ABSTRACT

Title of Thesis: OUTDOOR LOCALIZATION AND PATH PLANNING FOR REPOSITIONING A SELF-DRIVING ELECTRIC SCOOTER

Srijal Shekhar Poojari
Master of Science, 2023

Thesis Directed by: Professor Derek Paley
Department of Aerospace Engineering
Institute for Systems Research

The long-term goal of this research is to develop self-driving e-scooter technology to increase sustainability, accessibility, and equity in urban mobility. Toward this goal, in this work we design and implement a self-driving e-scooter with the ability to safely travel along a pre-planned route using automated, onboard control without a rider. We also construct an autonomous driving framework by synthesizing open-source robotics software libraries with custom-designed modules specific to an e-scooter, including path planning and state estimation. The hardware and software development steps along with design choices and pitfalls are documented. Results of real-world autonomous navigation of our retrofitted e-scooter, along with the effectiveness of our localization methods are presented.

OUTDOOR LOCALIZATION AND PATH PLANNING FOR AUTONOMOUSLY REPOSITIONING A SELF-DRIVING ELECTRIC SCOOTER

by

Srijal Shekhar Poojari

Thesis submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Master of Science
2023

Advisory Committee:
Professor Derek Paley, Chair/Advisor
Assistant Professor Michael Otte
Associate Professor Pratap Tokekar

Acknowledgments

I would like to take this opportunity to express my sincere gratitude to Prof. Derek Paley, my advisor and mentor. I thank him to trust me while taking me onboard, allowing me the independence to choose my work, and patiently waiting through unproductive weeks. I deeply respect and admire the way he effortlessly navigates his busy schedule, always able to squeeze in a meeting when needed. Perhaps his most remarkable ability, unlike any professor I've known, is to manage his email inbox so well that I've never had to wait until the next day for a response. Even my first email, before he had ever known me, I had a response in 4 minutes! This might seem trivial for some, but I greatly value the time and respect given to my queries. Thank you!

My gratitude also goes to my fellow team members at ReZoom, past and present. Starting with Raj Shinde, Pruthvi Sanghavi and Naman Gupta who I only briefly met as I joined the team; their work on developing the early prototype scooter was foundational. We still use some of the hardware design choices made by them. I then had the joy of working with Siddharth Telang, sharing ideas and pushing through the initial stages of getting onboard the project. Siddharth certainly helped me spend more time on the project than I would have on my own. I also enjoyed working alongside Vivek Sood, who helped develop remote monitoring capabilities for the scooter (these are not a part of this thesis) and also helped build our relation with Flo Mobility.

Additionally, I would like to thank Andrew Giorgi and Jeremy Kuznetsov for their wonderful work in developing the scooter hardware. The work in subsection 3.2.1 and subsection 3.2.2

are almost entirely their work. Jeremy is a total wizard with mechanical design and CAD. The newer 3D printed attachments and motor mount on revision 3 of the scooter are entirely his efforts. Andrew also helped me develop the electrical subsystem of the scooters.

Continuing with the ReZoom team, I thank Sharmitha Ganesan for her work in performing image segmentation to detect traversable/non-traversable terrain for the scooter, though her work has not been included in this thesis. I would then like to thank Vibhu Agrawal for his work in using historical scooter ride datasets to generate waypoints for navigation. This work has not yet been incorporated into our ROS stack, but looks very promising! I also thank Jaxon Lee and Matthew Fedora for their work enabling the scooter to be ridden by a user together with the electrical modifications to enable autonomy that we have done. Their work is set to be implemented in the upcoming scooter prototype, and not included in this thesis.

Next, I would like to thank Adarsh Sathyamoorthy and Kasun W. from the GAMMA Lab at UMD for their support. The discussions I've had with them about their research on the Husky has certainly helped me develop improved capabilities for the scooter.

A huge thank you to Ivan Penskiy for helping us procure (a lot of) components and parts for the scooter. Ivan gracefully manages multiple labs at the Maryland Robotics Center, including the Robotics and Autonomy Lab where ReZoom resides. I thank him for his help and support, and apologize for the mess that we create at our desks.

I thank the Maryland Transportation Institute and the Maryland Robotics Center for generously supporting me through this research. I would also like to thank the Graduate School for the Outstanding Graduate Assistant Award which has also helped support me.

And finally, but most importantly, I thank my parents and my sister for nourishing, guiding and supporting me throughout my life. I would not be here without them.

Table of Contents

Acknowledgements ii

Table of Contents iv

List of Figures vi

List of Abbreviations viii

Chapter 1: Introduction 1
- 1.1 Motivation . . . 1
- 1.2 Previous and Ongoing Work . . . 3
- 1.3 Technical Approach . . . 4
- 1.4 Contributions . . . 6
- 1.5 Outline . . . 7

Chapter 2: Background 8
- 2.1 Robot Operating System (ROS) . . . 8
- 2.2 Coordinate Frames . . . 9
  - 2.2.1 Types of North: True North . . . 10
  - 2.2.2 Types of North: Magnetic North . . . 10
  - 2.2.3 Types of North: Grid North . . . 10
- 2.3 Scooter Kinematic Model . . . 13
  - 2.3.1 Holonomic and Non-holonomic Robots . . . 13
  - 2.3.2 The Bicycle Kinematic Model . . . 14
  - 2.3.3 Expected Movements to Motor Commands . . . 15

Chapter 3: Experimental Testbed 16
- 3.1 Our Scooter Fleet . . . 16
  - 3.1.1 Ben Parker . . . 17
  - 3.1.2 Richard Parker . . . 18
  - 3.1.3 Peter Parker . . . 19
- 3.2 Mechanical Design . . . 21
  - 3.2.1 Steering Mechanism . . . 21
  - 3.2.2 Stabilizing Mechanism . . . 22
  - 3.2.3 Component Mounting and Placement . . . 24
- 3.3 Electrical Design . . . 27
  - 3.3.1 Connection Diagram . . . 27

3.3.2 Ground Loops . . . 27
3.3.3 Decoupling Capacitors . . . 28
3.3.4 Component Current Draw . . . 30
3.3.5 Analog/Digital Pathways . . . 31

Chapter 4: The ReZoom Autonomy Stack 32
4.1 Localization . . . 32
4.1.1 The Localization Problem . . . 32
4.1.2 The Extended Kalman Filter . . . 33
4.1.3 The Robot Localization package . . . 34
4.1.4 Localization Data Flow Overview . . . 40
4.1.5 Problems with Visual Odometry . . . 40
4.1.6 Initialization . . . 43
4.2 Planning and Navigation . . . 48
4.2.1 Move Base . . . 48
4.2.2 Preparing Waypoints . . . 51
4.2.3 Publishing Waypoints . . . 52
4.2.4 An Overview of the Autonomous Navigation Process . . . 53

Chapter 5: Autonomy Experimental Results 55
5.1 Evaluating Localization Performance . . . 55
5.1.1 Methodology to Evaluate Localization . . . 55
5.1.2 Localization Performance Results . . . 56
5.2 Local Planning Results . . . 61
5.3 Autonomous Waypoint Navigation . . . 62
5.3.1 Methodology for Autonomous Waypoint Navigation . . . 62
5.3.2 Results of Autonomous Waypoint Navigation . . . 63

Chapter 6: Conclusion 68
6.1 Summary of Contributions . . . 68
6.2 Future Work . . . 69

Bibliography 70

List of Figures

1.1 Scooters cluttering a sidewalk . . . 2
1.2 Self-stabilizing e-scooter by researchers at Stuttgart . . . 3
1.3 Self-driving scooter by Go X . . . 4
1.4 Teleoperated scooter by JUMPWatts . . . 5
1.5 ReZoom's autonomous scooter testbed . . . 6

2.1 UTM Grid in the United States . . . 11
2.2 Types of North Compared . . . 12
2.3 Typical UTM to map frame transformation . . . 12
2.4 The Bicycle Kinematic Model . . . 14

