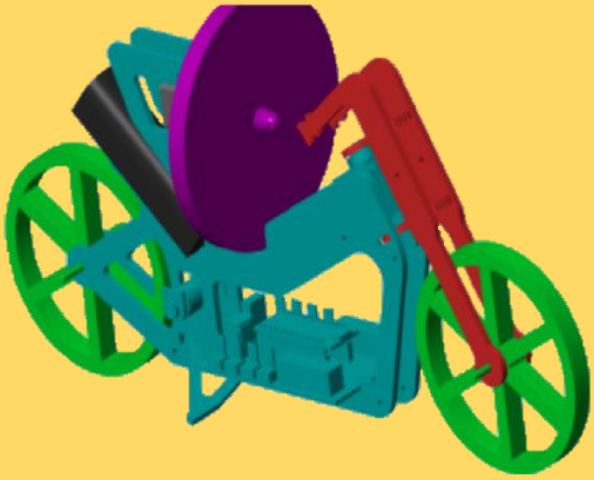
Faculty of Natural, Mathematical and Engineering Sciences

Department of Engineering

KING'S College LONDON



INDIVIDUAL ENGINEERING PROJECT

Matteo Liguori

Comparative Analysis of Sensor Fusion Techniques for State Estimation in Autonomous Navigation of a Self-Balancing Bicycle

Faculty of Natural, Mathematical and Engineering Sciences

Department of Engineering

6CCE3EEP/7CCEMEEP

Individual Project Submission 2021/22

Name: Matteo Liguori

Student Number: K19044407

Degree Programme: General Engineering

Project Title: Comparative Analysis of Sensor Fusion Techniques for State Estimation in Autonomous Navigation of a Self-Balancing Bicycle

Supervisor: Francesco Ciriello

Word count: 9581

RELEASE OF PROJECT

Following the submission of your project, the Department would like to make it publicly

☒ I **agree** to the release of my project.

☐ I **do not** agree to the release of my project.

RELEASE OF VIDEO

Following the submission of your project demonstration, the Department would like to

☒ I **agree** to the release of my project video.

☐ I **do not** agree to the release of my project video.

Signature: Matteo Liguori

Date: 08/04/2023

Table of Contents

Table of Figure

I verify that I am the sole author of this report, except where explicitly stated to the contrary.

Abstract

In this project, we developed a closed-loop controller for a self-balancing bicycle that can autonomously navigate towards waypoints. The system was designed to operate using noisy wheel encoder and IMU sensors, which were used to estimate the bicycle's state in real time. We implemented two different state estimation algorithms based on odometry and IMU data. We then combined their results using two sensor fusion techniques: the Extended Kalman Filter (EKF) and a weighted average method.

Our experimental results showed that the sensor fusion approach was effective in improving the accuracy and reliability of the state estimation process. The EKF method produced better results than the weighted average method, particularly in cases where the sensors were highly noisy or where the bicycle was subjected to sudden changes in its motion.

Overall, our project demonstrated a viable approach for constructing an autonomous navigation system for a self-balancing bicycle, using readily available sensors and state estimation techniques. The results of this project could have practical applications in fields such as transportation and robotics, where accurate and reliable control of dynamic systems is essential.

Nomenclature and Reference Frames

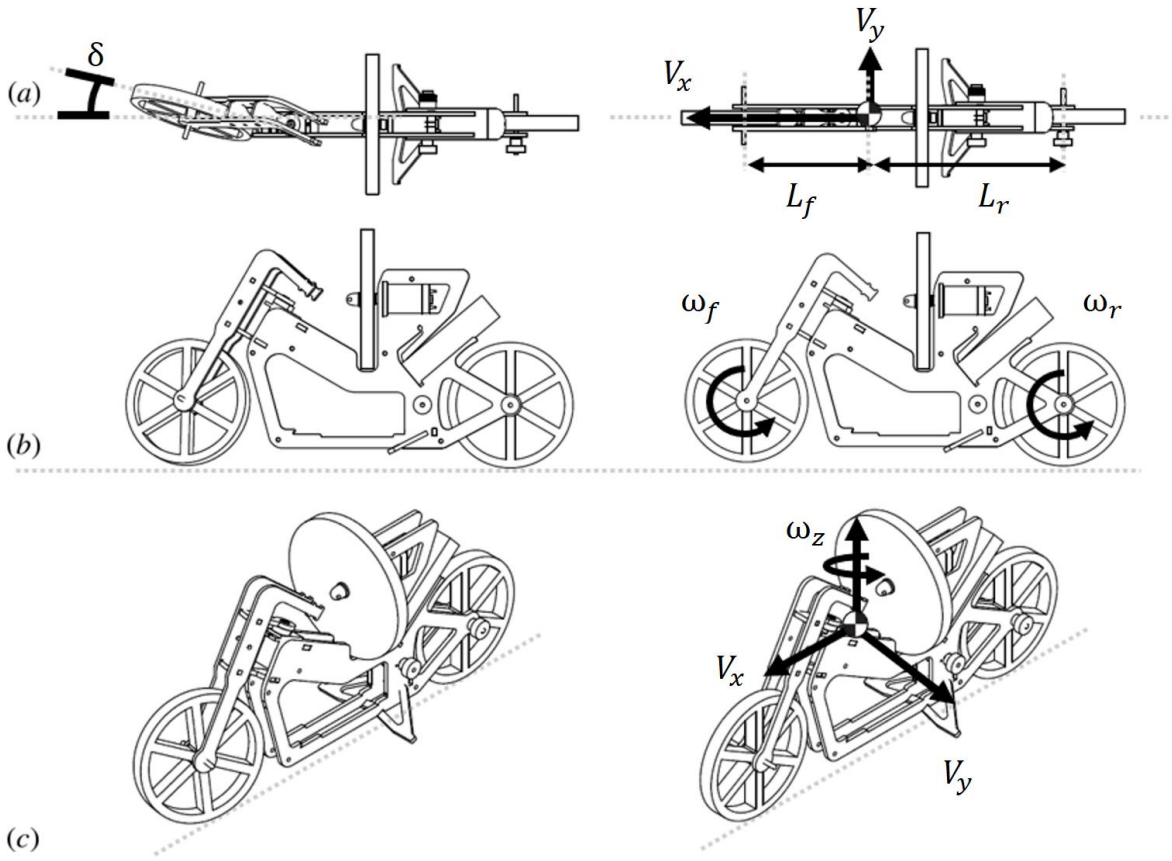


Figure 1 Diagram of bicycle axes and frame of reference.

Symbol	Meaning
A, B, C	State space model matrices
x	State vector
y	Sensor output vector
u	Input vector
w	Process noise random variable $w \sim N(0, Q)$
v	Measurement noise random variable $w \sim N(0, Q)$
k	Used to indicate current timestep, or to represent stiffness
P	Covariance Matrix
K	Kalman Gain

In the context of the Extended Kalman Filter, the $\hat{}$ superscript indicates that the variable is a prediction, and the $^-$ superscript refers to the variable being an a priori value.	
$a_x, a_y\ a_z$	Used to describe the linear acceleration of the bike along the x y z axis
v_x, v_y, v_z	Used to describe the linear velocity of the bike along the x y z axis
p_x, p_y, p_z	Used to describe the position of the bike at the canter of mass
$\omega_x, \omega_y, \omega_z$	Used to describe the angular velocity of the bike
$\theta_x, \theta_y, \theta_z$	Represent the angular displacement. θ_x gives the lean angle of the bike, while θ_z gives the angle of the bike way from the positive x axis on the $x-y$ plane
δ	Bicycle steering angle
ω_r, ω_f	Angular velocity of rear and front wheels, respectively.
L	The distance between front and rear wheel of the bicycle
R	The wheel radius of the vehicle.
d	Damping ration
l	Distance

1. Introduction

The development of autonomous control systems for dynamic systems has gained increasing interest in recent years due to the potential applications of such systems in areas such as transportation, robotics, and assistive technologies. Autonomous close-loop control systems are a fundamental part of the engineering required for applications such as autonomous driving, industrial robotics and more. (Antsaklis, Passino and Wang 1991)

Self-balancing bicycles are a particularly challenging system to control autonomously due to their unstable nature and have thus been a focus of recent research in the field. Controlling a self-balancing bicycle involves maintaining its balance, while also navigating towards a target location. This requires a precise and reliable state estimation process that takes into account the noisy and uncertain nature of the sensor data.

I. Motivation

I became personally interested in pursuing this project due to a previous project I worked on in my second year at King's College. In this project we worked in teams to come up with a very rudimentary controller for a small self-balancing bicycle from the Arduino Engineering Kit. Our team managed to achieve close loop balancing of the bicycle and very basic open loop control. This project really got me interested in control and motivated me to focus on this topic for my thesis. I also applied and was accepted to work as a TA (Teaching Assistant) to assist my supervisor Francesco Ciriello in delivering the class this year.

Figure 2 Arduino Bikes

Figure 2 shows the bicycles, after we finished setting up the room for one of the workshops.

Both of these experiences allowed me to begin to build the interest technical capabilities required to pursue my individual project.

II. Project Goals

In this section, we will outline the goals of the project. Our aim was to create a high-fidelity model of a bicycle in software, which would enable us to study the system of the bicycle. This required the creation of a CAD model of the hardware, which was imported into Simulink and animated using the Simscape Multibody library. A basic version of this model was already created by a collaboration between MATLAB and Arduino, and was modified as required during the course of the project.

Additionally, we aimed to develop a good closed-loop controller architecture that would allow the robot to autonomously complete waypoint missions. This required the robot to accurately keep track of its state and use that knowledge, along with the relative locations of its objectives, to give commands to the actuators in order to remain upright and navigate towards objectives. To achieve this, we controlled the revolute joints of the rear wheel, flywheel, and steering within the context of the simulation.

We also aimed to employ modern and effective state estimation and sensor fusion techniques. Accurate state estimation is fundamental for a system relying entirely on local reference frame sensors, such as IMUs and wheel encoders, as these sensors can be noisy and are prone to drift. We aimed to implement independent state estimation using two distinct methods: odometry and IMU readings to obtain an estimated vehicle pose. We then aimed to implement an extended Kalman filter to fuse these two estimates to gain a more accurate state estimation. We compared the performance of the extended Kalman filter against a simple weighted average scheme for pose estimation.

2. Background

I. Self-Balancing Bicycle

The project was based on the Arduino engineering kit self-balancing bicycle. It is a two wheeled robot which can balance using a flywheel controlled by a dc motor monitored by a rotary hall effect encoder. It uses a servo motor to move the front handlebar and another dc motor equipped with a hall effect encoder to drive the back wheel. (Arduino 2023)

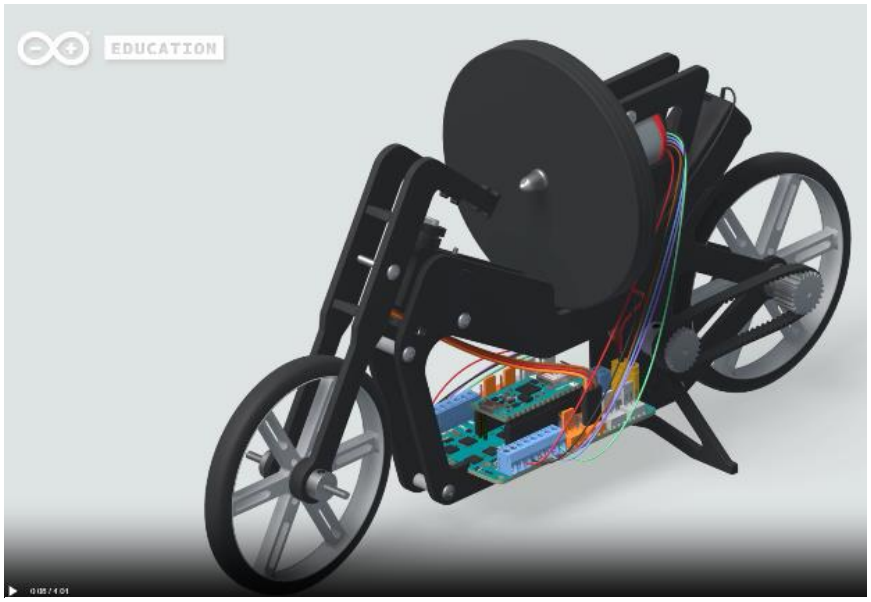


Figure 3 Diagram of Physical Bicycle Robot, from the Arduino Engineering Kit Website

A Simulink model was created based on the hardware, using the Simscape Multibody MATLAB library, to create a dynamics model of the bicycle. This was then used to build and test the close loop controller and the sensor fusion.

Specifications of relevant hardware:

The IMU used in the hardware version of the

bicycle is the BNO055 IMU. This sensor is produced by Bosh, and integrates a triaxial 14-bit accelerometer, a triaxial 16-bit gyroscope, a triaxial geomagnetic sensor, and a 32-bit cortex M0+ microcontroller running Bosch Sensortec Sensor Fusion. The sensor outputs fused data such as Quaternion, Euler angles, Rotation vector, Linear acceleration, Gravity, and Heading. (Bosch 2021) The sensors' accuracy and performance were kept in mind when designing the simulated sensors in the model.

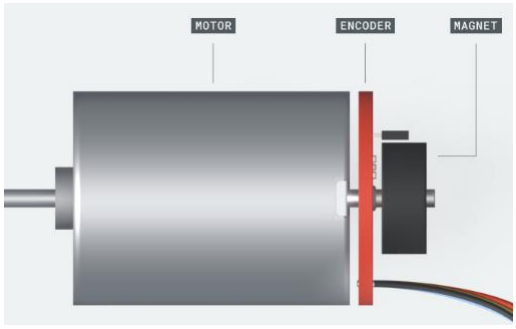


Figure 4 DC Motor Diagram with Integrated Encoder, from Arduino Engieniering Kit Website

The bicycle was also equipped with DC motors with integrated rotary encoders; which rely on the hall effect to detect the angular position of the motor shaft. These encoders were able to give 48 ticks per full rotation. The diagram in Figure 4 shows a schematic of the DC motors with integrated rotary encoders. The discontinuous nature of wheel position data is a challenge that state estimation systems need to address, and thus similar limitations were included in the software model. (Arduino 2023)

II. State Estimation Techniques

Accurate state estimation is a critical component of controlling dynamic systems such as self-balancing bicycles. State estimation involves estimating the system's internal state based on the available sensor data. In our case, this involves estimating the orientation and position of the bicycle.

There are several different state estimation techniques that can be used, including odometry-based and IMU-based methods. Odometry-based methods involve using wheel encoders to estimate pose of the vehicle. The pose is the current position and orientation of the vehicle. Wheel encoders provide information about the angular position of a wheel, combining this information with the current steering angle and the specific geometry of the vehicle, it is possible to obtain the pose of the vehicle. Odometry-based methods are typically accurate but can be subject to drift and errors due to wheel slip and other sources of measurement noise.. IMU-based methods, on the other hand, involve using the information provided by IMU's (Inertial Measurement Units) to obtain the vehicle pose once again, this is done essentially by integrating the acceleration and angular velocities provided by the sensor to obtain position and orientation. These methods are also accurate but can be subject to errors due to sensor drift and noise. (Moon 2020) (CHONG and KLEEMAN 1997)

In this project, we implemented both odometry and IMU-based state estimation methods and used a sensor fusion approach to combine their results. The goal of using sensor fusion was to improve the accuracy and reliability of the state estimation process by taking advantage of the complementary information provided by the two methods. By combining the results of the odometry and IMU-

based methods, we were able to reduce the effects of measurement noise and drift and obtain a more accurate estimate of the bicycle's state.

III. Sensor Fusion Techniques

Sensor fusion is the process of combining data from multiple sensors to obtain a more accurate and reliable estimate of the system's state. In our case sensor fusion was used to combine the results of odometry and IMU-based state estimation methods to improve the accuracy and reliability of the state estimate.

