 1960.

[8] idare function from mathworks. https://se.mathworks.com/help/control/ref/idare.html

[9] Maxon, "Dcx32," https://www.maxongroup.com/maxon/view/configurator/BOM:DCX32L01GBKL511:::::

[10] RS(Broadcom), "Encoder," https://se.rs-online.com/web/p/optical-rotary-encoders/8188849

[11] RS(CTi), "Gps antenna," https://int.rsdelivers.com/product/cti/gps_wp/trk/sma_3-0/cti-gps_wp/trk/sma_3-0-dome-omnidirectional-gps-antenna-with-

sma/2053331

[12] SHIMANO, “Shimano steps,” https://bike.shimano.com/en-EU/product/component/citytrek-ebike-e6000/DU-E6010.html

[13] Maxon, “Escon 50/5,” https://www.maxongroup.com/maxon/view/product/control/4-QServokontroller/409510etcc_cu=onsite&etcc_med=Header%20Suche&etcc_cmp=mit%20Ergebnis&etcc_ctv=Layer&query=escon%2050%2F5

[14] FLIPSKY, “Fsesc 6.7 pro,” https://flipsky.net/products/fs-esc-6-6

[15] RS(SparkFun), “Gps module,” https://www.digikey.it/it/products/detail/sparkfun-electronics/GPS-15136/9856841

[16] InvenSense, “Mpu-6050 datasheet,” https://invensense.tdk.com/wp-content/uploads/2015/02/MPU-6000-Datasheet1.pdf

[17] Emilio Sanjurjo, Miguel A. Naya, Javier Cuadrado & Arend L. Schwab (2019). Roll angle estimator based on angular rate measurements for bicycles, Vehicle System Dynamics, 57:11, 1705-1719, DOI: 10.1080/00423114.2018.1551554. https://doi.org/10.1080/00423114.2018.1551554

A

Appendix 1

The trajectory computes the δ_{ref} angle that the bike needs to have to follow the predefined path. On the other hand, the balancing controller of the bike works with rolling angles to balance the bike as explained in section 1.2.3. Therefore, the δ_{ref} needs to be transformed into φ_{ref} to make the bicycle track the predefined path without falling.

The expression for φ is taken from [17]. It is obtained by assuming steady-state cornering motion and using the lateral dynamics equilibrium (see Figure A.1):

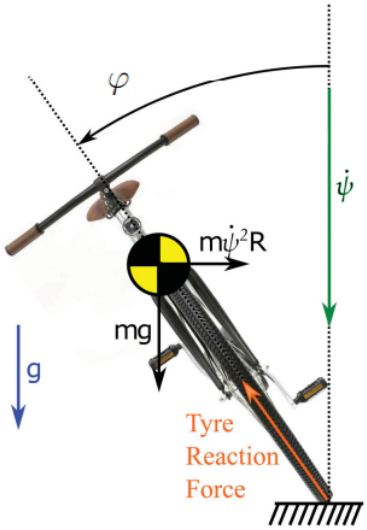


Figure A.1: Dynamic force equilibrium during steady-state cornering.

$$\varphi = -\arctan\left(\frac{\dot{\psi}^2\, R}{g}\right) = -\arctan\left(\frac{\dot{\psi}\, v}{g}\right) \tag{A.1}$$

where v is the forward speed and $\dot{\psi}$ is the yaw rate. The expression for $\dot{\psi}$ is derived in [1].

According to Figure A.2, the path radius of the rear wheel of the bicycle (ρ_{rear}) is defined:

$$\rho_{rear} = \frac{b}{\tan(\delta^{eff})} \tag{A.2}$$

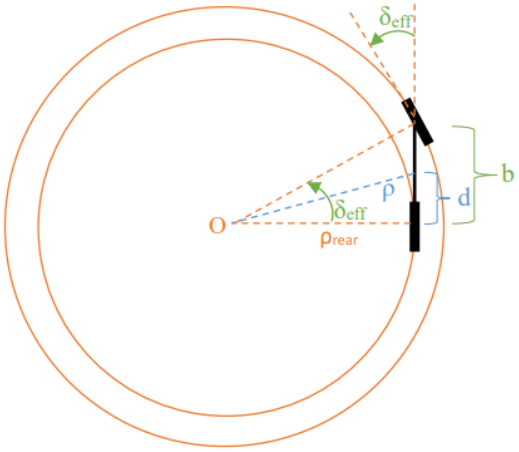


Figure A.2: Top view of the bicycle on a circular path.

Considering Figure A.2 and A.3, $\dot{\psi}$ is defined considering the contact point of the rear wheel:

$$\dot{\psi} = \frac{v}{\rho_{rear}} = \frac{\tan(\delta^{eff})\, v}{b} \tag{A.3}$$

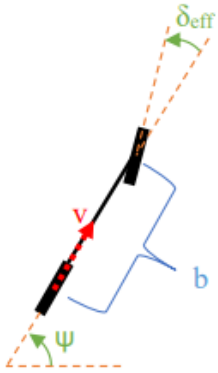


Figure A.3: Yaw angle definition.

Introducing (A.3) into (A.1) and changing b to the notation used in this work (l_r+l_f), the relation is obtained:

$$\begin{aligned} \delta_{ref}^{eff} &= \delta_{ref}\, \sin(\lambda) \\ \varphi_{ref} &= -\arctan\left(\tan(\delta_{ref}^{eff})\, \frac{\frac{v^2}{(l_r+l_f)}}{g}\right) \end{aligned} \tag{A.4}$$

This relation is used to change the δ_{ref} computed by the trajectory controller into φ_{ref} to be compared in the balancing controller.

DEPARTMENT OF SOME SUBJECT OR TECHNOLOGY
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se

CHALMERS
UNIVERSITY OF TECHNOLOGY