3.1 All three ReZoom scooters . . . 17
3.2 Ben Parker, our first scooter testbed . . . 18
3.3 Richard Parker, our second scooter testbed . . . 19
3.4 Peter Parker, our second scooter testbed . . . 20
3.5 The old motor steering mechanism on Richard Parker . . . 21
3.6 The new motor steering mechanism on Richard Parker . . . 22
3.7 The rear wheels on Richard lift off the ground over certain uneven surfaces . . . 23
3.8 Peter's new support wheels have suspension to allow for tilt and improved ride quality . . . 23
3.9 A diagram showing an overview of the scooter's component placement . . . 25
3.10 A picture comparing the camera placement on Ben and Richard. . . . 26
3.11 A picture showing the placement of GPS antenna . . . 27
3.12 Scooter Electrical Connection Diagram . . . 28
3.13 The problem with Ground Loop . . . 29
3.14 Scooter's underbelly showing the ground to chassis connections . . . 29
3.15 Decoupling capacitors added to the motor encoder pins . . . 30

4.1 All coordinate frames for the scooter shown in a TF tree - Part A . . . 38
4.2 All coordinate frames for the scooter shown in a TF tree - Part B . . . 39
4.3 Overview of the scooter's localization data flow . . . 41
4.4 Visual tracking fails in low feature situations . . . 42
4.5 ORB SLAM2 incorrect tracking with the robot's shadow in view . . . 43
4.6 A picture showing our initialization test setup . . . 45
4.7 Flow diagram for the initialization node using April Tags . . . 47
4.8 An overview of move_base . . . 49
4.9 Screenshots showing the manual generation of waypoints from Google Maps . . . 52

4.10 A flow diagram of the outdoor waypoint navigation node 53
4.11 Overview of the scooter's autonomy data flow 54

5.1 Selecting Ground Truth Points 56
5.2 Starting the scooter facing an AprilTag 57
5.3 Pushing the scooter manually along a known path to evaluate EKF performance [CSI to Iribe] 58
5.4 Pushing the scooter manually along a known path to evaluate EKF performance [IDF to CSI] 59
5.5 Pushing the scooter manually along a known path to evaluate EKF performance [ERB to Regents Dr.] 59
5.6 Pushing the scooter manually along a known path to evaluate EKF performance [Looneys Loop] 60
5.7 A zoomed in view of the end of the long EKF test 60
5.8 Plot showing error in EKF vs distance travelled 61
5.9 And example of TEB local planner's planned path 62
5.10 Plot showing results of the scooter's autonomous mission [CSI to Iribe 1] 64
5.11 Plot showing results of the scooter's autonomous mission [CSI to Iribe 2] 65
5.12 Plot showing results of the scooter's autonomous mission [ERB to Regents Dr.] . 66
5.13 Plot showing distance to waypoint vs distance travelled for three autonomous missions 66
5.14 The scooter drives into grass and gets stuck 67

List of Abbreviations

ReZoom	Research in Electric Scooter Urban Mobility (RESUME)
ROS	Robot Operating System
IMU	Inertial Measurement Unit
GPS	Global Positioning System
REP	ROS Enhancement Proposals
TN	True North
MN	Magnetic North
GN	Grid North
UTM	Universal Transverse Mercator
MGRS	Military Grid Reference System
LL	Latitude and Longitude coordinate pair
RPM	Revolutions Per Minute
ICR	Instantaneous Center of Rotation
TEB	Time Elastic Band (local planner)
OEM	Original Equipment Manufacturer
USB	Universal Serial Bus
CPR	Counts Per Revolution
ADC	Analog to Digital Converter
UART	Universal Asynchronous Receiver Transmitter
SPI	Serial Peripheral Interface
I2C	Inter-IC (Integrated Circuit) Communication
CAN	Controller Area Network
EKF	Extended Kalman Filter
UKF	Unscented Kalman Filter
VO	Visual Odometry
VIO	Visual Inertial Odometry
Hz	Hertz
ENU	East North Up
NED	North East Down
URDF	Unified Robotics Description Format
TF	Transform
SLAM	Simultaneous Localization and Mapping
ORB SLAM	Oriented FAST and Rotated BRIEF (ORB) SLAM, a type of SLAM algorithm
RTABMap	Real Time Appearance Based Mapping

Chapter 1: Introduction

1.1 Motivation

Shared micromobility systems like e-scooters are envisioned to be a vital part of a new urban mobility model that promises improvements in sustainability, accessibility, and equity over current modalities like cars and buses for short travel distances. The United States Environmental Protection Agency estimates that eliminating half of US car trips less than one mile would save $575M/year in fuel costs and $900M/year overall, including savings on maintenance and tire replacement, as well reduce about 2 million metric tons of CO2 emissions per year [1]. These short trips are significant.

Despite their promise of sustainable, accessible, and equitable transportation, shared electric scooters clutter city sidewalks and landscapes. This public nuisance causes city planners to limit the number of shared operators and the size of their scooter fleets, which in turn reduces revenue and diminishes the potential benefits of dockless scooters. Very recently, the city of Paris banned all rental electric scooters from the city, with the mayor describing them as a "nuisance" [2]. Further, to rent a share scooter, a user must find one nearby and walk to it; often, it has insufficient charge. Rentals must also end in suitable parking spots that may not be near the user's desired destination.

A self-driving scooter can be summoned like a car ride-share vehicle and park itself. Au-

Figure 1.1: Improperly parked Veo rental scooters clutter a sidewalk.

tomated e-scooters can also be deployed in so-called transit deserts with poor or limited public transportation options, also known as Economic Opportunity Zones, where residents may lack access to a personal vehicle or convenient transit, making even short essential trips a challenge. Electric scooters therefore have the potential to reduce racial, generational, and geographical transportation disparities.

There are large daily operating expenses with the rebalancing of conventional scooters using cars and vans. Not only is rebalancing very expensive, but it ends up undoing a lot of the environmental benefit of shared electric scooters in terms of both traffic and pollution. Instead, with an autonomous electric scooter, the scooter can reposition itself, albeit over short distances. Additionally, a rider using a self-driving e-scooter fleet is less likely to find a scooter without sufficient battery charge. Self-driving scooters can also reposition for batch pickups, saving on emissions and logistics costs.

From a technology standpoint, most self-driving vehicles are either large and operate over larger distances (think self-driving cars and trucks) or are small and operate indoors. Self-driving electric scooters present a novel challenge because of their small form-factor, small battery ca-

Figure 1.2: Researchers at the University of Stuttgart have presented their scooter self-stabilizing using a reaction wheel[4].

pacity and low unit costs. An autonomous e-scooter is required to travel along roads, follow traffic signs, and avoid pedestrians, just as an autonomous car would. There is a gap in existing knowledge about how to meet these needs when constrained by size, weight, and power.

1.2 Previous and Ongoing Work

Previous research toward an autonomous electric scooter include the work done by Wenzelburger et al. (2020)[3] and Soloperto et al. (2021)[4] at the University of Stuttgart. Their research presents a method to self-stabilize the scooter using a reaction wheel[1] as shown in Figure 1.2. They have also presented methods for planning and navigation of their scooter in simulation. But the scooter in their presented video is teleoperated, and autonomous driving is still under development.

Go X, a company in San Francisco, seem to have deployed their self-driving electric scooter

[1]The Stuttgart electric scooter can be seen on YouTube: https://www.youtube.com/watch?v=nD8vZYZFTTU

Figure 1.3: Go X has deployed its self-driving electric scooter fleet in Georgia in 2020.

fleet in 2020-21 as shown on YouTube[2] and several news outlets[5–7]. Post 2021, there are no recent news or updates on their website or elsewhere on the internet. As seen in Figure 1.3, the Go X Apollo scooters appear to have used retractable support wheels, a friction drive mechanism at the front wheel and a camera mounted on at the top of the handlebars. The level of autonomy achieved by these scooters is not known.

Another company based in Los Angeles, JUMPWatts, develops remote management technology for light electric vehicles, including electric scooters. An example of their scooter is shown in Figure 1.4. Their focus is on teleoperation for remotely getting scooters back to a parking lot or charging station. These scooters are not autonomous.

1.3 Technical Approach

Our approach is to customize commercial off-the-shelf electric scooters with the hardware and sensors required to achieve micro-scale re-position up to several hundred meters. This project

[2]The Go X Apollo can be seen on YouTube at: https://www.youtube.com/watch?v=JZDzZzPYXls

Figure 1.4: A teleoperated scooter by JUMPWatts for remote fleet management. Also seen is the dockable red charging port in the front.

was originally funded as a seed grant from the Maryland Transportation Institute entitled "Research in Electric Scooter Urban Mobility (RESUME)", which we have renamed ReZoom. Our ongoing efforts have led to three iterations, an example of which can be seen in Figure 1.5.