One of the most widespread and popular techniques for sensor fusion and filtering is the Kalman filter. The Kalman filter uses a mathematical representation of a dynamical system, known as a state space model, in the form:

$$x_k = Ax_{k-1} + Bu_k + w_k$$

$$y_k = Cx_k + v_k$$

(1)

Where x_k is the current state vector and x_{k-1} is the previous state vector, A is the system matrix, B is the system matrix, u_k is the input vector, w_k $w \sim N(0, Q)$is the process noise, y_k is the system output vector, C is the output matrix and v_k ($v \sim N(0, R)$)is the measurement noise.

This is based on the vehicle model:

$$\hat{x}_k = A\hat{x}_{k-1} + Bu_k$$

$$\hat{y}_k = Cx_k$$

(2)

The hats on the state vector and the output vector represent the fact that this is an estimate due to the uncertainty caused by the noise.

It is important to keep in mind the sensors in this model cannot directly measure the state, rather they monitor some other value, and then rely of the characteristics of the system to infer the state.

The Kalman filter is a recursive algorithm which operates in two distinct stages:

1- Prediction

The algorithm uses information about the vehicle previous state and the physics of the system to calculate the new a priori estimate of the new state.

$$\hat{x}_k^- = A\hat{x}_{k-1} + Bu_k$$

(3)

Note that the superscript minus is used to indicate that an estimate is calculated based on a priori logic.

It also predicts the value of P_k^- which is the covariance matrix of the estimates state. In other word, it provides an estimate of the uncertainty in each state estimates.

$$P_k^- = AP_{k-1}A^T + Q$$

(4)

2- Update

In the update step, the algorithm uses the a priori state prediction to update the a posteriori state $\hat{x}_k$ and error covariance P_k . To do so it incorporates indirect measurements of the state y_k as well. To determine how much to trust the a priori estimate versus the indirect measurement, the algorithm uses the Kalman gain K_k.

$$K_k = \frac{P_k^- C^T}{C P_k^- C^T + R}$$

$$\hat{x}_k = \hat{x}_k^- + K_k(y_k - C\hat{x}_k^-)$$

$$P_k = (I - K_k C)P_k^-$$

(5)

The Kalman gain is calculated in such a way as to minimize the process covariance matrix P_k, thus ensuring that the filter always provides the best possible estimate given the available information. Multiple different sensors can thus be combined to provide a unified and state estimate. (Kalman 1960) (MATLAB 2017)

The extended Kalman filter builds on these concepts, but does not rely on the liner state space model, but rather assumes that the system is described by:

$$x_k = f(x_{k-1}, u_{k-1}) + w_{k-1}$$

$$y_k = g(x_k) + v_k$$

(6)

Where f and h are both continuous and differentiable. The EKF (Extended Kalman Filter) computes the Jacobians f and g functions to linearize the model around the mean of the current state estimate.

$$F = \left.\frac{\partial f}{\partial x}\right|_{\hat{x}_{k-1},u_k}$$

$$G = \left.\frac{\partial g}{\partial x}\right|_{x_k}$$

(7)

Once we obtain the Jacobians, we are able to generate a good linear approximation of the system around the current timestep and proceed in much the same way as with the standard Kalman filter. (Choset 2005)

IV. *Close Loop Controllers*

Closed-loop control systems are a type of control system that use feedback to adjust the system's behaviour in response to changes in the system's state or environment. In the case of self-balancing bicycles, closed-loop control systems can be used to adjust the position and orientation of the bicycle to reach a desired target.

There are several different types of closed-loop control systems that can be used for self-balancing bicycles, including proportional-integral-derivative (PID) controllers, linear quadratic regulators (LQRs), and model predictive controllers (MPCs). PID controllers are a common type of closed-loop control system that use feedback from the system's sensors to adjust the system's behaviour. LQRs and MPCs, on the other hand, are more advanced types of closed-loop control systems that use mathematical models of the system's behaviour to optimize the control actions.

In this project, we implemented a closed-loop control system for the self-balancing bicycle using a modified PD controller to control the balancing, and a pure pursuit trajectory tracking algorithm cupped with inverse kinematics to control steering and vehicle speed to move towards a desired target. The controller was designed to be robust to external disturbances and to handle the nonlinear dynamics of the self-balancing bicycle.

V. Related Work

Autonomous bicycle balancing and control is a popular area of research in control engineering, as it offers some unique challenges due to the inherent instability of the system. Two particular papers stand out, although they are not unique. In the first, a team from Asian Institute of Technology in Thailand worked on a full-size bicycle. They explore the use of a structure specified mixed H2/H∞ controller design using particle swarm optimization (PSO) instead of a more typical PD controller to control balancing of Bicyrob. (Thanh 2008). Another interesting paper on the topic was published by a Swedish team. Their goal was to come up with a control strategy based entirely on

changing the front wheel angle of the bicycle to balance it when it was already moving forwards. This of course meant that the bike was unable to balance when at rest, but the author argued that their stabilization system was much more energy efficient than design which relied on extremely fast turning gyroscopes or shifting heavy weights. (GRÖNLUND 2018)

These papers, among others in the field of control and robotics guided us towards the design choices which we ultimately settled on.

3. Modelling and Control System Design

I. Dynamic Model

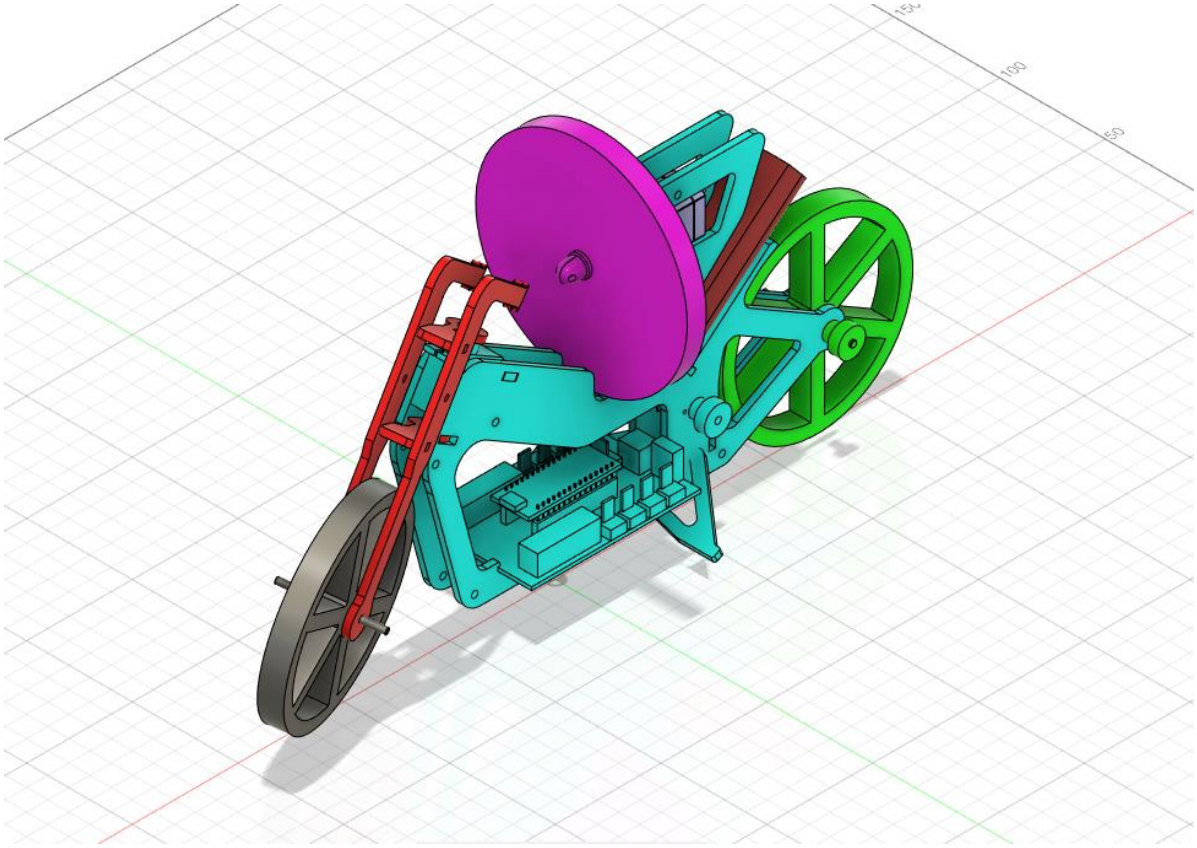


Figure 5 Fusion 360, Bicycle CAD Model

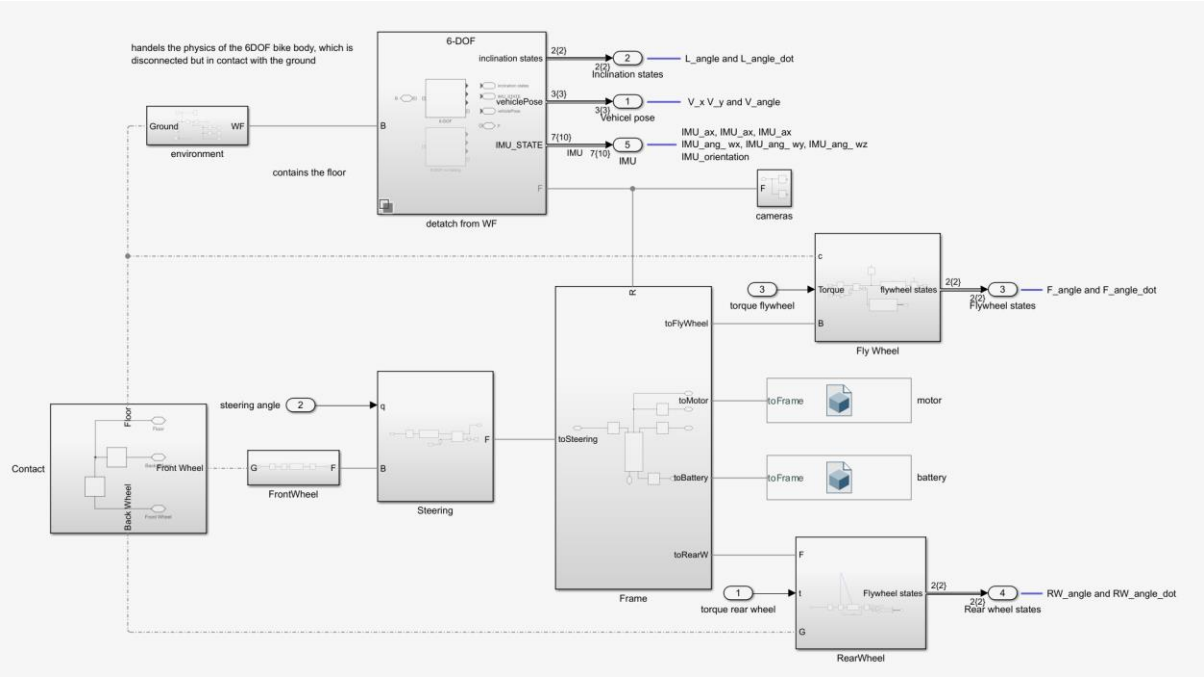


Figure 6 Simulink Diagram, Bike Model

The development of comprehensive equations for a detailed dynamical model of the bicycle was beyond the scope of this project. Instead, the Simscape Multibody Library was utilized to define the system, based on an accurate CAD model of the bike developed through collaboration between Arduino and MATLAB (Figure 5). The CAD model comprised six basic parts: battery, flywheel, flywheel motor, rear wheel, front wheel, frame, and steering frame, which were attached to the frame using rigid joints and revolute joints and a series of rigid body transforms. The frame was modelled as a six degree of freedom joint, allowing for rotation and translation in all directions, enabling the bicycle to move in the simulation. The revolute joints between the frame and rear wheel, frame and steering frame, and flywheel motor and flywheel were modified to accept torque inputs to drive them based on control signals from the controller. Figure 6 shows an overview of how the bike components were assembled in Simulink.

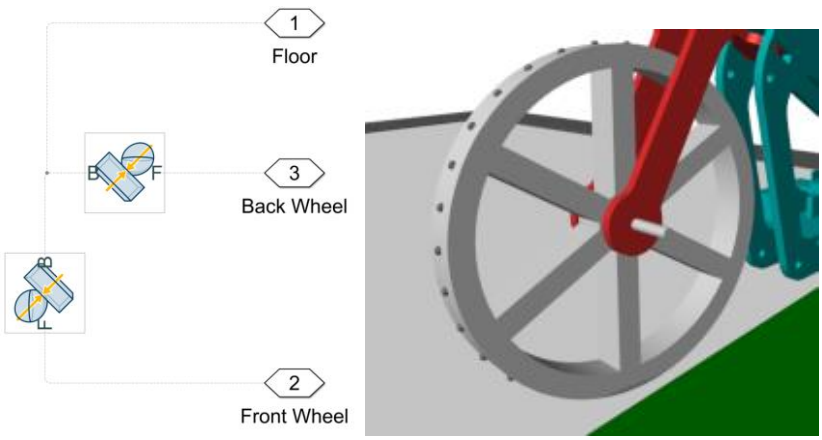


Figure 7 Simulink Model, Contact modelling between floor and wheels

A spatial contact force model was used to simulate the contact between the wheel and the ground, with normal forces calculated using a Smooth Spring-Damper model ($k = 20\ kNm^{-1}, d = 200\ N(ms^{-1})^{-1}, l = 0.01\ m$), and frictional force calculated using the Smooth Stick-Slip model ($\mu_s = 0.7, \mu_d = 0.5, v_c = 0.0001\ ms^{-1}$). Initially, contact forces were calculated using a cylinder to model the wheels, but this significantly slowed the simulation due to the computational expense of computing contact between two surfaces. To address this, a more efficient approach was implemented. A point cloud was generated to match the centreline of the external wheel edge and was used to simulate contact with the ground. As the simulation utilized a perfectly flat surface for the ground, this provided a reasonable approximation of the wheel-ground interaction behaviour while significantly decreasing computing time when a sufficient number of points was used (40). This can be seen in Figure 7.

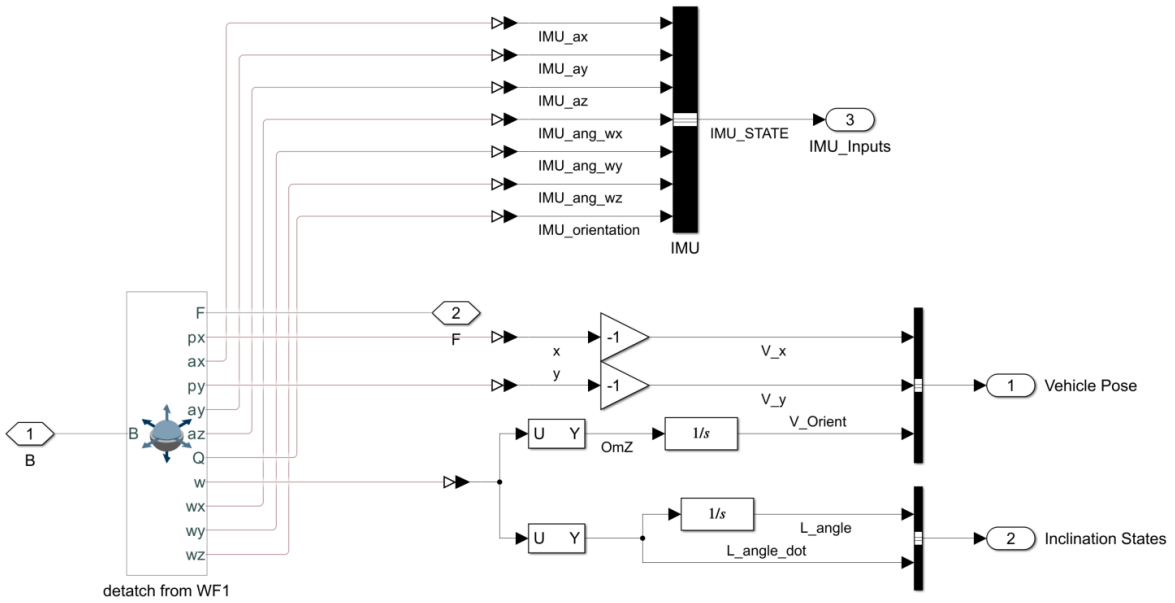


Figure 8 Simulink Model, 6DoF Subsystem

In order to obtain the state of the bicycle the revolute and six degree of freedom joints were modified so they outputted their state. An example of this can be seen in Figure 8, which show is the six degrees of freedom joint, and how data relating to the position, orientation and acceleration of the joint is extracted and processed to obtain the bicycle state vector. In total nine variables are kept track of in this way, these represent the state of the vehicle.