The ReZoom electronic stack includes a single board computer that performs all onboard computing and runs the Robot Operating System (ROS) on Ubuntu Linux. Additionally, we have a stereo camera for depth sensing, Inertial Measurement Units (IMUs) and GPS modules for localization, a motor driver and accompanying power circuitry for navigation.

Creating a self-driving electric scooter poses novel challenges in autonomous vehicle research. By comparison, self-driving cars are equipped with sensors and computation capabilities that are too large and expensive for e-scooters. Therefore, our approach towards these challenges is to leverage advances in mobile robotics, while keeping in mind the physical, technical, and financial constraints of a shared e-scooter system including the unit cost. Our efforts focus on developing hardware and software specific to this system to help achieve the goal of building a

Figure 1.5: The version 2 prototype of our autonomous scooter testbed named Richard Parker.

sustainable, accessible, and equitable micromobility platform for shared rental operators.

We seek to develop a hardware and software stack capable of achieving autonomous micro-scale re-positioning of the scooter; planning a path through roads, trails, and sidewalks; navigating while avoiding cars and pedestrians; and being sufficiently low-cost to be deployed on a large scale. Historical rental scooter ride data will allow the scooter to take advantage of previous rides performed by humans and learn the optimal paths to navigate through the urban landscape. In addition, we will develop onboard perception and state estimation algorithms to enable the scooter to perceive its environment and navigate relative to its goal, up to hundreds of meters away.

1.4 Contributions

The primary contributions of this thesis are as follows:

1. We first lay the hardware setup and procedure to develop an autonomous scooter from a commercial off-the-shelf scooter. The scooters are modified both mechanically and electri-

cally to allow autonomous control by an onboard single board computer. Support wheels are added to stabilize the scooter without a rider.

2. Next, we develop the software and integration specific to the hardware for achieving outdoor autonomous waypoint navigation. This includes the localization module, which forms the backbone for the path planning module. Algorithms for initializing the scooter's pose and navigating the scooter through waypoints have also been developed. We individually test these modules and then bring them together for several autonomous missions.

3. In relation to the above two, the findings from this thesis provide a foundation for further development toward autonomous scooters and other, similarly constrained, ground robots.

1.5 Outline

The rest of this thesis is laid out as follows:

- Chapter 2 provides the reader a Background of some necessary prerequisite concepts.
- Chapter 3 describes the build of our Experimental Testbed; including the mechanical design, electrical design, and component placement.
- Chapter 4 describes the software aspects developed for Autonomy; including the hardware interface nodes, development of localization, planning and navigation.
- Chapter 5 provides Experimental Results for a few planned outdoor and indoor missions which demonstrate the capabilities developed.
- Chapter 6 summarizes and concludes the work done as part of this thesis.

Chapter 2: Background

This chapter provides the reader with background information that support the rest of this thesis.

2.1 Robot Operating System (ROS)

From the ROS Wiki[8]: ROS is an open-source, meta-operating system for your robot. It provides the services you would expect from an operating system, including hardware abstraction, low-level device control, implementation of commonly-used functionality, message-passing between processes, and package management. It also provides tools and libraries for obtaining, building, writing, and running code across multiple computers.

ROS acts as a middleware, providing a common set of rules and communication protocol between different pieces of robotic software. The primary goal of ROS is to support code reuse in robotics research and development. Because of this common set of rules, ROS packages are modular, and can be easily distributed. This makes prototyping quick, and allows you to reuse software from its large community.

If you have not used ROS before, or have not used it in a while, I highly recommended going through the beginner tutorials[1]. The sections ahead assume basic familiarity with ROS and

[1] ROS Beginner Tutorials: http://wiki.ros.org/ROS/Tutorials

its terminology.

2.2 Coordinate Frames

Coordinate Frames are probably the single most important thing to keep in mind as you navigate your way through the rest of the autonomy software.

If you check the ROS message template for most of the common ROS message types, for example *sensor_msgs/Imu* [2] or *geometry_msgs/TransformStamped* [3], they have *std_msgs/Header* as one of their fields. *std_msgs/Header* [4] has the field *frame_id*, which is the coordinate frame the data is associated with. Almost all messages in ROS—sensor measurements, robot goals, maps, etc.—have a coordinate frame associated to them as part of the *std_msgs/Header* field. Its a good practice to check message templates of the topics you are working with.

ROS has its own specific naming conventions for a few common coordinate frames which are in general applicable and present in most ROS based mobile robotic platforms. The ROS Enhancement Proposals (REPs) 103[9] and 105[10] specify units, naming convections and semantic meaning for common coordinate frames for ROS. It is highly recommended in giving these a read if you are unfamiliar.

Another thing to be aware of is the existence of True North, Grid North and Magnetic North directions.

[2] http://docs.ros.org/en/noetic/api/sensor_msgs/html/msg/Imu.html
[3] http://docs.ros.org/en/noetic/api/geometry_msgs/html/msg/TransformStamped.html
[4] http://docs.ros.org/en/noetic/api/std_msgs/html/msg/Header.html

2.2.1 Types of North: True North

True North (TN), also called geodetic north or geographic north and denoted by a star, is the direction along the line of longitude that points to the north pole. As a GPS sensor reports latitude and longitude coordinates, its data can be said to be in True North and True East frame of reference. Note that this is not a "Cartesian" frame, as latitude and longitude lines are like rings on a globe.

2.2.2 Types of North: Magnetic North

Now the earth's magnetic field, generated by the fluid motion inside the planet's core, is not constant. So the direction that a magnetic compass points to, and similarly a magnetometer in a robot's IMU, is not toward True North, but aligned with the earth's magnetic field. The angle between True North and Magnetic North (MN), commonly referred to as the *magnetic declination*, varies with time (as the earth's core flows around) and your location on the earth's surface. For College Park, MD, USA, this angle is about 10.8 degrees—not insignificant!

The National Oceanic and Atmospheric Administration provides a helpful calculator to find the magnetic declination at any location[5].

2.2.3 Types of North: Grid North

As most robots operate in a relatively small area on the surface of the earth, it is convenient to have a planar, cartesian coordinate system that still gives you a unique (x, y) coordinate for each point in the plane for each latitude/longitude on earth. The Universal Transverse Mercator (UTM)

[5]NOAA's Magnetic Declination Calculator: https://www.ngdc.noaa.gov/geomag/calculators/magcalc.shtml

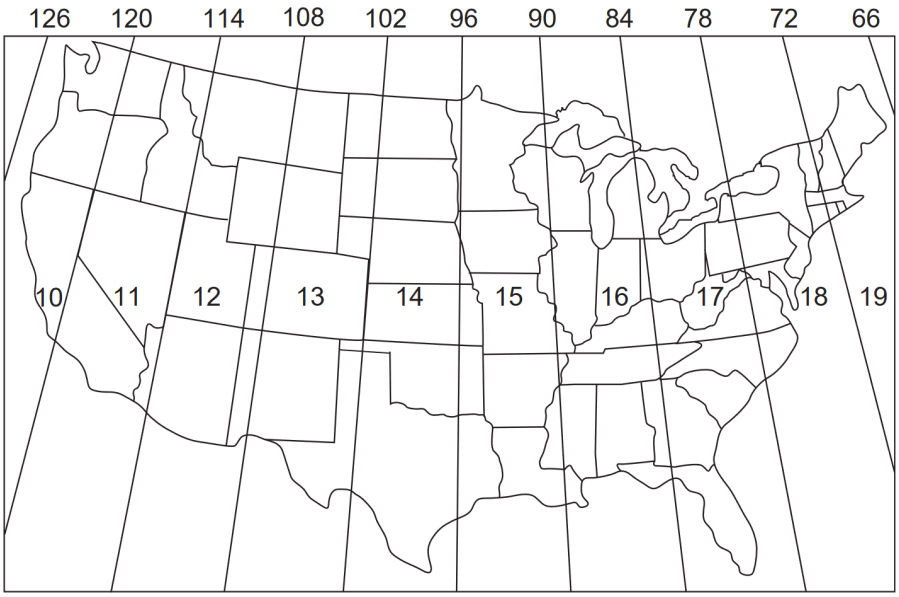

Figure 2.1: The Universal Transverse Mercator grid that covers the conterminous 48 United States comprises 10 zones. Source [11]

system is a map projection system that does exactly this. UTM divides the earth into 60 planar UTM zones, and projects each latitude/longitude to the plane. If you get a latitude,longitude reading from your GPS sensor, it is possible to convert it into a unique UTM zone and (x,y) coordinate in that UTM plane.

Figure 2.1 shows the UTM zones covering United States. At College Park, MD, USA, we operate in UTM Zone 18N, not to be confused with the Military Grid Reference System (MGRS) Zone 18S for College Park. This UTM Zone, like all others, has a globally fixed origin (at a fixed latitude and longitude).

Grid North (GN) is the direction of north for this planar UTM grid system. There is a small angle between TN and GN (I have observed about 1.2 degrees), which is just the inherent effect of projecting the earth's spherical surface on to a plane.

Figure 2.2 illustrates the angles between GN, TN and MN at College Park. Figure 2.3

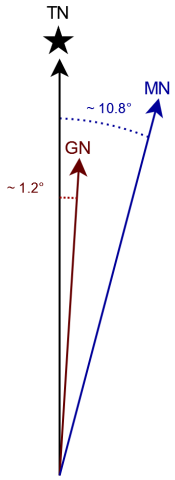

Figure 2.2: Angles between TN, GN and MN in College Park, MD, USA (not to scale).

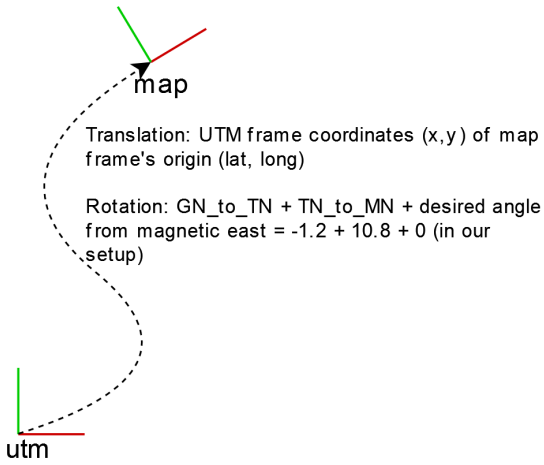

Figure 2.3: A typical `utm` to `map` frame transformation in ROS.

shows these angles affecting the transformation between the `utm` frame and `map` frame in our setup. Section 4.2 in Chapter 4 will explore coordinate frames specific to the robot and their significance in further detail.

2.3 Scooter Kinematic Model

After a robot plans its path from a start location to a goal, we want the robot to *move* and follow the path. The required movements would be a series of commands sent to the robot as the robot moves, something like: move forward at 2 meters/sec; stop; turn left 20 degrees; move forward at 1 meters/sec. Now for each of these commands, how many revolutions per minute (RPM) should my motors spin at? This question is dependent on the geometry of the robot, specified by a robot's kinematic model.

2.3.1 Holonomic and Non-holonomic Robots

A robot is non-holonomic if its controllable degrees of freedom are less than the total degrees of freedom, and holonomic otherwise. For example, a robot with mecanum wheels[12] which allow for omni-directional motion would be holonomic. A car in a 2D plane has three degrees of freedom (x, y position and its orientation), but only has two controllable degrees of freedom (acceleration and steering), and is therefore non-holonomic.

Being non-holonomic restricts the paths that a robot can take to reach a given goal configuration. For example, to parallel park a car, we have to perform a series of forward and backward motions, and cannot simply slide sideways as a holonomic robot would.

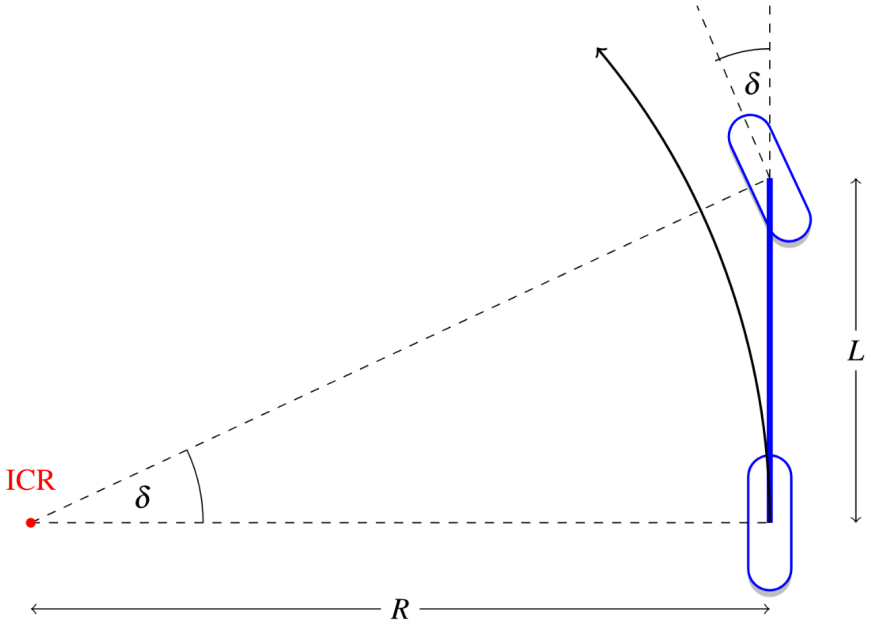

Figure 2.4: The Bicycle Kinematic Model. Source [15]

2.3.2 The Bicycle Kinematic Model

Most common robots used in academic settings like the TurtleBot have a differential drive[13]. This enables the robot to spin wheels on both sides of its body in the opposite direction and turn on-the-spot without translating. Our scooter, on the other hand, is more car-like, and cannot really turn without translating.

The Bicycle Kinematic Model is a popular and consistent[14] method used to simplify car-like vehicle dynamics. As seen in Figure 2.4, due to the two wheel, front steer design, it is a great representation of a scooter's dynamics.

Important variables from the real scooter used to configure our dynamical model are:

1. The wheelbase, L.

2. Minimum turning radius, which is the turning radius R when steering angle δ is at the

maximum deviation from the center.

The wheelbase can be calculated simply by measuring the wheel to wheel distance. For the minimum turning radius, I set the scooter's steering to the maximum angle and make it turn in circles (Either by teleoperating the scooter or by directly writing commands to the ODrive using *odrivetool*). By measuring the diameter of these circles, we can get the minimum turning radius.

2.3.3 Expected Movements to Motor Commands

For a given command from the path planning module, with forward or backward drive velocity v, the drive motor RPM is simply: $RPM = v/(2\pi r)$, where r is the drive wheel radius. And, for a given command to turn the steering wheel, the steer motor position would be the product of the expected turn angle and the gear ratio between the pulley-belt mechanism connecting the handlebar with the steer motor. The ODrive is configured in position control mode for the steer motor, and accepts desired position angle in radians.

These calculations to convert commands from the path planning package to values to send to the ODrive are performed in a separate ROS node, which we call the *kinematics node*. The output of the *kinematics node* is then sent to the *odrive node*, which actually performs the task of talking to the ODrive to set the motors in motion. We will touch upon this in Chapter 4.

Chapter 3: Experimental Testbed

Developing the experimental testbed involves taking an OEM scooter and modifying/retrofitting it with components to enable autonomy. This process is described in this chapter, along with considerations for component placement, mechanical design and electrical design.

3.1 Our Scooter Fleet

As of writing, ReZoom's scooter fleet consists of three scooters as shown in Figure 3.1. The scooters were developed incrementally, starting with Ben Parker, to the upcoming Peter Parker. Each scooter builds over the previous version in terms of mechanical design, component choice and autonomy capabilities. The key differences are summarized in Table 3.1

	V1: Ben Parker	V2: Richard Parker	V3: Peter Parker
Stabilizing Mechanism	Naturally stabilized	Rigid support wheels	Flexible supports with suspension
Camera Placement	Handlebar	Fixed to base	Fixed to base
Camera(s) Used	Intel RealSense T265 and D435i	Stereolabs ZED2i	Stereolabs ZED 2i
Compute	Jetson Nano	Jetson Nano/Orin	Jetson Orin
User Rideable	No, small and unstable	No, deck not clear	Yes!