Model Buss Name	Bus Content Model name	Description
V_Pose_G	*V_x, V_y, V_Orient*	This contains, p_x, p_y *and* θ_z as describe above
L_State_G	*L_angle, L_angle_dot*	This contains θ_x *and* ω_x
F_State_G	*F_angle, F_angle_dot*	The angular position of the flywheel and its derivative
RW_State_G	*RW_angle, RW_angle_dot*	The angular position of the rear wheel and its derivative

This approach to vehicle dynamic modelling is much simpler and easier to implement then writing the full dynamics equations for the system. Perhaps the largest difference between the real-world bike and a simulation are the motors, as in the simulation the joint torques are directly controlled by the controller, and thus they can change instantly and have infinite power. This is of course very unrealistic in general; however, at slow speeds and by appropriately tuning the controller this remains mostly realistic.

The world frame and mechanism configuration blocks are used to define the global coordinate system as well as the physics of the simulation.

II. Controller

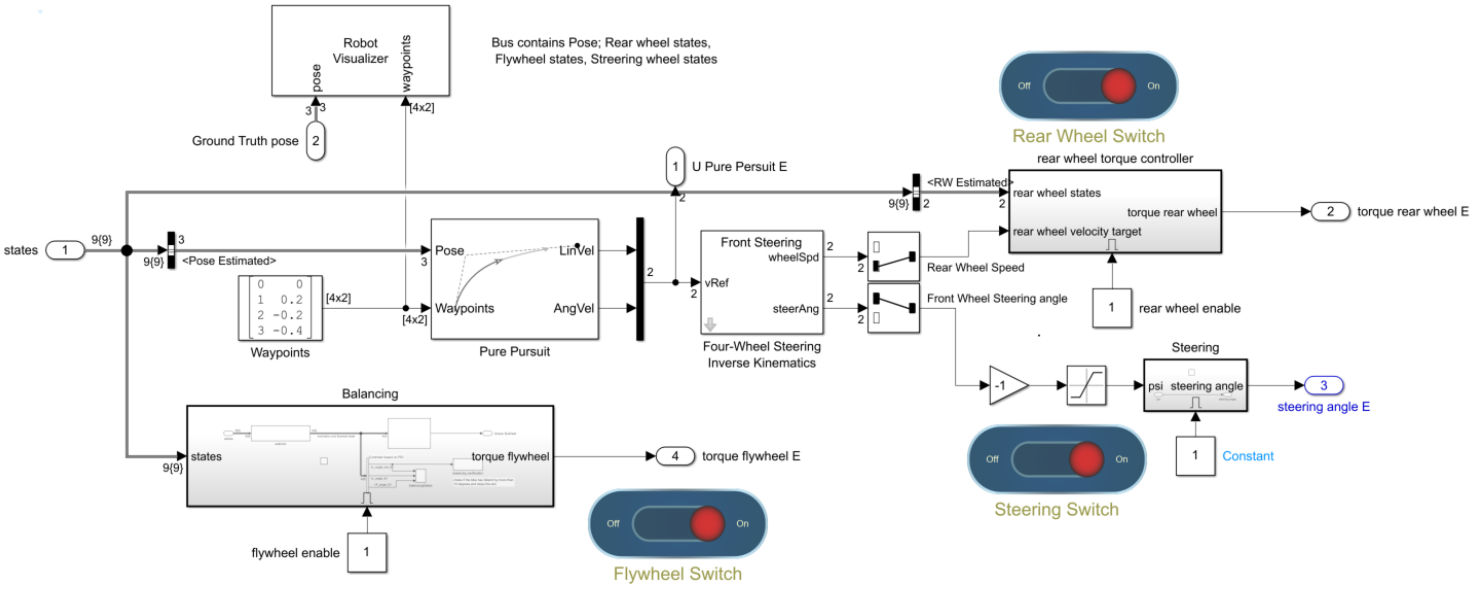


Figure 9 Simulink Model, Controller overview

The controller architecture utilized in this project (depicted in Figure 9) is composed of two distinct components: the balancing controller and the path planning controller. The separation of these components serves to simplify the overall complexity of the controller design. The balancing controller is responsible for maintaining the lean angle and its rate of change at 0, which allows the bike to remain upright regardless of its desired direction. Consequently, the bike is unable to lean into turns, which may be problematic at high speeds, but this is not a concern at low speeds. We

hold that this design choice is a reasonable simplification for the scope of this project and still enables good controller performance.

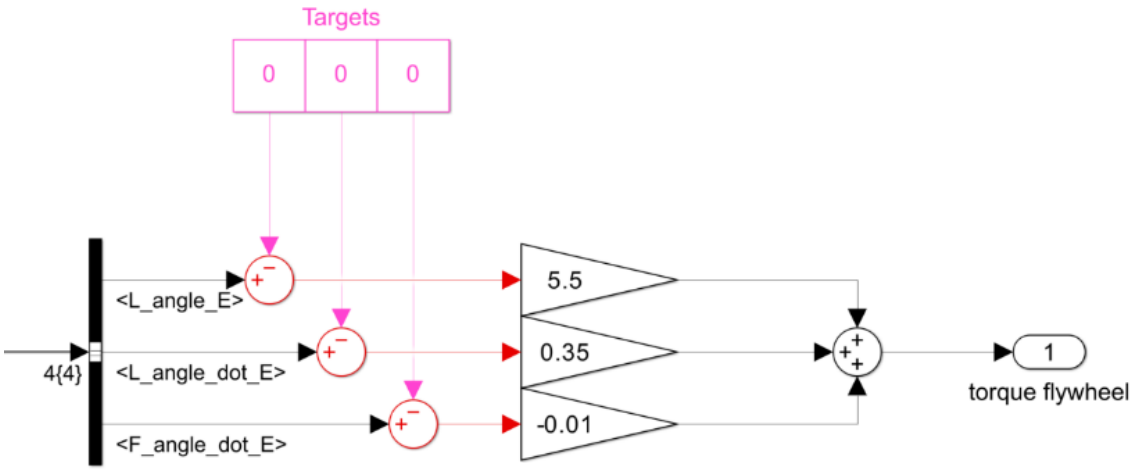


Figure 10 Simulink Model, Balancing Controller

Error! Reference source not found. shows the balancing controller, this is a modified PD controller, which includes a third term, which refers to the angular velocity of the flywheel. Adding this term leads to much lower controller signals, which means that the bike is able to balance at lower flywheel angular velocities which reduces vibrations and saves energy. This is because in this system, balancing is achieved not by exploiting gyroscopic effects, but rather from the change in momentum of the flywheel which when accelerated imposes an opposite torque to the frame of the bike used to balance it.

The use of this controller system was inspired in part by previous work on the topic of flywheel stabilized inverted pendulums. (Olivares 2014) (Albertos 2018) This is because the bicycle system, for the purposes of maintaining an upright orientation can be modeled as an inverted pendulum. This is also the reason why the controller could be split into parts.

The second half of the controller is much more complex, as unlike the modified PID system, which only has one output, to control the flywheel motor, the path planning controller needs to provide control signals for both the rear wheel and the steering wheel.

The first challenge is to produce a way to determine, from any given position, how to reach a waypoint. This is quite a common problem in robotics and has many workable solutions. Amongst these are Pure Pursuit, Stanley's Method, LQR control, model predictive control (MPC) and artificial potential fields. Despite more recent methods such as MPC and LQR control showing better performances in similar circumstances (Liu 2021), the Pure Pursuit algorithm (Coulter 1992) was chosen in this case, due to its simplicity and robustness, especially for low speeds and smooth curves, and because of its very low computational cost. The Pure Pursuit algorithm takes the current position and orientation of a vehicle or robot, as well as a list of waypoints that define the

desired path. The algorithm first creates a continuous trajectory by linearly interpolating between the waypoints. It then calculates the closest point on the path to the vehicle's current position, known as the lookahead point, and determines a pursuit point that is a fixed distance ahead of the lookahead point, called the lookahead distance.

Using the pursuit point and the vehicle position, the algorithm generates a smooth curve that leads the vehicle towards the pursuit point. The algorithm calculates the necessary angular velocity and linear velocity to intercept the pursuit point. These control inputs are used to guide the vehicle towards the pursuit point.. As the vehicle moves, the pursuit point slides along the path, staying a fixed distance ahead of the vehicle, and the vehicle chases the pursuit point by adjusting its steering and velocity based on the pursuit point's location. (Coulter 1992)

To go form angular velocity ω_z and the linear velocity v_x obtained by the pure pursuit algorithm to the target values for steering δ, and rear wheel velocity (ω_r) inverse bicycle kinematics were used. The steering angle δ and the angular velocity of the rear and front wheels ω_r, ω_f can be obtained using the Ackermann steering model.

$$\delta = \operatorname{atan}\left(\frac{\omega_z L}{v_x}\right), \quad \omega_f = \frac{v_X}{R\cos\delta}, \quad \omega_r = \frac{v_X}{R} \tag{8}$$

This assumes no wheel side slip is present.

As in our case we are only driving the back wheel, we ignore the ω_f result. The last step is to calculate the error between the target rear wheel velocity and the measured rear wheel velocity, which is measured by the rotary encoder on the rear wheel motor. This is then fed through a standard PD controller. As the steering motor does not return encoder values, this is only done for the rear wheel motor.

To summarize the controller architecture is divided into two part, the first only takes care of balancing the bike along the x axis, and is handled by a modified PD controller which directly controls the flywheel motor, the second handles the path planning and tracking, and is accomplished by feeding the output of a pure pursuit algorithm into the inverse kinematic equations of the bicycle to obtain the desired rear wheel velocity and steering angle.

III. *Sensor* Noise Simulation

As we have seen earlier, we extract 9 state variables from the dynamic system model, these are used as ground truth values in the simulation, as they give the real state of the bike. This however is highly unrealistic, as real-world systems are monitored by error prone and noisy sensors.

In our case, the system relies on three main sensors to estimate the state of the bicycle: the Rotary wheel encoders on the flywheel motor and the rear wheel motor, and the IMU (inertial measurement unit). In this section we outline how we built our sensor models. These models take in ground truth information, and then apply noise and drift to output realistic measurement data for the sensor being modeled.

a. Rotary encoder simulation

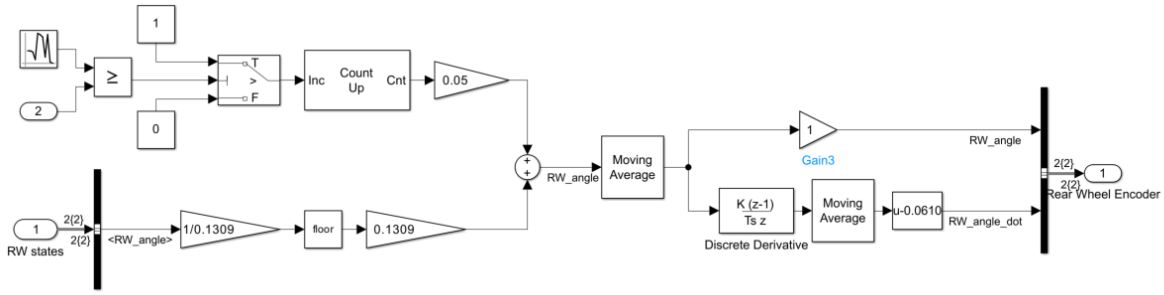


Figure 11 Simulink Model, Rotary Encoder Simulation

Figure 11 shows the model for a Rotary encoder. This model encapsulates two of the main sources of noise in encoder readings. The first is the discrete nature of the measurements. In our case the Rotary encoder has forty-eight distinct states, which indicate the angular position of the wheel. This means that our readings from the encoder will not be continuous, but will instead take the form of a staircase, with discrete steps corresponding to each of the 48 distinct states of the encoder. In the model this is achieved by using the floor around function, in order to hold the discretized angular position of the wheel in between encoder ticks. The second main source of uncertainty in encoder readings is drift. This may be due to a variety of reasons, such as changes in the wheel diameter, due to tire pressure, or wheels lip. In the model this is simulated using a gaussian random number generator ($N \sim No(0,0.16)$) . At each time step, we check whether the random number generated is higher than threshold value, if so, we increment a counter. The scaled output of that counter is added to the current angular position, thus simulating drift.

b. IMU sensor simulation

The IMU sensor is a much more complex sensor than the rotary encoders. The IMU used in the hardware version of the bike, is a complicated system, with four main sensors which are used and combined to provide accurate measurements. Fully simulating this type of sensor accurately is beyond the scope of this project, instead a simplified IMU sensor model was devised and implemented. While the BNO055 IMU outputs Quaternions, Euler angles, Rotation vector, Linear acceleration, Gravity, and Heading. (Bosch 2021) The bike controller does not require all of these. Only accelerations in the x and y axis as well as angular velocity and position (θ_x, ω_x and θ_z, ω_z)

are necessary. Thus, only these four measurements were simulated as coming from the IMU model. IMU's are prone to two distinct error types, random noise, and drift.

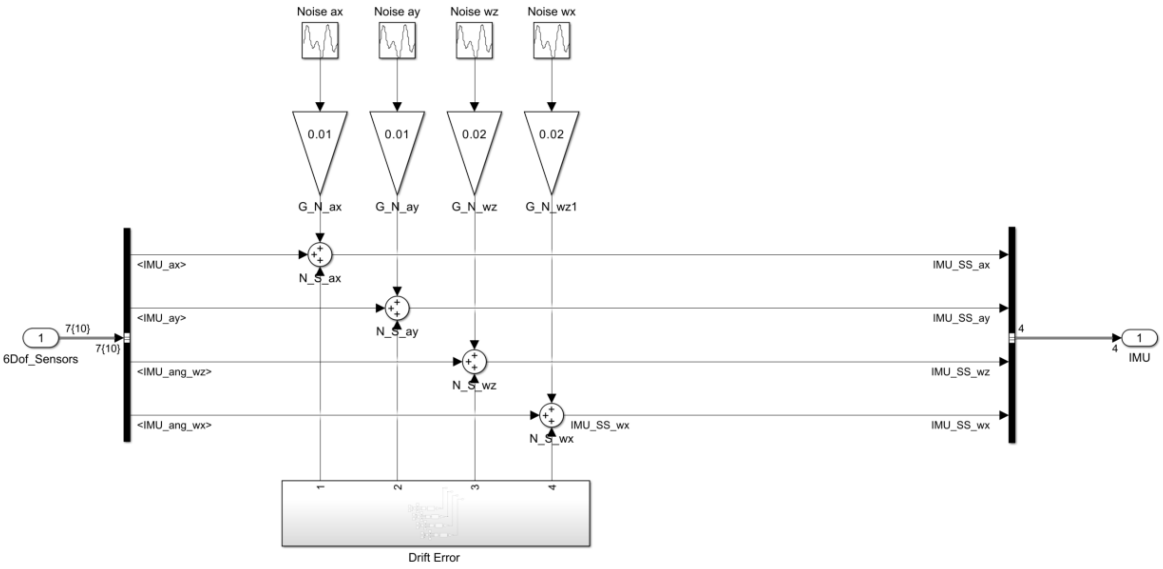


Figure 12 Simulink Model, IMU simulator

Figure 12 Shows the IMU simulator block: the relevant ground truth signals from the 6DoF block, are extracted using a bus selector; then random error produced by noise functions (for the accelerometer signals $N \sim No(0,0.5)$, while for the angular velocity signals it is $(N \sim No(0,0.1)$) and scaled appropriately is added to the signal, along with drift noise, which is generated in the drift error subsystem, using a similar technique as described in the previous section.