Table 3.1: The key differences between each ReZoom scooter.

Figure 3.1: All three ReZoom scooters; (L–R) Ben Parker, Richard Parker and Peter Parker.

3.1.1 Ben Parker

The first ReZoom scooter, based on the Hoover-1 Switch scooter, is the smallest and most adorable of the three (Figure 3.2).

Ben's chassis is a 2-in-1 design, combining a skateboard and an electric scooter. The rear wheels on the back would naturally stabilize the scooter—but not very well.

Some of ReZoom's early configuration, localization and planning tests were confined indoors, and Ben proved to be an excellent testbed. However, because of his smaller wheels and high center of gravity due to the added cameras and no rider, Ben would be highly unstable in autonomous tests outdoors and would topple over at the slightest of bumps.

Specifications:

Figure 3.2: Ben Parker, our first prototype autonomous e-scooter.

- Compute: Nvidia Jetson Nano
- Motors: ODrive D6473 150KV Brushless for Steer and stock drive motor.
- Motor Driver: ODrive v3.6 24V
- Vision: Intel Realsense D435i and T265 cameras
- Power: 6S LiPo packs placed on the deck

3.1.2 Richard Parker

Richard Parker, as seen in Figure 3.3, is a modified Mi M365 Scooter which is a popular scooter in the electric scooter community.

Most of the work developed in this thesis is tested on Richard. Unlike Ben, Richard is not naturally stabilized, and is fitted with support wheels instead.

Specifications:

Figure 3.3: Richard Parker, our second prototype autonomous e-scooter. Most of the work in this thesis has been developed on Richard.

- Compute: Nvidia Jetson Orin, and previously the Nano
- Motors: ODrive D6473 150KV Brushless for Steer and stock drive motor.
- Motor Driver: ODrive v3.6 56V
- Vision: ZED 2i with Polarizer and 2.1mm Focal Length
- Sensing: 2x PhigetSpatial Precision IMUs and Emlid Reach M+ GPS modules
- Power: Stock 10S, 7.2Ah Li-ion battery pack

3.1.3 Peter Parker

Peter Parker is the latest scooter being added to our fleet. Peter is based on the Hiboy S2 scooter platform, and the chassis is very similar to the Mi M365 model.

Figure 3.4: Peter Parker, our third and latest prototype autonomous e-scooter. Peter is still under development and not yet operational.

Peter is equipped with a better motor mount (which Richard later adopted), better support wheels, and improved placement for electronics which allows the deck to be clear for a user to ride the scooter.

- Compute: Nvidia Jetson Orin
- Motors: ODrive D6473 150KV Brushless for Steer and stock drive motor.
- Motor Driver: ODrive v3.6 56V
- Vision: ZED 2i with Polarizer and 2.1mm Focal Length
- Sensing: 2x PhidgetSpatial Precision IMUs and Emlid Reach M+ GPS modules
- Power: Stock 10S, 7.2Ah Li-ion battery pack

Figure 3.5: The old motor steering mechanism on Richard Parker

3.2 Mechanical Design

Two major aspects of the mechanical design include—gear and motor mount to move the handlebar and steer the scooter (A task which is normally done by a human rider); and some form of stabilization to keep the scooter upright. Other tasks include placement of sensors and supplementary components.

3.2.1 Steering Mechanism

We use 3D printed structures along with belt and pulley mechanism for the steering mechanism. This allows us to turn the handlebars at the desired angle. Encoders placed on the motor shaft precisely detect the angle at which the steering is turned.

The older motor steering mechanism was held on to the handlebar by a C-shaped clasp as shown in Figure 3.5. This design would cause the mount to sag and twist around the handlebar, and often required manual correction. A plastic bumper was attached at the bottom of the mount to protect the stereo camera from accidental bumps.

Figure 3.6: The new motor steering mechanism on Richard Parker is a big improvement over the previous, fixing any sagging or turning issues.

After months of testing this motor mount on Ben and Richard, we improved upon the design and fixed the mount sagging and twisting around the handlebar. The newer motor mount, shown in Figure 3.6, is much sturdy and allows easy removal of the drive pulley.

This mount has been designed for Peter Parker, but has also been adopted by Richard Parker.

3.2.2 Stabilizing Mechanism

The design of the support wheels on Richard Parker is simple, with stiff aluminum bars and castors. It offers good stability, even over large bumps and pits, but the lack of suspension made the scooter's rear wheel lift off the ground in certain situations (As shown in Figure 3.7). Also, ignoring the components placed on the deck, the scooter is unpleasant to ride on because of the stiff support wheel mounting.

Peter Parker has improved support wheels with a springy suspension (Figure 3.8) that allow a user to ride the scooter comfortably, while offering good support for autonomous movements.

Figure 3.7: The rear wheels on Richard lift off the ground over certain uneven surfaces because of the stiff support wheels on either side.

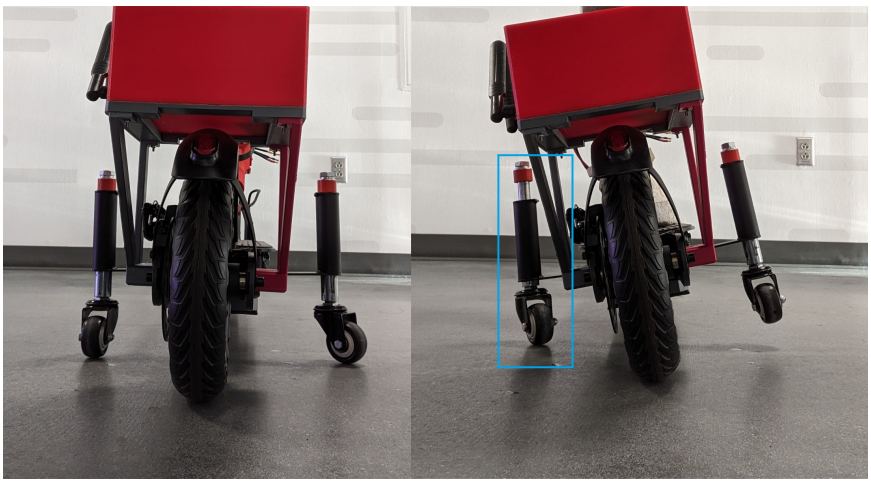

Figure 3.8: Peter's new support wheels have suspension to allow for tilt and improved ride quality.

In the future we could consider alternative designs for a support mechanism, for example, support wheels that could be kicked up/down when needed, similar to the Go X Apollo (as seen in Fig. 1.3).

Another possible method to stabilize the scooter without the use of support wheels would be to use a reaction wheel as done by researchers at Stuttgart (Fig. 1.2). I explored this method in simulation (using Gazebo), but there are a few potential drawbacks to consider for our intended use in the future. First, spinning the reaction wheel would tap into the scooter battery reserve and reduce range, requiring it to charge more often. I expect this reduction in battery life to be significant, but it needs further examination. Secondly, support wheels are both mechanically and financially easier to mount and implement. Regardless of these, a reaction wheel might still be a good alternative to consider in future ReZoom scooters.

3.2.3 Component Mounting and Placement

3.2.3.1 Overview of the Scooter's Components

A general overview of a ReZoom scooter's components and their placement is as shown in Figure 3.9. Richard Parker had some of his components on the deck which does not allow a user to ride the scooter. The upcoming Peter Parker has been designed with user ridability in mind, and the deck is kept clear.