IV. *State* estimation and Sensor Fusion

To recap so far, we have specified a controller design that takes information from three sensors on the bike: the Rotary wheel encoders on the flywheel motor and the rear wheel motor, and the IMU. Within the simulation the information is actually obtained from the dynamic model of the bike. These ground truth sensor readings are then modified to add realistic measurement noise. However, only relying on the noisy sensor readings is not sufficient to achieve a viable controller architecture. This is where sensor fusion and state estimation come in. Their goal is to take the sensor readings and combine and filter them in an effort to reduce noise and increase the accuracy of the state estimation.

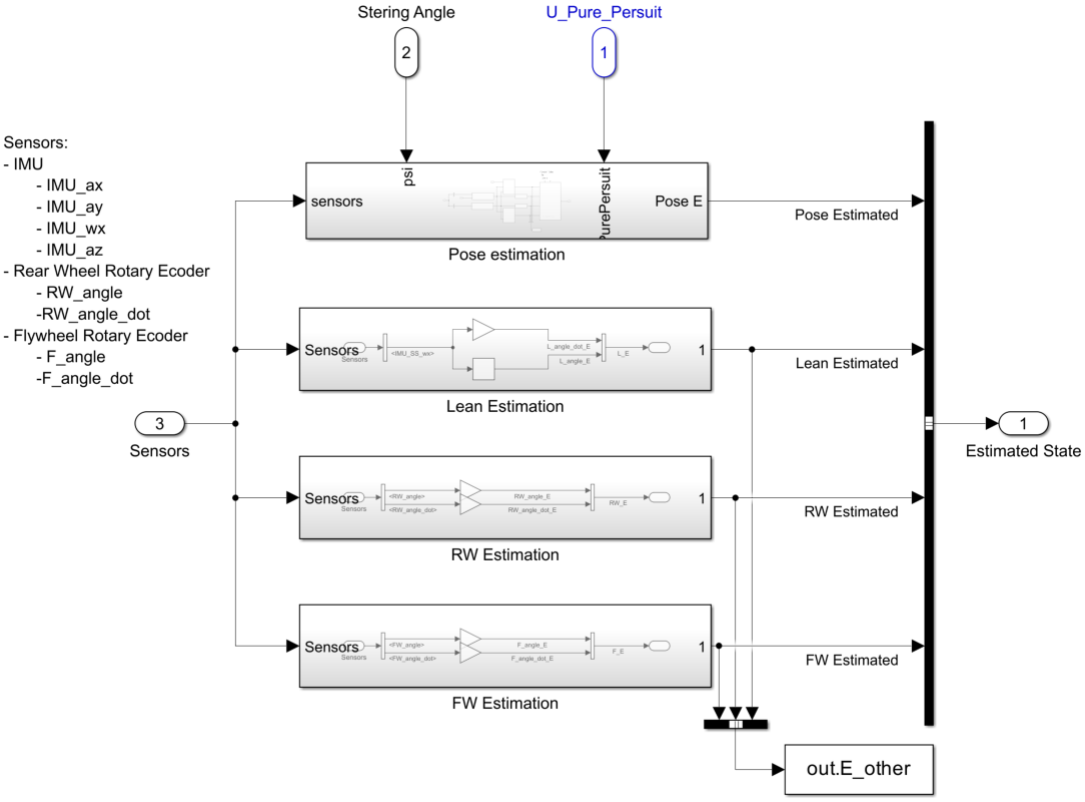


Figure 13 Simulink Model, Overview of State Estimation Subsystem

The state variables monitored and used by the controller can be divided in two main categories, those which are only measured by a single sensor and cannot be easily derived from the other sensors, and those which can be independently calculated from data from a least two sensors. More specifically lean estimation and the angular position and velocity of the flywheel and rear wheel all come directly from the sensors, the way these signals are smooth and calculated will be discussed later.

The focus of this section is rather on the pose estimation; as pose (p_x, p_y, θ_z) can be obtained with two distinct methods, by integrating the output of the IMU, or from odometry. Once a pose estimation is obtained using both methods, the output can be combined using sensor fusion techniques, to further refine the prediction and reduce error and drift in the pose estimation.

a. IMU based pose estimation:

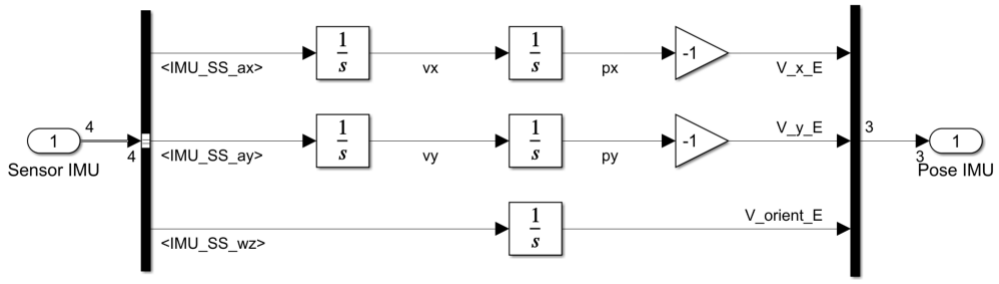


Figure 14 Simulink Model, IMU based state estimation.

Figure 14 shows the subsystem responsible for taking noisy data from the IMU and integrating it to obtain the pose. Especially in the case of position estimation, we expect the integration process to amplify the effects of noise and drift.

$$p_x = \iint_{t=0}^{t=T_{final}} a_x dt \qquad p_y = \iint_{t=0}^{t=T_{final}} a_y dt \qquad \theta_z = \int_{t=0}^{t=T\_final} \omega_z dt$$

(9)

b. Odometry Based Pose Estimation:

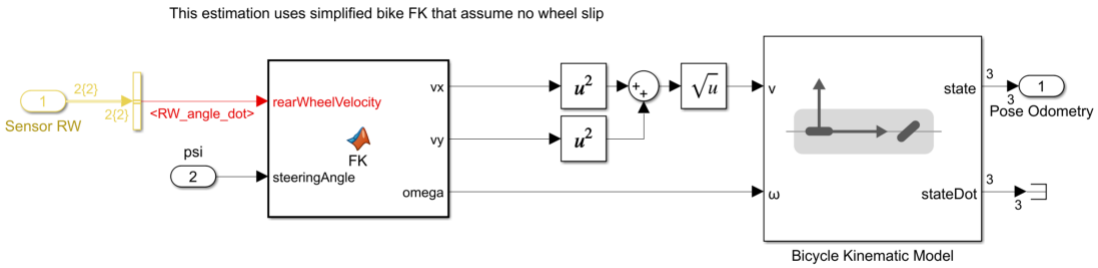


Figure 15 Simulink Model, Odometry Based State Estimation

The odometry state prediction is based on the rear wheel angular velocity and the steering angle. We can obtain the state of the bicycle using a standard bicycle kinematic model; however, we first need to obtain the velocity vector of the bicycle along the XY plane, and the bike's angular velocity ω_z.

Beginning from the geometry of the bicycle, which was derived in a similar fashion to the work done in this paper (Bonnabel 2019), but accounting for the fact that in our case the wheels are not in line, and only one is powered, we obtain the following equations:

$$v_x = \frac{R\omega_{RW}}{2}(1 + \cos(\delta))$$

$$v_y = \frac{R\omega_{RW}\sin(\delta)}{2}$$

$$\omega_z = \frac{R}{L}(\omega_{RW}\sin(\delta))$$

(10)

To obtain the velocity vector, the $x-y$ components of velocity are combined using Pythagoras.

$$v = \sqrt{v_x^2 + v_x^2} = \frac{\sqrt{-R^2\omega_{RW}^2(\cos(S)-1)}}{\sqrt{2}}$$

(11)

Finally, we feed ω_z and v into the kinematic bicycle model:

$$p_{x_k} = p_{x_{k-1}} + v_k\cos(\theta_{x_k})$$

$$p_{y_k} = p_{y_{k-1}} + v_k\sin(\theta_{x_k})$$

$$\theta_{x_k} = \theta_{x_{k-1}} + \omega_{z_k}$$

(12)

And we finally obtain the full pose of the vehicle.

Note that this method of calculating pose relies on various assumptions; we assume a flat surface and no slipping or skidding. Within the context of the simulation, and at low speeds these assumptions are reasonable.

c. Sensor Fusion:

We now have derived two independent methods for state estimation, but these rely on noisy sensor data, in order to improve the quality of our state estimation filtering techniques and sensor fusion can be used. Our sensor fusion architecture is similar to the one employed in the paper "Multisensor Fusion Using Fuzzy Inference System for a Visual-IMU-Wheel Odometry" (Zhu 2021).

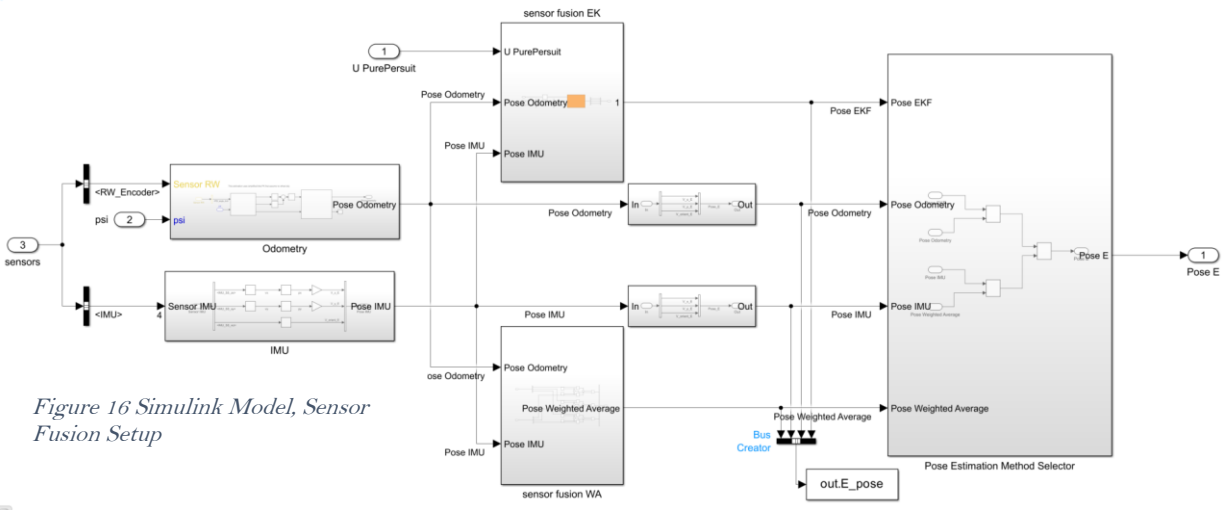


Figure 16 Simulink Model, Sensor Fusion Setup

As can be observed in Figure 16, two distinct methods for sensor fusion we are implemented. An extended Kalman filter (EKF) and a weighted average. The weighted average is included as a benchmark for the EKF algorithm.

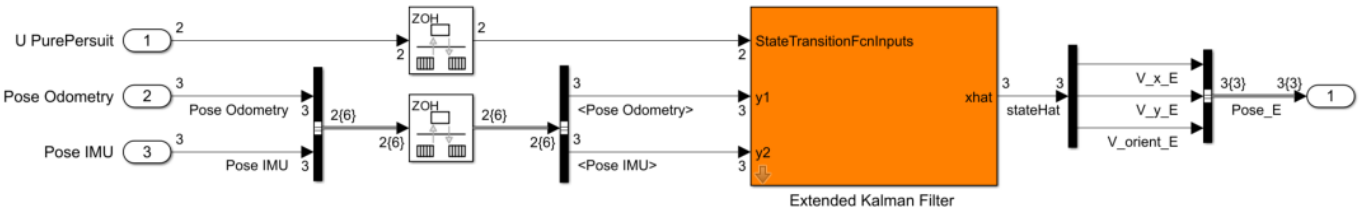


Figure 17 Simulink Model, EKF

As mentioned in an earlier section, the EKF Seeks to linearize the model around each prediction time step, to enable it to apply the linear Kalman filter predict-update cycle. From an implementation point of view, the main difference between a linear Kalman filter and the EKF, is that EKF requirees a specified dynamic function of the system. This is used by the EKF to predict future states based on the current state, and its knowledge of system dynamics. Because it is practically unfeasible to generate a perfect dynamic model, the EKF utilizes sensor-based state estimation to adjust its model-based prediction, and dynamically changes the contributions of each of these state predictions to the final prediction to minimize error.

Our system can be modelled as having two inputs and three states:

$$u = \begin{bmatrix} v \\ \omega_z \end{bmatrix}, x = \begin{bmatrix} p_x \\ p_y \\ \theta_z \end{bmatrix} \tag{13}$$

Using the Euler discretization method, the system dynamics can be approximated as

$$\begin{bmatrix} p_{x_{k+1}} \\ p_{y_{k+1}} \\ \theta_{z_{k+1}} \end{bmatrix} = \begin{bmatrix} p_{x_k} \\ p_{y_k} \\ \theta_{z_k} \end{bmatrix} + \Delta T \begin{bmatrix} v\cos(\theta_{z_k}) + \dfrac{\Delta T\,\omega_{z_k}}{2} \\ v\sin(\theta_{z_k}) + \dfrac{\Delta T\,\omega_{z_k}}{2} \\ \omega_{z_k} \end{bmatrix} \tag{14}$$

Where ΔT is the size between consecutive time steps, in our case this was set at 0.01. The EKF also requires the covariance matrices for sensor-based post predictions. These describe the uncertainty of the measurements. In this case, as our state vector has three elements, the covariance matrices will be three by three diagonal matrices.

As we have performed the transformations from raw sensor data to state prediction for the IMU and the odometer previously, the sensor functions for the Y1 and Y2 channels can simply be the state vectors produced by each method.

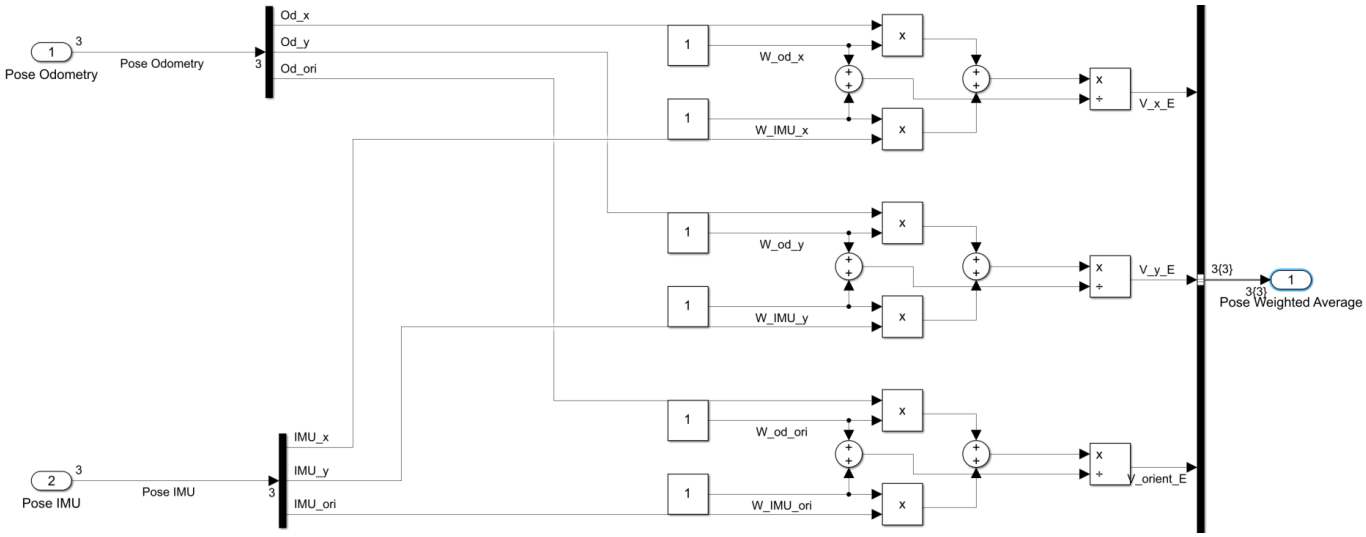


Figure 18 Simulink Model, WA Sensor Fusion

Figure 18 shows the subsystem responsible for calculating the weighted averages for the weighted average-based sensor fusion. This is a far less capable scheme compared to the EKF but should nevertheless provide some interesting results when analyzed. The figure above shows the system before tuning.