3.2.3.2 Camera Placement

Placement of the stereo camera on the scooter was an important design choice. On Ben Parker, which used the Intel Realsense depth and tracking cameras, they were placed on top of

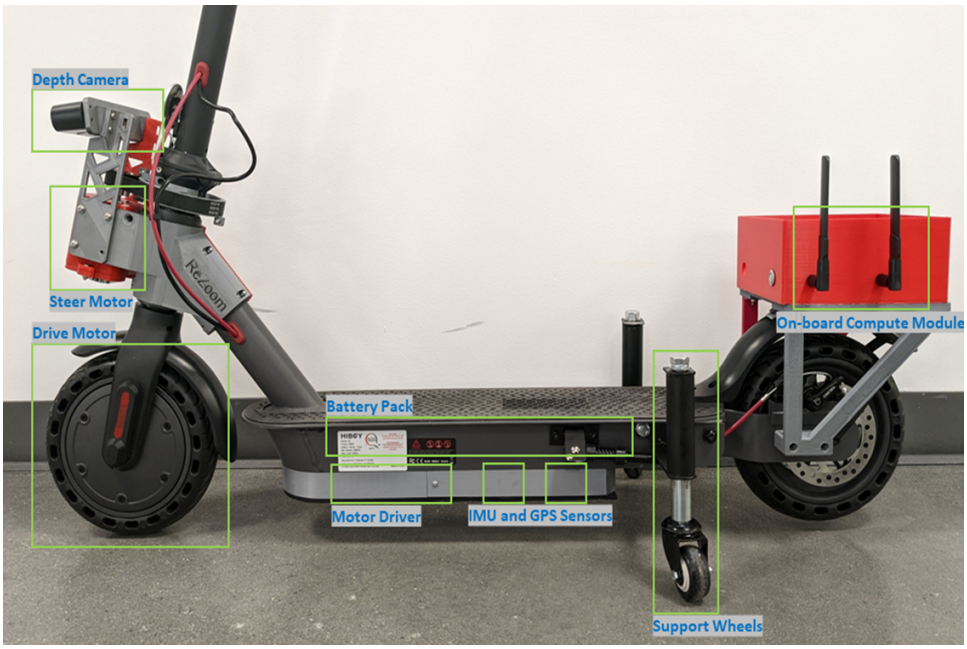

Figure 3.9: A picture of Peter Parker showing the general layout and build of a ReZoom scooter.

the handlebar as shown in Figure 3.10. With this configuration, the camera turns as the handlebar turns. This was detrimental to localization performance, as we noticed that with quick movements of the handlebar, (1) the camera would lose tracking; (2) the coordinate transform of the camera w.r.t. the robot's base, necessary to process the image correctly would not update quickly enough.

We then decided to have the camera fixed to the base of the scooter, and also moved to the ZED2i camera platform because of problems[1] and an end-of-life notice from Intel for the T265[16].

3.2.3.3 Placement of IMUs

Placement of IMUs are not as critical as ROS makes it easy to handle the necessary transformations from any sensor axes. The ZED2i has an internal IMU, and the Phidgets have been placed low and close to the center of the robot to keep the lever arms short[17, 18]. We also en-

[1]This post summarizes the reasons why we stopped using the Realsense T265: https://msadowski.github.io/Realsense-T265-First-Impressions/

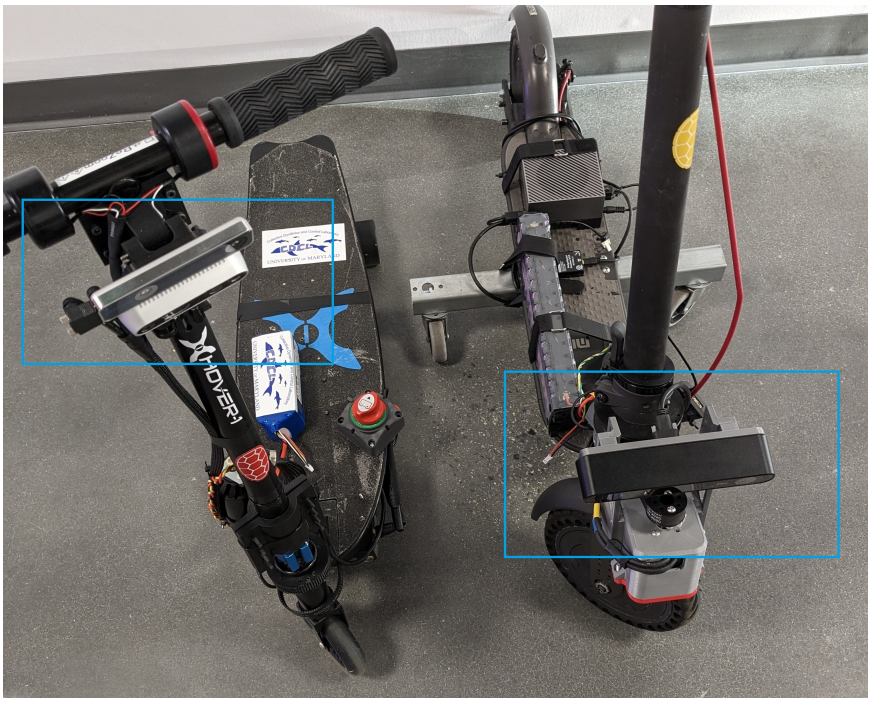

Figure 3.10: A picture comparing the camera placement on Ben Parker vs Richard Parker. Ben's camera turns with the handlebar, while Richard's does not.

sure that the IMU axes are aligned with the robot's base_link axes to keeping the transformations simple and easier debugging.

3.2.3.4 Placement of GPS Antenna

The GPS antenna needs to have a clear view of the sky 30 degrees above the horizon[2]. Also, to avoid multipath[19], we add a ground plane under the antenna as seen in Figure 3.11. The manufacturer recommends the ground plane to be no less than 70X70 mm.

Previously, I have also used the Adafruit Ultimate GPS MT3339 modules instead of the Reach M+, but the performance was found to be inferior. The MT3339 modules would sometimes be as much as 60 meters inaccurate; though it has not been tested if the accuracy can be improved with better mounting and a ground plane as used with the Reach M+.

[2]GPS antenna placement for the Emlid M+: https://docs.emlid.com/reach/before-you-start/antenna-placement/

Figure 3.11: A picture showing the placement of GPS antenna with the metal ground plane added at the bottom to reduce multipath.

3.3 Electrical Design

3.3.1 Connection Diagram

The scooter's electrical connection diagram is as shown in Figure 3.12.

The wiring has to be done keeping in mind the high current pathways, using appropriate gauge wires, and keeping distances short where necessary. Also note that the ZED 2i stereo camera is a high bandwidth device, and is thus connected directly to the Jetson's USB port instead of a USB Hub.

3.3.2 Ground Loops

We initially faced a problem where the ODrive would (seemingly) randomly disconnect from the Jetson during operation. After troubleshooting, I realized that the problem was because

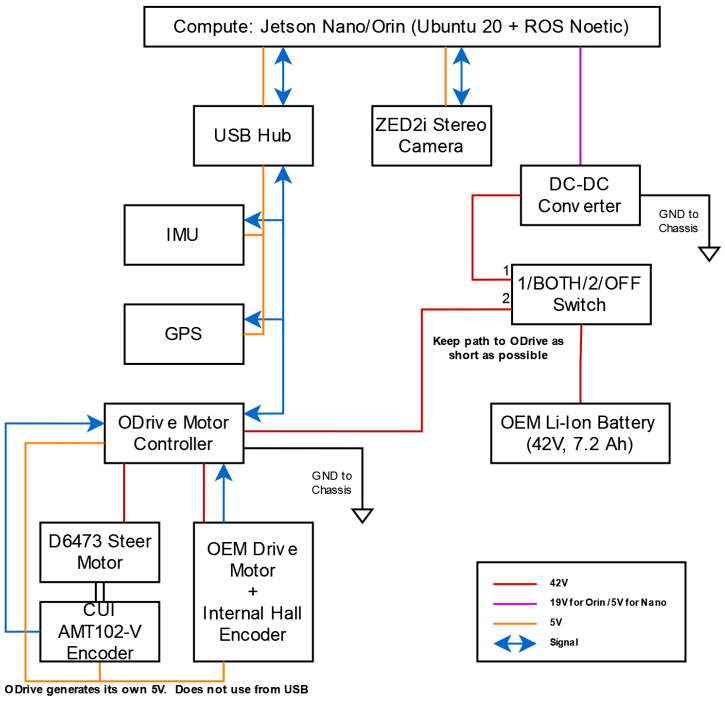

Figure 3.12: A block diagram showing our electrical connection diagram.

of the formation of a ground loop[3]. As shown in Figure 3.13, the high fluctuating current drawn by the ODrive while driving the motors, combined with the parasitic inductance of wires, create a voltage difference between the ODrive's ground and the Jetson's ground. This causes a current to be injected in the Jetson's USB port, causing it to disconnect the device.