V. Tuning Methodology And Filtering

Parameter tuning is a vital component of closed-loop control system design. Below we outline a few examples of areas in the model which required parameter tuning, and the approach taken to finding appropriate parameters in these areas.

a. Encoder Noise reduction and Integration

Figure 11 showed the setup for simulating the output of the rotary encoder. As mentioned above the system captures both the jagged nature of Rotary encoder output as well as its tendency to drift.

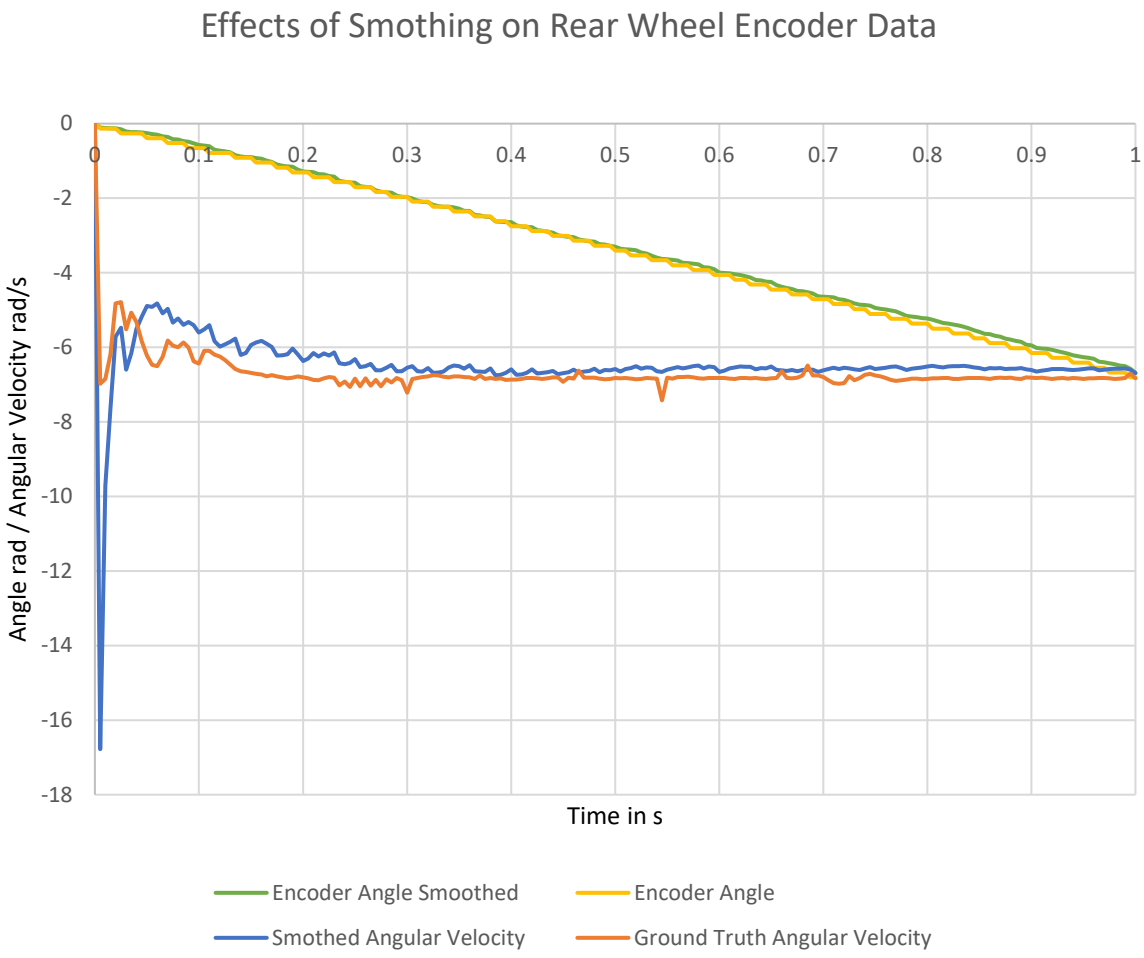


Figure 19 Plots of RW Angular Position and Velocity

The staircase like structure of encoder data create issues when attempted to find the discrete time derivative. A moving average filter utilizing an exponential weighting function was used to smooth the data to increase the accuracy of the derivative. The final exponential factor used was 0.999, this value was selected by inspection of the plot shown in Figure 19.

Figure 19 shows the ground truth and calculated values for angular velocity, obtained by computing the discrete time derivative of the signal. As seen in the graph, a steady state error is present. In an effort to attempt to correct this error a bias was added as can be seen in Figure 11. The value for the bias was calculated using a least square regression search method. This involves minimizing the sum of the squares of the difference between the ground truth values for angular position and the sensor simulation value by modifying the bias.

b. Weighted average Weights

In a similar fashion to what is described above the weights for the weighted average sensor fusion method were calculated utilizing a least squares optimization method. Three error terms were

calculated as the difference between each ground truth pose a state and the weighted average pose state.

$$E = \begin{bmatrix} e_1 \\ e_2 \\ e_3 \end{bmatrix} = \begin{bmatrix} p_x\ G\ T \\ p_y\ G\ T \\ \theta_z\ G\ T \end{bmatrix} - \begin{matrix} \frac{W_{11} * p_x\ IMU + W_{12} * p_x\ Odom}{W_{11} + W_{12}} \\ \frac{W_{21} * p_y\ IMU + W_{22} * p_y\ Odom}{W_{21} + W_{22}} \\ \frac{W_{31} * \theta_z\ IMU + W_{32} * \theta_z\ Odom}{W_{31} + W_{32}} \end{matrix}$$

(15)

The error terms were calculated at each time step of the simulation, once enough data was obtained the sum of the squares of the errors was found and used as the minimization target for the least squares algorithm. The path used to obtain the data used in this optimization was sinusoidal paths, along the x axis. This may mean that the WA pose estimate will be more accurate for paths similar to the one used to tune it.

c. EKF Covariance Matrices

The covariance matrices for the EKF were modified simply using trial and error to improve the overall accuracy of the filter. This also included looking at the error result plots for various runs. Despite the lack of a more formal method used, this still resulted and an improvement in the overall results.

4. Model Performance and Analysis

The analysis of the model performance was divided into three main sections, in order to give a holistic view of the performance of the approach described in the paper. We begin being analysing the controller performance.

I. Controller Response Analysis

We proceed with an analysis of the controller performance. As we are interested only in the performance of the controller in this section, the ground truth states of the bicycle are used, to ensure that the results are independent form the state estimation performance. As the controller architecture is split in two, the balancing a path planning portions of the controller will be analysed separately.

a. Path Planning controller

We begin with an analysis of the controller's step response.

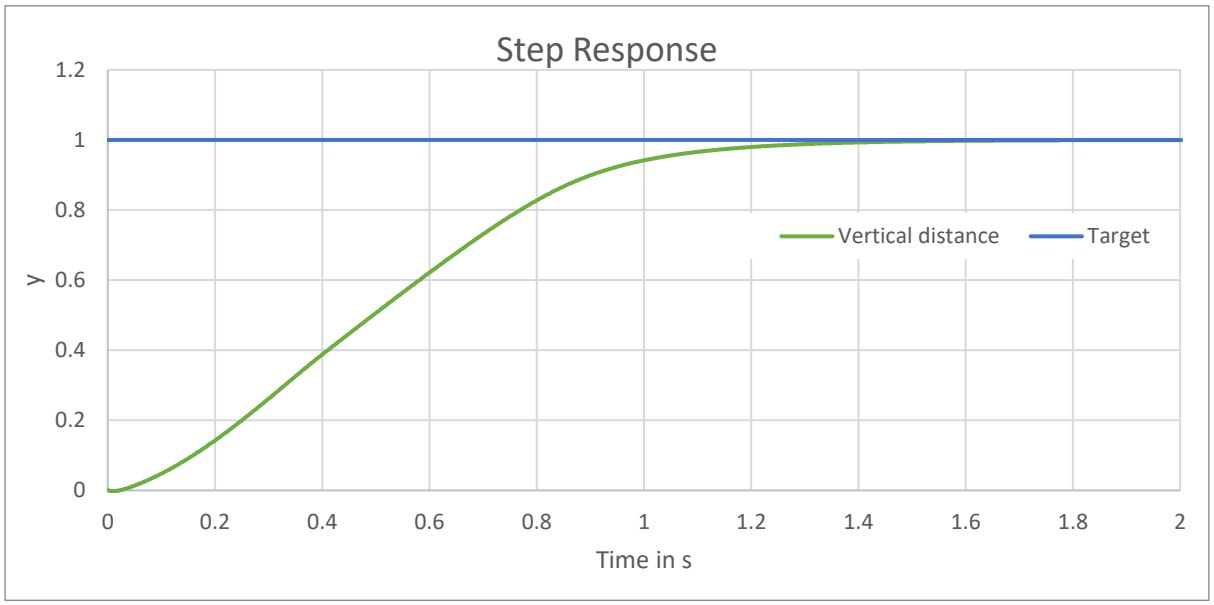


Figure 20 Step Response

Figure 20 shows the step response of the path tracking controller. The controller exhibits a rise time of 0.7351 seconds and a transient time of 1.1576 seconds. The settling time is 1.1580 seconds, with a settling minimum of 0.8955 and a settling maximum of 1.0063. The controller exhibits an overshoot of 1.1563 and an undershoot of 0.2716. The peak value of the response is 1.0063, and it occurs at a time of 2.1050 seconds. These results demonstrate the effectiveness of the path tracking controller in following the desired trajectory. Response

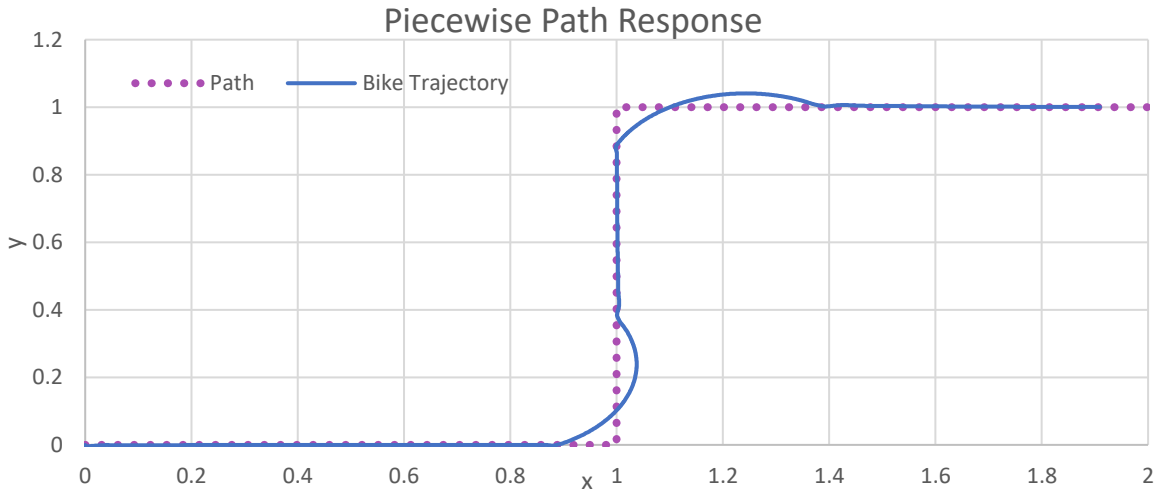


Figure 21 Piecewise Path Response

Pure Pursuit algorithms sometimes struggle with sudden turns, to ensure our controller was able to cope with these kinds of paths, the path show in Figure 21 was implemented. The results indicated that the controller was able to successfully find an acceptable path and achieve the desired steady-state behaviour. The settling distance was found to be within acceptable limits, and the overshoot and steady-state error were both minimized within the limits of the turning radius of the bike. These

results suggest that the developed controller is effective in achieving stable and reliable control of the self-balancing bicycle even when faced with particularly difficult paths. The path tracking results were highly dependent on steering angle limits. The results above were obtained with $\delta \in [-40\ 40]$ where δ is in degrees.

Despite the pure pursuit being ill suited for short turns, the results above are encouraging. Below is the system response to a more realistic path, without sharp turns.

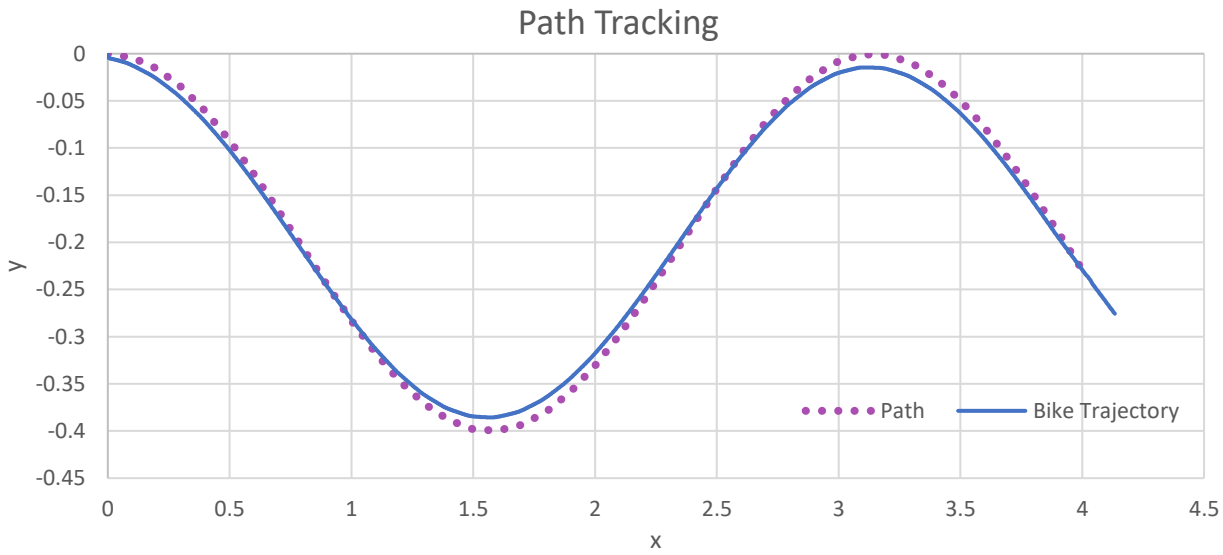


Figure 22 Path Tracking with Ground Truth States

Figure 22 displays "corner cutting" behaviour typica of pure pursuit algorithms. This is due to the lookahead distance allowing the vehicle to remain on the inside of sharper turns. Reducing the lookahead distance would reduce divergence from the path in areas of high curvature, but going too low induces instability in the system. We are satisfied with the current performance obtained with the look ahead distance of 0.15, as lower values ended up creating resonance in the system leading to the controller oscillating about the path centerline.