Keeping the wires short reduces the parasitic inductance, but the best solution that worked for us was to use the scooter's chassis as a ground plane (Figure 3.14). This essentially provides a large, low inductance pathway for ground currents, thus minimizing the effect.

3.3.3 Decoupling Capacitors

The scooter's OEM drive motor includes an internal hall effect encoder with a Counts Per Revolution (CPR) of 90. The optical encoder (part number CUI AMT102-V) added for the steer-

[3]Read more about ground loops on ODrive Docs: https://docs.odriverobotics.com/v/latest/ground-loops.html

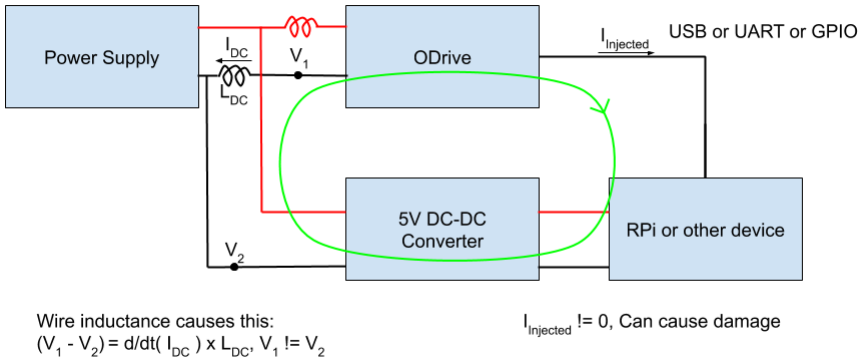

Figure 3.13: A block diagram showing the cause of ground loops when a high, fluctuating current is drawn by the ODrive. Source: ODrive Docs[3]

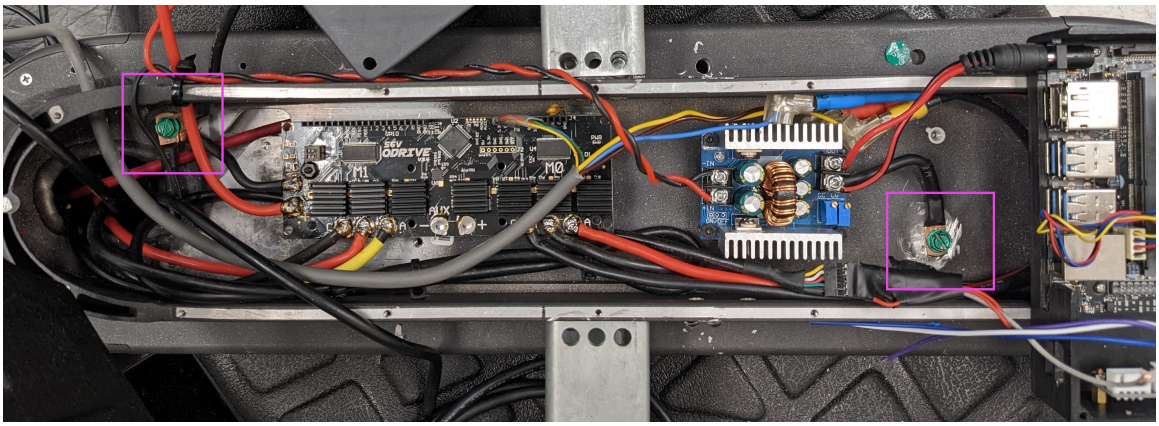

Figure 3.14: The ground to chassis connections are shown (in purple boxes) in a picture of Richard Parker's underbelly. Note that the connections are made close to the ODrive (on the left), and between the DC-DC converter and Jetson (on the right).

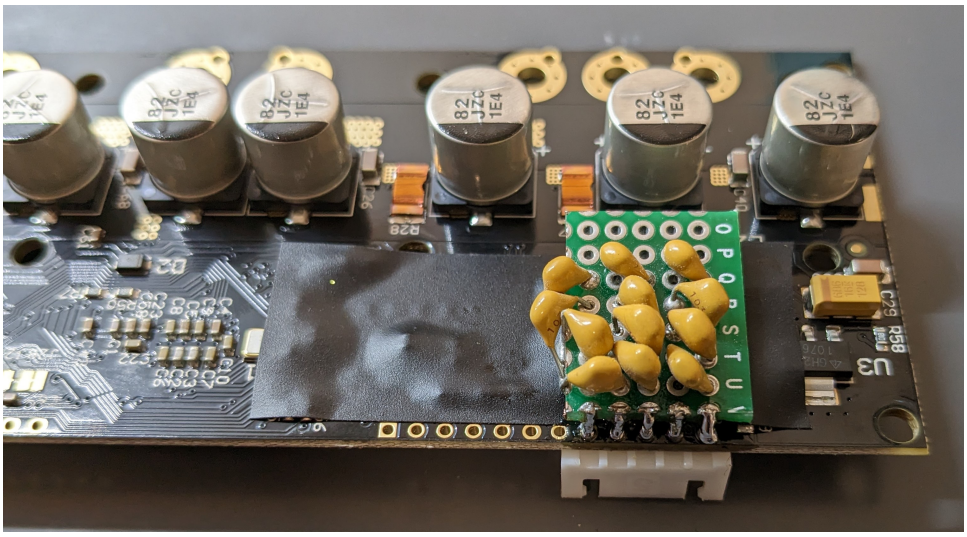

Figure 3.15: Noise decoupling capacitors were added to the hall effect encoder pins close to the ODrive.

ing motor in comparison, has CPR of 8192. Hall effect encoders are therefore not nearly as good as optical encoders, but they are easy and cheap to integrate as part of the motor. Another drawback of hall effect encoders are that they are highly susceptible to noise, and therefore shielded cables must be used whenever possible. In our scooter, this noise caused errors in motor position measurement[4], and we had to include decoupling capacitors as shown in Figure 3.15. These capacitors, ranging from 47pF to 47nF, must be added close to ODrive to remain effective.

3.3.4 Component Current Draw

A common error which leads to erratic component/sensor behavior is to not meet the current draw requirements for each device. It is important to consider this while adding any new device to a power source, especially when multiple devices are connected to the same USB port through a USB Hub. In such a situation, all devices draw current through the same USB port, which has

[4]We faced ERROR_ILLEGAL_HALL_STATE errors as decribed here:
https://discourse.odriverobotics.com/t/encoder-error-error-illegal-hall-state/1047

a rating of 0.9A[5]. The individual devices themselves would have been designed with the USB power specification in mind, not considering for other devices connected to the same port.

We can find a device's current requirements in its datasheet; for example, the PhidgetSpatial Precision IMUs we use have a maximum current draw rating of 21mA, while the Emlid M+ GPS requires a maximum of 500mA. It would thus be safe to connect these into a USB Hub drawing power from a single USB port on the Jetson.

3.3.5 Analog/Digital Pathways

Another design choice while placing components is how far or close they can/should be placed to each other. For example, the scooter's thumb throttle is a potentiometer with the various throttle positions being effectively measured as an analog voltage varying between 0–3.3V. If the Jetson wanted to read these throttle inputs[6], we need to route this signal from the top of the handlebar, all the way to the bottom rear of the scooter (where the Jetson is placed). These 0–3.3V analog signals would be highly susceptible to noise over that length of wire, and would almost certainly lead to incorrect measurements. So instead, we would have to digitize the throttle information close to the throttle, using an ADC module[7] or even an Arduino, and then route the digital signal (e.g. UART/SPI/I2C/CAN) to the Jetson.

To summarize, keep in mind the amount of current while deciding wire guages; the type of signal and signal quality while choosing wire length; current supplying capacities of USB ports; and the bandwidth required by USB devices.

[5]Some devices allow for a larger current draw, but its safe to assume 0.9A as the limit for USB 3.0 ports.

[6]Jetson devices do not have an in-built ADC, so they cannot read analog signals on their own. But the explanation remains the same if we assumed to have an ADC module close to the Jetson.

[7]An example of an ADC module for the Jetson would be: https://www.adafruit.com/product/1083

Chapter 4: The ReZoom Autonomy Stack

This chapter discusses the localization, planning and navigation stack of the ReZoom scooter.