b. Balancing controller

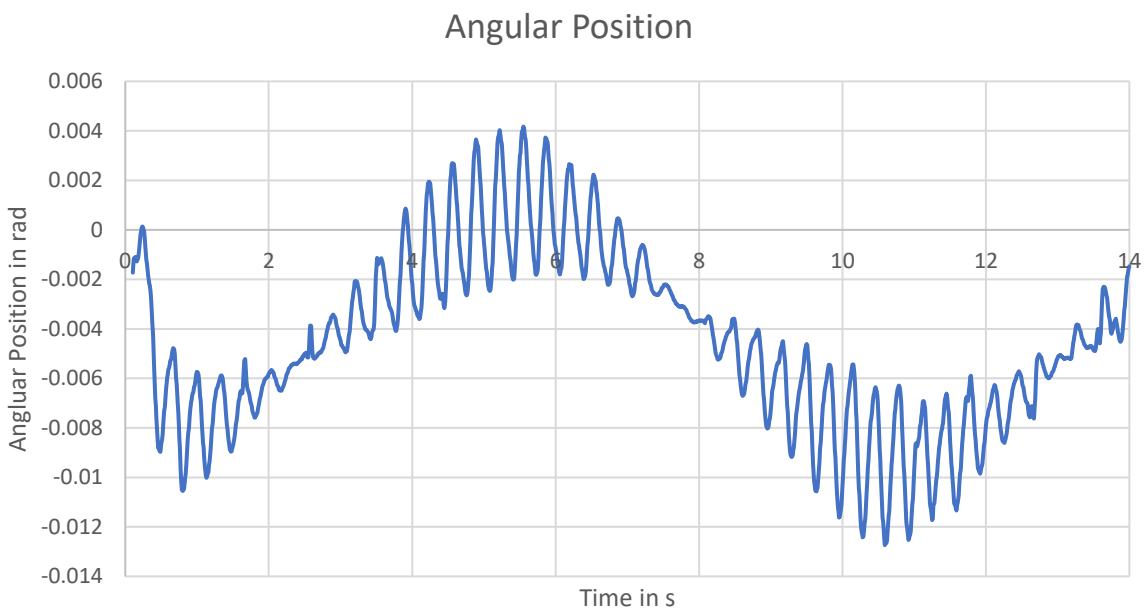


Figure 23 Graph showing the lean angle over time

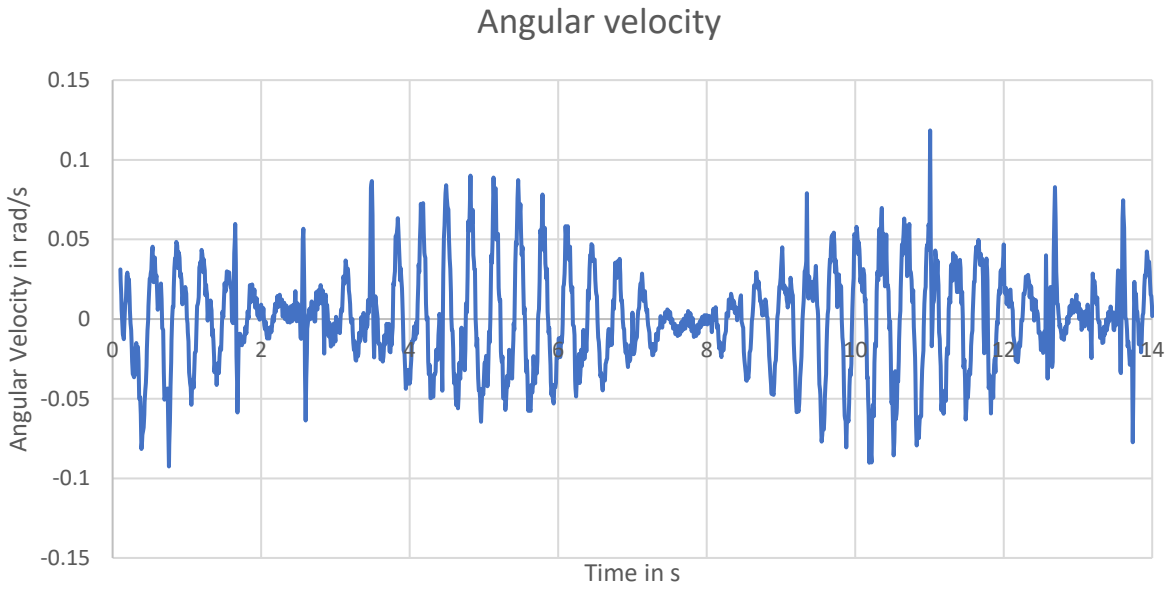


Figure 24 Graph showing the lean angular velocity over time

Figures Figure 23 and Figure 24**Error! Reference source not found.** displays the values of θ_x and ω_x over time, when the bicycle is following the trajectory depicted in Figure 22. The graphs indicates that the controller is capable of maintaining the bicycle in an upright position, even when the bicycle is in motion and executing turns. The maximum error in θ_x is found to be $1.3 * 10^{-2}\ rad$, which is indicative of the controller's efficacy in maintaining the angular position at the

desired value. However, the controller introduces vibrations, having a frequency of approximately 0.3 Hz, which is not an optimal outcome. Further fine-tuning of the controller may lead to reducing these vibrations, although it is plausible that a different controller architecture may be required to entirely mitigate them. Additionally, the graph portrays a lower frequency signal that appears to correspond with the bicycle's path. This could potentially be attributed to a coupling between the θ_x and ω_z, although further investigation is warranted to substantiate this hypothesis.

II. Error Measurement Methods

Four metrics were used to assess the overall performance of each algorithm. These are the differences between the ground truth and estimated position of the vehicle, and the Euclidian distance between the ground truth position and the estimated position.

$$e_{p_x} = |G_{p_x} - E_{p_x}|$$

$$e_{p_y} = |G_{p_y} - E_{p_y}|$$

$$e_{\theta_z} = |G_{\theta_z} - E_{\theta_z}|$$

$$d = \sqrt{(G_{p_x} - E_{p_x})^2 + (G_{p_y} - E_{p_y})^2}$$

Where e represents the error, G is used to indicate ground truth states and E to indicate estimated states. d is the Euclidean distance.

III. State Estimation Performance

We have shown the implemented controller architecture is capable of accurately following a desired path, even in the event of jagged paths. In order to assess the overall capability of the closed loop controller design, we now switch the controller from relying on the ground truth state, to using the estimated states, obtained by our state estimator. Below we show the performance of the closed loop controller, when basing its decisions on our different state estimations obtained from the sensors and refined by the sensor fusion algorithms implemented. The results show the controller performance for two distinct paths.

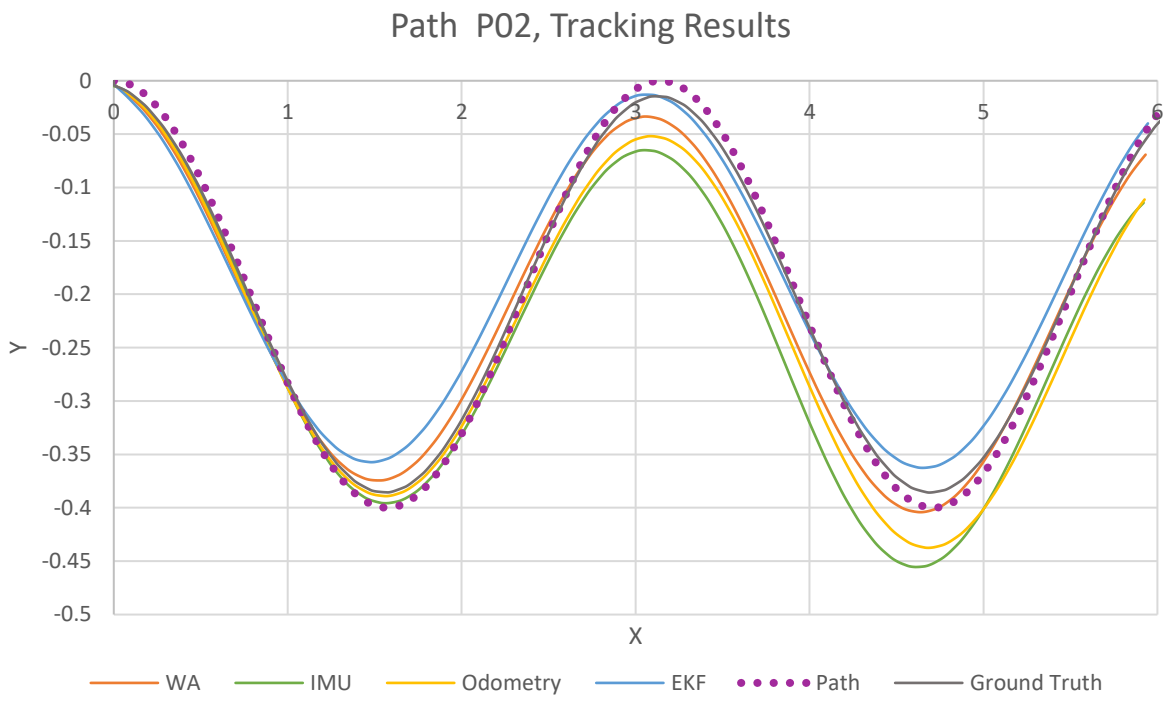


Figure 25 Path P02, Tracking Results. The x axis show time is seconds, and the y axis the error magnitude.

Figure 25 shows the trajectory of the bike, following a sinusoidal path $P = \cos(2x) * 0.2 + 0.2$, when the controller was using pose estimates based on:

- The Extended Kalman Filter (blue)
- A Weighted Average of the IMU and Odometry based estimations (Orange)
- Only the IMU (Green)
- Only the Odometry (Yellow)
- The Ground Truth values for the state (Gray)

The graph shows that all pose estimation methods lead to similar results for this path, although the EKF and the WA based pose estimations did produce somewhat better results.

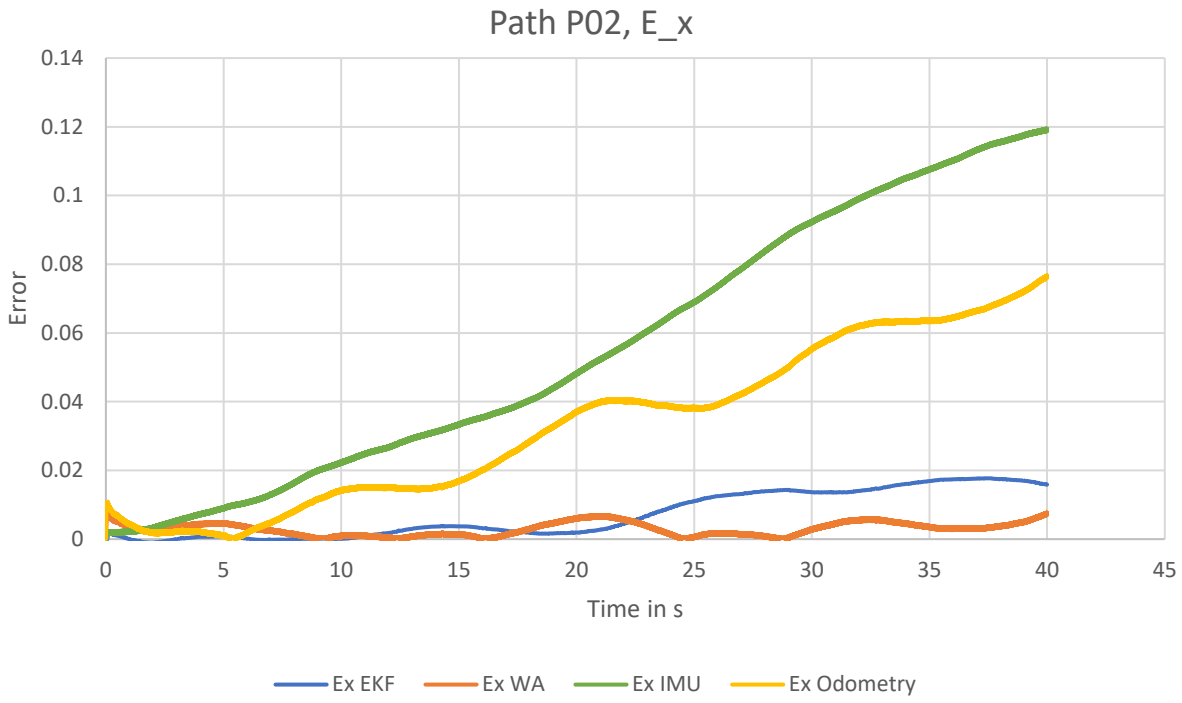


Figure 26 Path P02, Error in x position. The x axis show time is seconds, and the y axis the error magnitude.

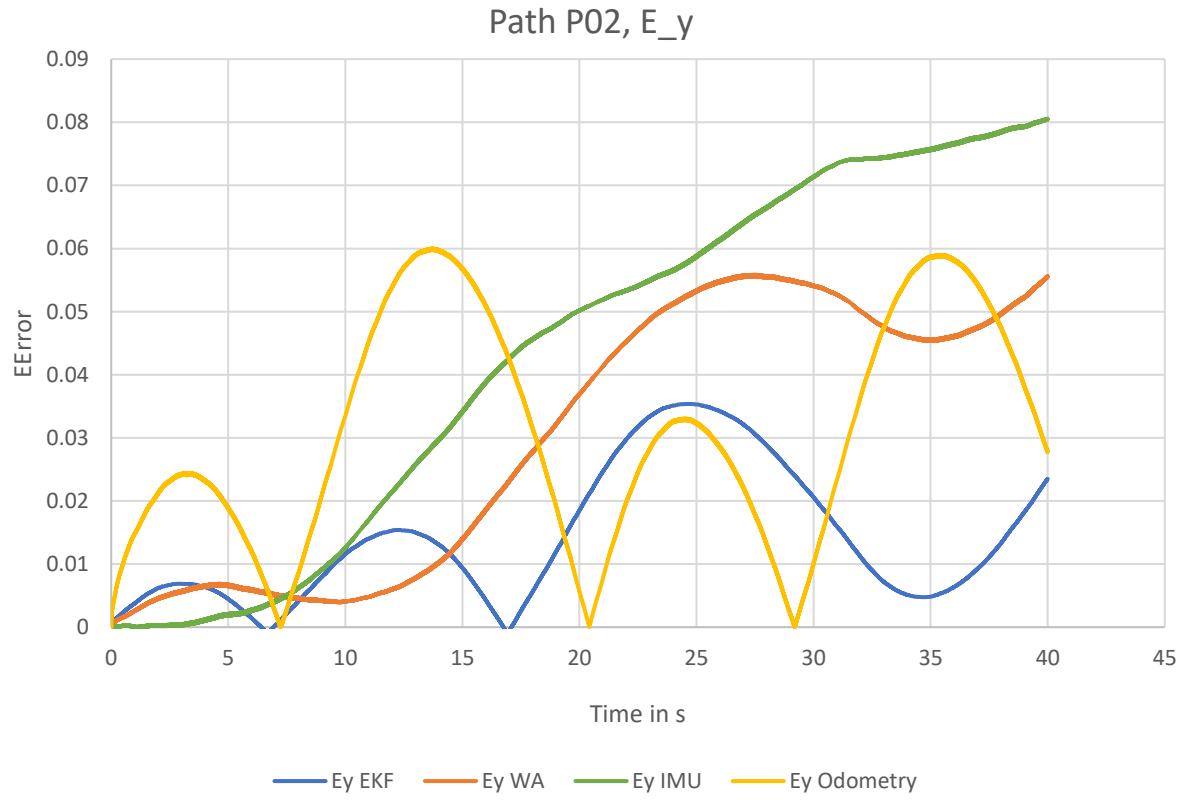


Figure 27 Path P02, Error in y position. The x axis show time is seconds, and the y axis the error magnitude.

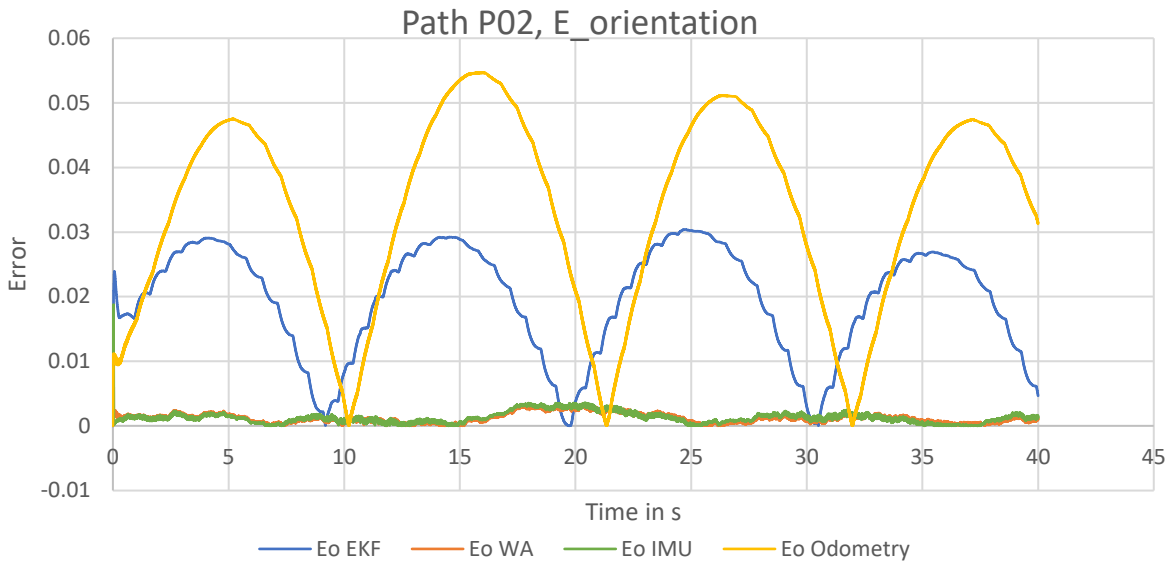


Figure 28 Path P02, Error in orientation. The x axis show time is seconds, and the y axis the error magnitude

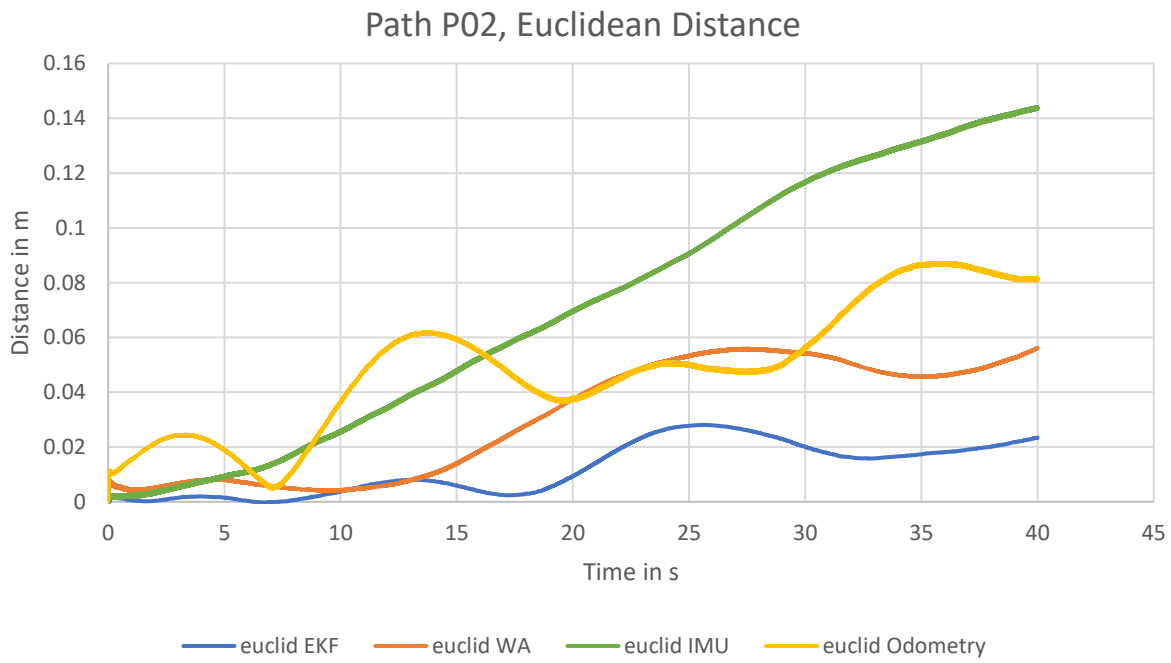


Figure 29 Path P02, Error Euclidean Distance

Figures Figure 26, Figure 27, Figure 28 and Figure 29 present errors and Euclidean distance plots for each state estimation method over time. The x-axis denotes the time step, and the y-axis represents the error. Each plot contains four lines, each line corresponding to a different pose estimation method; the EKF, a weighted average, IMU-based pose, and odometry-based pose. The error plots demonstrate the degree of deviation between the estimated and actual x and y positions of the vehicle at each time step. The Euclidean distance plots display the distance between the

predicted and actual positions of the vehicle at each time step. The legend facilitates easy identification of the different pose estimation methods.

The outcomes suggest that the weighted average and extended Kalman filter techniques exhibit analogous performance with this path. Nonetheless, upon analyzing the Euclidean distance, the extended Kalman filter technique surpasses the others by showing its ability to lead the bike along the path closes to the one obtained using ground truth states. It is worth noting that both the errors and Euclidean distance tend to escalate over time due to increasingly inaccurate state estimations, which ultimately diminishes the precision of subsequent predictions as we would expect.

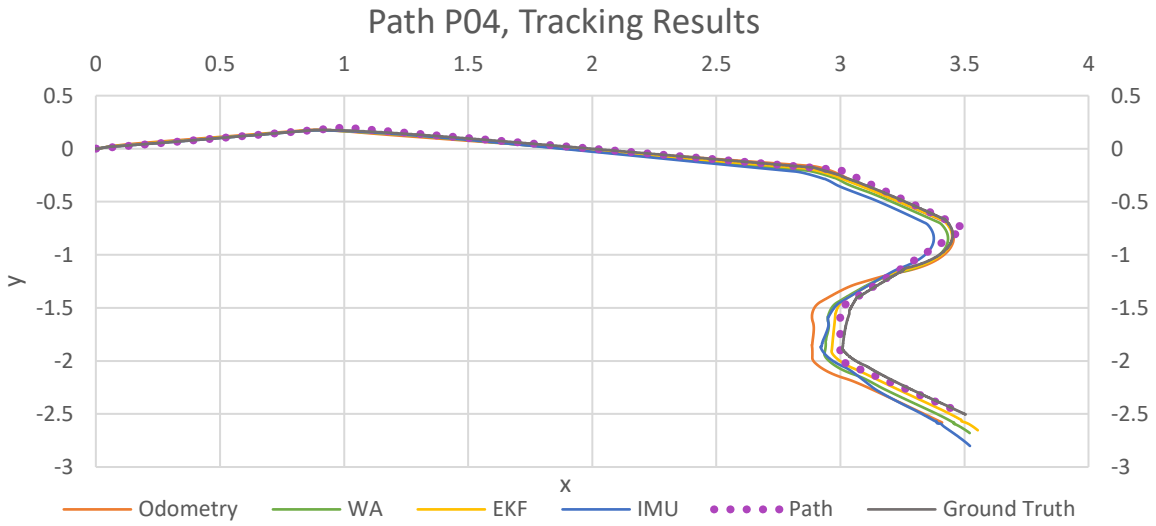


Figure 30 Path P04, Tracking Results

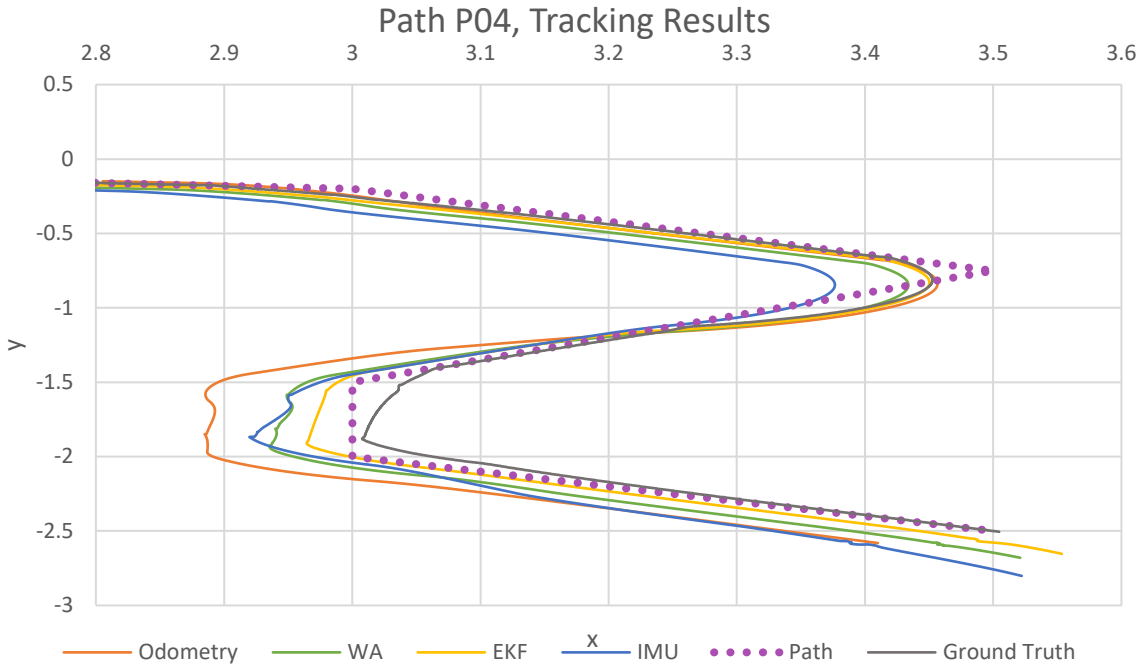


Figure 31 Path P04, Tracking Results Zoomed on End

Figures Figure 30 and Figure 31 and show the performance of the path planning controller when utilizing different pose estimates. Figure 31 gives a closer view of the final parts of the path. The EKF clearly outperforms all other algothism investigated, with the WA mathod, also performing well.

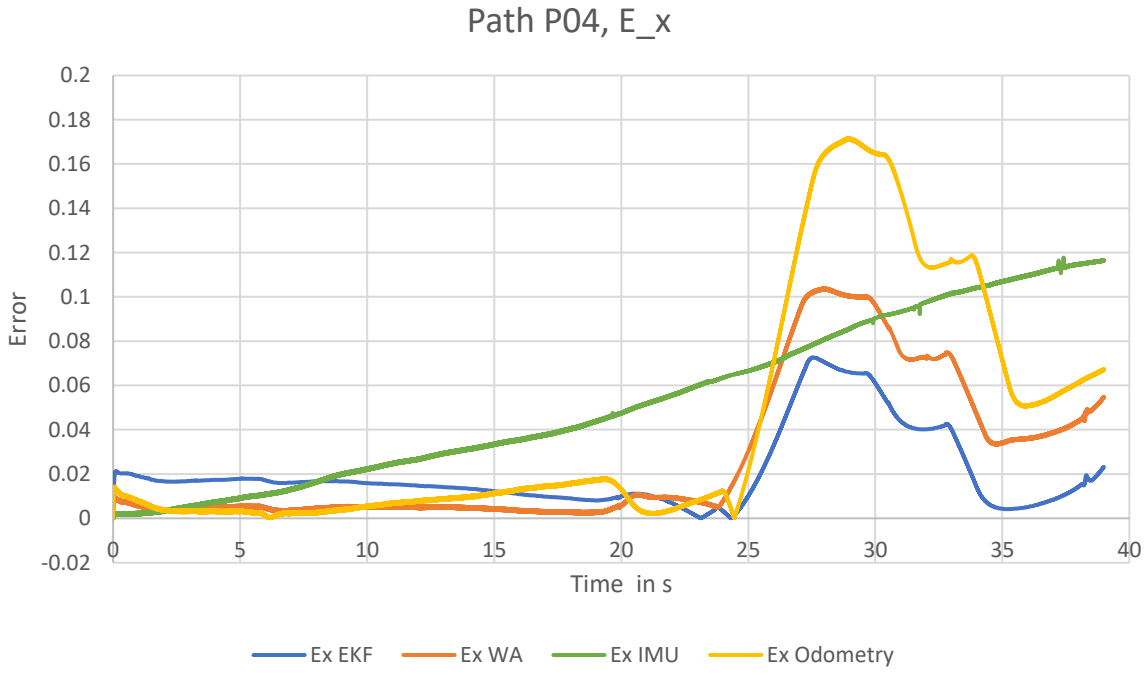


Figure 32 Path P04, Error in x Position. The x axis show time is seconds, and the y axis the error magnitude.

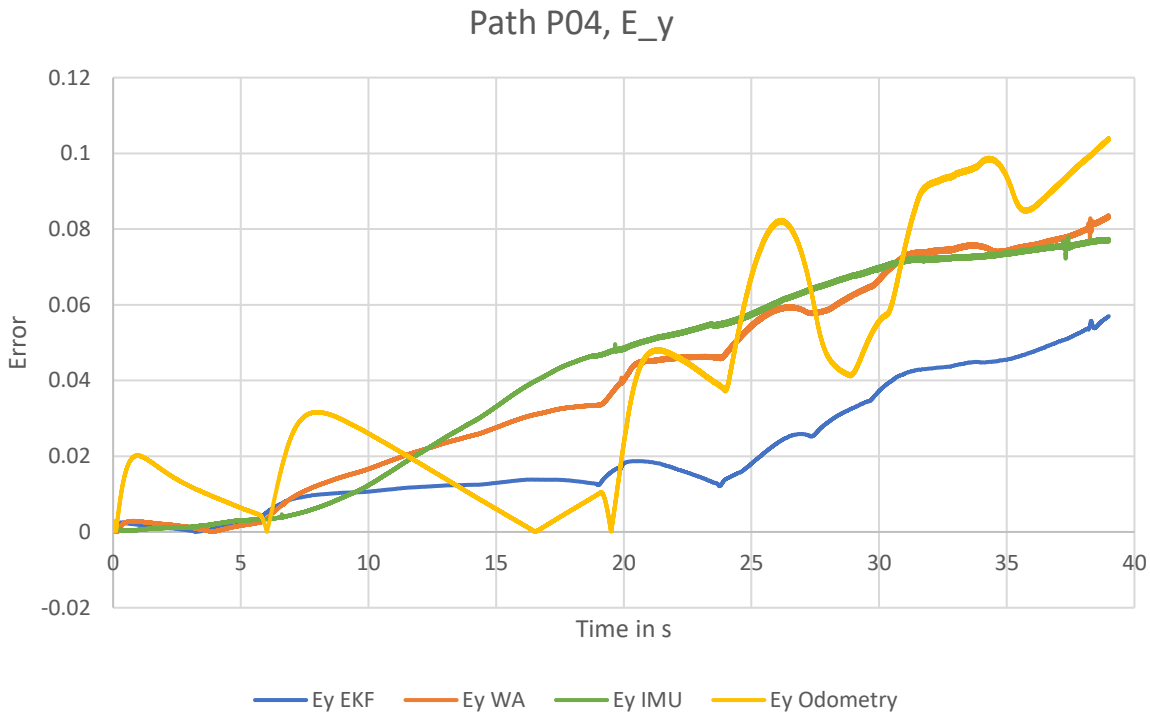


Figure 33 Path P04, Error in y Position. The x axis show time is seconds, and the y axis the error magnitude.

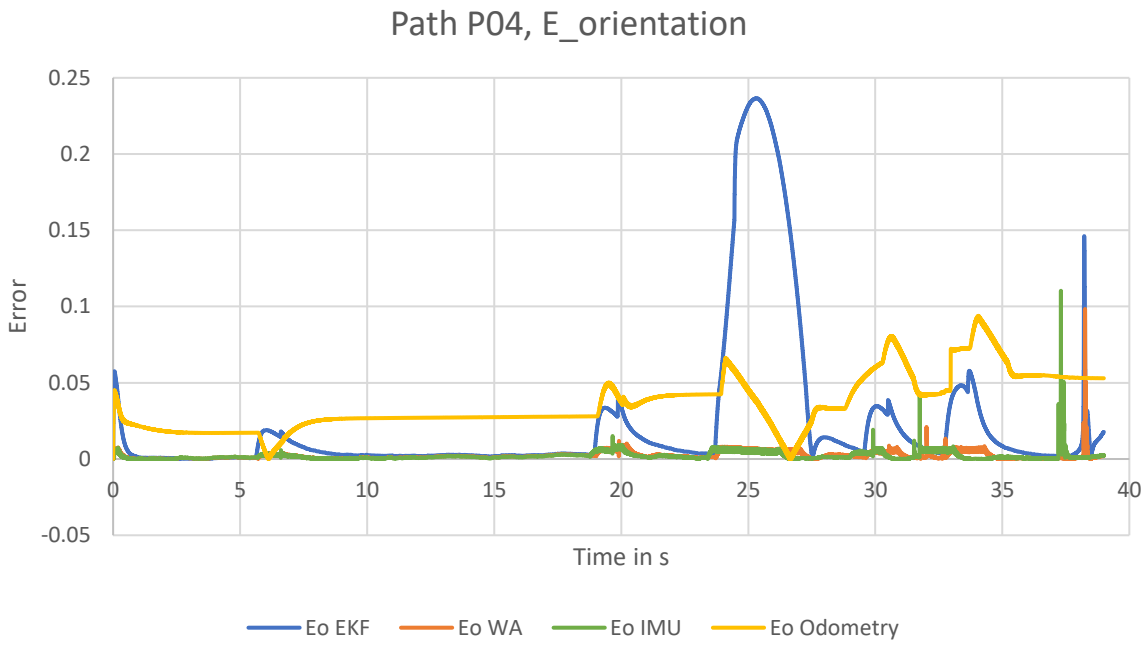


Figure 34 Path P04, Error in orientation. The x axis show time is seconds, and the y axis the error magnitude.

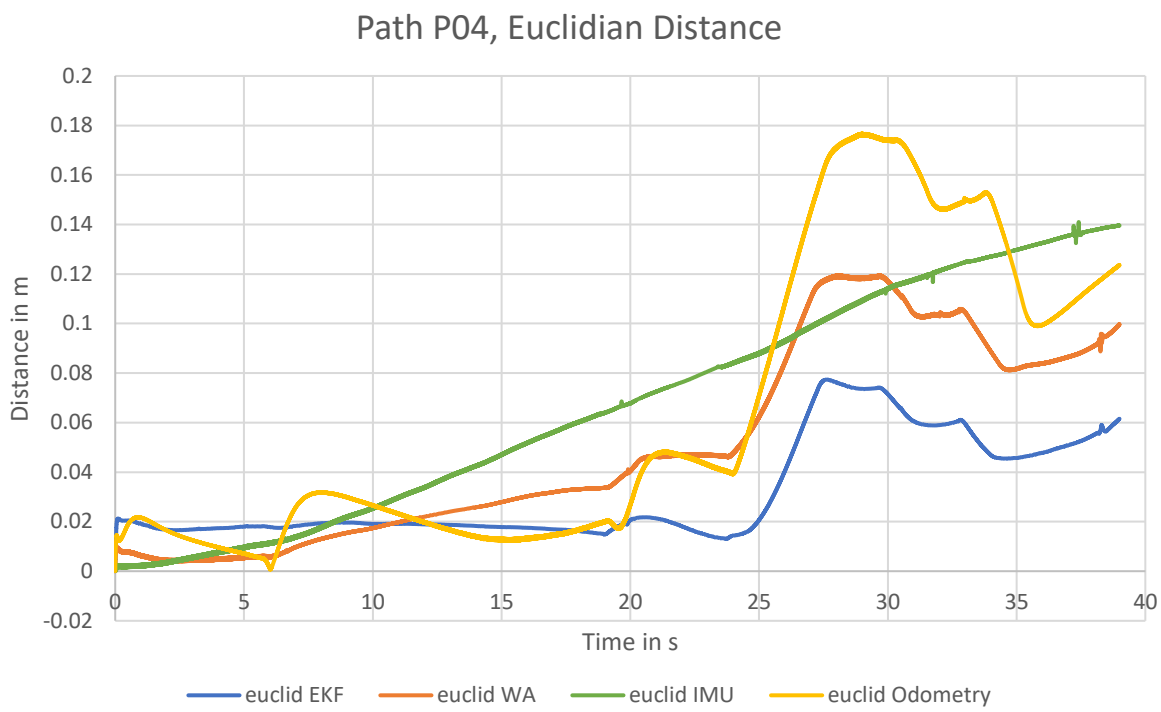


Figure 35 Path P04, Euclidian Distance. The x axis show time in seconds, and the y axis the Euclidean distance in meters.

Figures Figure 32Figure 33Figure 34Figure 35, present the errors and Euclidean distance plots resulting from utilizing the explored pose estimation techniques to inform bike controller decisions. In this context, the extended Kalman filter (EKF) technique stands out as the most effective. Notably, an intriguing pattern emerges, indicating that the magnitude of the error in the x position is largely influenced by the error attributable to odometry. This observation is further supported by the Euclidean distance graph, which demonstrates a shared pattern among signals based partly on odometry results (blue, green, and orange). Specifically, the odometry estimation exhibits satisfactory performance at the onset of the path, predominantly during the straight section. However, it loses accuracy during the curved second half of the path, likely due to wheel slip at high turn rates, as well as the inaccuracy of the odometry model at high steer angles. This accounts for the observed error pattern, characterized by low and consistent errors during the initial straight path section, followed by a sharp increase in error as the bike engages in more aggressive manoeuvres.

The IMU estimation, however, seems to be much more independent of the path characteristics, simply increasing linearly over time.

5. Conclusions and Further Research

I. Conclusions

In this project, we aimed to create a high-fidelity model of a bicycle in software and to develop a closed-loop controller architecture that would enable autonomous completion of waypoint missions. The successful achievement of these goals was dependent on accurate state estimation, which we addressed through the use of independent odometry and IMU readings, and the implementation of an extended Kalman filter to fuse these estimates. Our approach utilized a CAD model of the hardware, imported into Simulink and animated using the Simscape Multibody library. We controlled the revolute joints of the rear wheel, flywheel, and steering within the context of the simulation. Our model allowed us to study the system of the bicycle in detail and provided valuable insights into the dynamics and behaviour of the system.

We have shown our controller architecture, incorporating a pure pursuit trajectory planner and a modified PD controller for balancing was able to successfully maintain the position of the bicycle upright, and enabled it to navigate various paths.

We have also shown, that despite adding noise to the sensors used for state estimation, our state estimation output was still sufficiently accurate to allow the closed loop controller to effectively complete waypoint missions.

We have analyzed the performance of the extended Kalman filter and compared it to a WA based sensor fusion approach and to the raw pose estimates based on rotary encoder and the IMU. This highlighted the EKF's superior ability to combine the output of the sensors with a model of the bicycle to obtain a very robust state estimation.

Overall we judge that we have suitably met all initial objectives for this project.

II. Further Research

The completion of this project has opened up several avenues for future research. One promising area for further investigation involves conducting more extensive analyses of the controller and state estimation performance under various conditions. This could involve testing the system with different levels of sensor noise and drift, varying EKF parameters, and evaluating the system's behaviour under varying path lengths and wheel slip rates. By conducting such analyses, we can gain a more comprehensive understanding of the system's behaviour under diverse conditions.

Another area for potential improvement involves refining the bike model. Although the model utilized in this project was sufficient for our purposes, developing more accurate models that incorporate a more realistic wheel contact model, a full model of the motors and power systems using the Simscape Multibody blocks, and a more authentic representation of the road surface could

be beneficial. Additionally, incorporating a more accurate weight distribution model could provide a better understanding of the system's behaviour under different loading distributions.

Another possible direction for future research involves adapting the proposed controller architecture to run on the hardware version of the bike. This approach would involve using camera tracking to monitor the vehicle's ground truth pose, which could be compared to the simulation output to ensure its realism. Such an approach could offer valuable insights into the dynamics of the system and identify any discrepancies between the simulation and physical model. It would also demonstrate the viability of our approach in a real-world setting.

The results also revealed two areas which could be improved in our controller model. The first is the presence of a high frequency vibration induced by the balancing controller. This is problematic, and further investigations could be carried out on the cause of this vibration mode and possible solutions for mitigating it. The second area was revealed by the error plots for the path P04, which showed that the odometry state estimation quickly lost accuracy when the bike was manoeuvring aggressively. Further investigation could be carried out to find the source of this issue, and if possible modify the odoemtry system to obtain more reliable estimates.

Overall, this project has demonstrated the potential of using advanced control and state estimation techniques to develop a high-fidelity model of a bicycle in software. By further exploring these techniques and improving the model, we believe that this research could lead to new insights into the behaviour of these systems and have practical applications in areas such as autonomous vehicle navigation and control.

6. Responsible Engineering and Potential Impacts

Our project has focused on the development and assessment of a novel controller architecture for unstable dynamic systems. Specifically, our architecture includes autonomous path planning and sensor fusion capabilities based on the Extended Kalman Filter (EKF). While the target audience for this paper includes experts in the field of control and robotics, we also believe that this work could be used in a university classroom setting to provide students with the opportunity to explore complex dynamical systems and learn about advanced controller design.

We recognize that the technical complexity of this paper may limit its accessibility to the average individual. However, we are confident that university level students interested in robotics and control will find the material both engaging and informative. Furthermore, since our work is entirely digital, it can be used in classrooms that lack access to large robotics labs, providing more students with the opportunity to explore this fascinating field.

The model we have developed can be easily adapted to different vehicle configurations and sensor types, making it a versatile tool for educational purposes. Our hope is that the availability of such models, combined with online-based pedagogy, will improve the overall quality of education for future students. Specifically, we believe that this model can be used to improve students' familiarity and capability with MATLAB software and tools.

Our work has also demonstrated the incredible potential and usefulness of tools such as Simulink and MATLAB. In general, research in this area has the potential to increase knowledge in the field of robotics control, especially within the context of self-balancing unstable vehicles such as bicycles. This, in turn, can lead to better and more capable autonomous systems and robots being available in the future.

While we are confident in the stability and reliability of our proposed controller design, we must emphasize that it has not been sufficiently tested to be implemented in embodied systems outside of a controlled test environment. It is essential to conduct more verification and testing work before applying this controller to a physical vehicle, such as a car, to reduce risks and ensure safety.

References

Albertos, Manuel Olivares and Pedro. 2018. "Coordinated control strategy for a rotating flywheel pendulum." *2018 IEEE 14th International Conference on Control and Automation (ICCA)* 210-216.

Antsaklis, P.J., K.M. Passino, and S.J. Wang. 1991. ""An introduction to autonomous constrol systems"." *IEEE Control Systems Magazine* vol. 11 no. 4 pp. 5-13.

Arduino. 2023. *PROJECT SELF-BALANCING MOTORCYCLE.* January. Accessed April 2, 2023. https://engineeringkit.arduino.cc/aekr2/module/engineering/project/06-self-balancing-motorcycle.

Bonnabel, Martin Brossard and Silv{\'e}re. 2019. "Learning Wheel Odometry and IMU Errors for Localization." *{2019 International Conference on Robotics and Automation (ICRA)* 291-297.

Bosch. 2021. "BNO055 Datasheet." *Smart sensor: BNO055.* October. https://www.bosch-sensortec.com/products/smart-sensors/bno055/.

CHONG, Kok Seng, and Lindsay KLEEMAN. 1997. "Accurate Odometry and Error Modelling for a Mobile Robot." *Proceedings of the 1997 IEEE International Conference on Robotics and Automation.* Albuquerque, New Mexico.

Choset, Howie, Lynch, Kevin M., Hutchinson, Seth, Kantor, George A., Burgard, Wolfram, Kavraki, Lydia E., and Thrun, Sebastian. 2005. *Principles of Robot Motion : Theory, Algorithms and Implementations.* Cambridge: MIT Press.

Coulter, R. Craig. 1992. "Implementation of the Pure Pursuit Path Tracking Algorithm." *Coulter1992ImplementationOT.*

GRÖNLUND, A., & TOLIS, C. 2018. *Riderless self-balancing bicycle : Derivation and implementation of a time variant linearized state space model for balancing a bicycle in motion by turning the front wheel.* Dissertation , Stokolm : KTH ROYAL INSTITUTE OF TECHNOLOGY SCHOOL OF INDUSTRIAL ENGINEERING AND MANAGEMENT.

Kalman, Rudolph Emil. 1960. "A New Approach to Linear Filtering and Prediction Problems." *Transactions of the ASME--Journal of Basic Engineering* 35-45.

Liu, Jia and Yang, Zhiheng and Huang, Zhejun and Li, Wenfei and Dang, Shaobo and Li, Huiyun. 2021. "Simulation Performance Evaluation of Pure Pursuit, Stanley, LQR, MPC Controller for Autonomous Vehicles." *2021 IEEE International Conference on Real-time Computing and Robotics (RCAR)* 1444-1449.

MATLAB. 2017. *Understanding Kalman Filters.* April. Accessed 2023. https://www.youtube.com/playlist?list=PLn8PRpmsu08pzi6EMiYnR-076Mh-q3tWr.

Moon, Hansol & Zhang, Wenlong & Frank, Daniel & Delp, Deana & Sugar, Thomas. 2020. *Balancing Control and Model Validation of Self-Stabilizing Motorcycle.* PhD Thesis, ResearchGate.

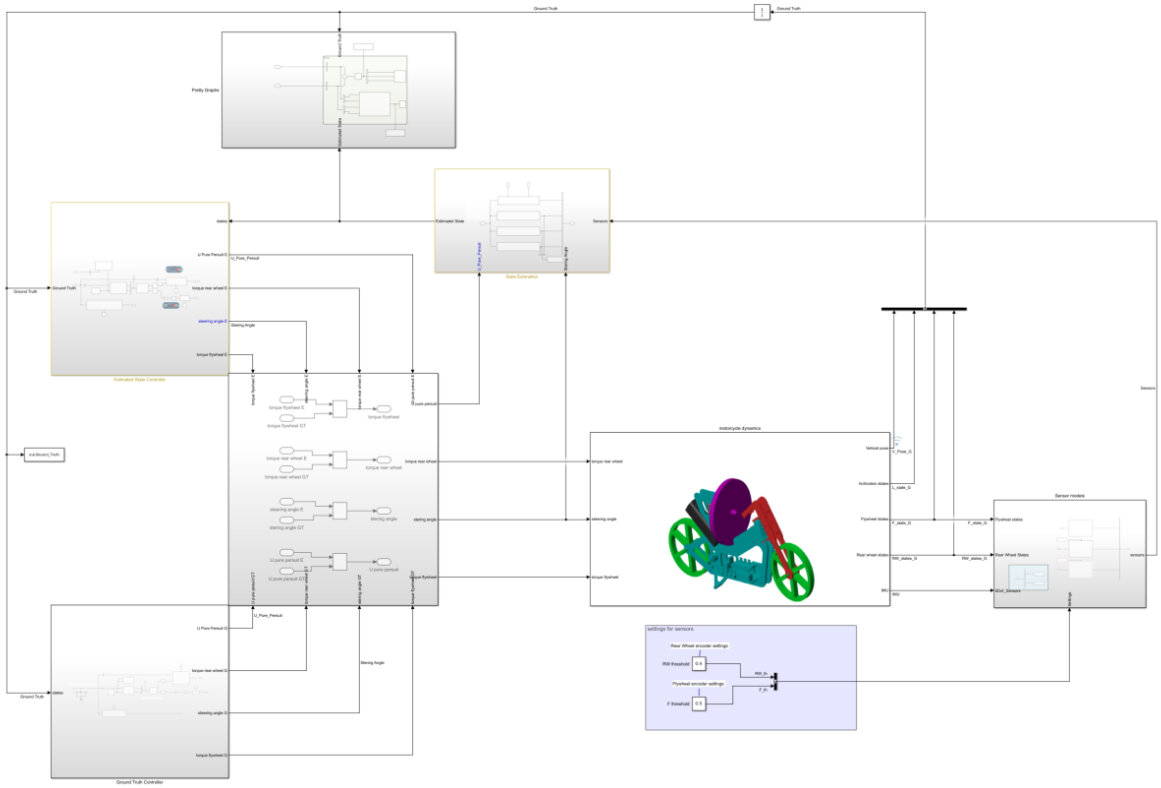
Olivares, Manuel and Albertos, P. 2014. "Linear control of the flywheel inverted pendulum." *ISA Transactions* 1396-1403.

Thanh, Bui and Parnichkun, M. 2008. "alancing Control of Bicyrobo by Particle Swarm Optimization-Based Structure-Specified Mixed H2/H∞ Control." *International Journal of Advanced Robotic Systems.*

Zhu, Jie and Tang, Yujie and Shao, Xin and Xie, Yangmin. 2021. "Multisensor Fusion Using Fuzzy Inference System for a Visual-IMU-Wheel Odometry." *IEEE Transactions on Instrumentation and Measurement.*

7. Appendix

1. Full Model



The full model and associate code for the project can be found here:

https://github.com/matteo-liguori/Individual-Engineering-Project-Comparative-Analysis-of-Sensor-Fusion-Techniques-for-State-Estimatio