4.1 Localization

4.1.1 The Localization Problem

As described by Leonard and Durrant-Whyte (1991)[20], the problem of building autonomy for a robot can be most simply broken down into answering the following three questions: "Where am I?", "Where am I going", and "How should I get there?". The answer to the first question can be found through the robot's various sensors, and is the localization problem we are trying to solve.

Assuming the scooter moves on a plane, it's location in an earth fixed frame is completely described by two position coordinates and an orientation. In an ideal world, we can trivially calculate and keep track of this location by using values reported by the scooter's sensors such as an IMU, wheel encoders and GPS. However, in the real world, sensors are not perfect, and there is some uncertainty associated with any measurement we get. Further, some sensors are more or less accurate than others, and this accuracy can even vary with external factors, including time.

Therefore, we need a statistical approach to calculate the most *optimal*[1] estimate of our robot's position and orientation.

4.1.2 The Extended Kalman Filter

The Kalman Filter[21] is a commonly used state estimator which can perform this task by combine sensor data from multiple sources. It can even estimate the system states indirectly, for example, compute position using acceleration or velocity information only. The Kalman filter, and the Extended Kalman Filter (EKF) are heavily documented in literature, and I suggest the reader to refer [22, 23] for a theoretical background and derivation, or [24] for a practical overview. The important thing to note is that the Kalman filter (not EKF) is a statistically optimal (in minimum mean square error sense) observer for a *linear* system with Gaussian uncertainties (disturbance and noise). It is the best we could possibly do for a linear system [22] under these conditions.

However the Extended Kalman Filter is an extension of the Kalman Filter for nonlinear systems, and is not an optimal filter. It is essentially a linearization of the nonlinear system around each estimate, and is effective only if the estimation errors remain small. The filter can and will diverge if the initial estimate and uncertainties are large (See [22], section 7.2. The authors call the EKF an *ad hoc* filter).

Better nonlinear filtering methods exist, such as the Unscented Kalman Filter (UKF)[25] and Monte Carlo Localization[26], but the EKF is arguably the most popular and easiest to get started with.

[1] *Optimal* may refer to optimality in the sense of minimum mean squared error, minimum variance or some other optimality criterion. It is important to ask: "optimal in what sense?"

4.1.3 The Robot Localization package

The robot localization ROS package is a collection of two state estimation nodes, `ekf_localization_node` and `ukf_localization_node`. Additionally, it contains the `navsat_transform_node`, which helps in integrating GPS data into the EKF or UKF nodes. The package is well documented[2], and the authors have also published their results as a conference paper [27].

The `ekf_localization_node` keeps track of 15 state variables for any robot:

$$[X, Y, Z, roll, pitch, yaw, \dot{X}, \dot{Y}, \dot{Z}, \dot{roll}, \dot{pitch}, \dot{yaw}, \ddot{X}, \ddot{Y}, \ddot{Z}]$$

The node allows fusion of any arbitrary number of sensors, and not all state variables need to be measured by each sensor.

In our setup, we assume that the scooter operates in a planar environment, and therefore configure `ekf_localization_node` in 2D mode. In this case, measurements fused for Z,$roll$,$pitch$,$\dot{Z}$,$\dot{roll}$,$\dot{pitch}$, and $\ddot{Z}$ are internally set to zero, even if they are included in sensor measurements.

Another thing to note is that the robot localization package assumes an omni-directional motion model for robot dynamics as it is intended to be a general purpose estimation package for all robots. However, the scooter is clearly not omni-directional, and the scooter's unique kinematics are not considered anywhere in this package. This introduces some error into the filter's predict stage, which gets corrected in the update stage. Presently, this scheme works in our setup, but in the future, it should be beneficial to modify the filter to account for the robot's

[2] `robot_localization` Wiki page: http://docs.ros.org/en/noetic/api/robot_localization/html/index.html

dynamics[3].

4.1.3.1 Local and Global EKF Nodes

Sensors like an IMU or wheel encoders measure accelerations and velocities, which are then integrated by the `ekf_localization_node` to calculate *relative* position from start. As the robot covers more and more distance from start, error in position gets accumulated, and can increase without bound. Therefore, these sensors are not a good long term reference. GPS on the other hand, gives us position estimates with a bounded error. The values remain more or less the same regardless of how much the robot travels, i.e., the error for GPS is dependent on the satellite to antenna signal quality, and not distance traveled. However, getting a reading from a GPS sensor is slow (at a rate of about 1Hz for us), and is not guaranteed throughout the mission, for example when urban structures overshadow the GPS antenna. IMUs and wheel encoders will report measurements at a fixed rate as long as the robot is running.

When fusing sensors like the IMU or wheel encoders into our EKF, we ensure that the output of the EKF remains smooth (at a fast rate) and continuous (no sudden jumps). But when we include measurements from GPS, which can arrive at random time intervals, the EKF output will also be subject to sudden jumps. These sudden jumps would be bad when, for example, the robot tries to navigate around an obstacle.

To get the best of both worlds, following REP-105 [10], we use the `map-odom-base` coordinate frame setup. To get each of the transformations, i.e. `map-odom` and `odom-base`, we use two independent instances of the EKF, which are referred to as the global EKF and local

[3] See relevant discussion: https://answers.ros.org/question/221837/robot_localization-ekf-internal-motion-model/, and https://answers.ros.org/question/281513/kinematic-model-used-in-robot-localization-package/

EKF, respectively. In our present setup, the only difference between the two is that the global EKF includes measurements from a GPS sensor, whereas the local EKF does not. I explain this further in the next section. [28, 29] are also helpful resources that go over this.

4.1.3.2 Coordinate Frames for the Scooter

Figure 4.1 illustrates all the coordinate frames for the scooter, and is called the TF tree. The tf2 package[4] in ROS keeps track of all coordinate frames, updating them and sharing transforms between any two frames upon request. This is one of the core ROS packages and is preinstalled.

A brief description of some important frames are given below:

- `utm`: This is an earth fixed frame with the origin always at the local UTM zone origin, and ENU oriented (x-axis points grid east, y-axis grid north, z-axis up). Note that Grid North varies from True North (which always points to the north pole) by about 1.2 degrees, which is a result of flattening the earth to generate UTM planes. `navsat_transform` node will automatically add/subtract this angle when generating the `utm` to `map` transformation.

- `map`: This frame is initialized by the scooter at a earth fixed latitude-longitude (LL), and heading as defined by the AprilTag it sees during initialization (see subsection 4.1.6). At the time of writing, orientation of `map` has x-axis pointing to the magnetic east, though we can change it to any desired heading. Figure 2.3 illustrates a typical `utm` to `map` transformation. Once initiated, `map` does not move w.r.t. `utm` throughout the mission unless we want to set it at a new LL.

- `odom`: This frame is initialized by the scooter perfectly aligned (zero translation and rota-

[4]Read more about tf2 on its wiki page: http://wiki.ros.org/tf2

tion) with the base_footprint frame. The odom frame is earth semi-fixed, as it moves (w.r.t. the map frame) to compensate for long term errors accumulated in the odom to base_footprint transformation, which is calculated by the local EKF. This movement of odom is an effect of the global EKF correcting the robot's position, base_footprint w.r.t. map. Global EKF calculates the map to base_footprint transform, but since base_footprint already has odom as its parent, global EKF publishes the map to odom transform and effectively moves odom around to correct the robot's position, i.e. base_footprint. Local path-planning and obstacle avoidance is done in odom.

- base_footprint: This frame is the top-most parent of the scooter's URDF, and all parts of the scooter begin here. base_footprint is just a projection on the ground of base_link. As the scooter moves, the local EKF handles the transformation between odom and base_footprint, by combining only the continuous (IMUs, Wheel Odometry) sensor information. All other frames after base_footprint are spawned by the URDF (robot model) file, and for most intents and purposes, we only need to worry about the position of base_footprint.

We will encounter references to the "world" frame in many ROS packages, though world is not a particular frame in our TF tree. Simply stated, world is the top-most relevant frame for any particular subtask in our autonomy stack, usually either map or odom. For example, for the global EKF, map can be considered as the robot's world frame, as this node does not really care anything above map on the TF tree. Similarly for the local path planner, odom would be the world frame as we only care about the given destination (which would be pre-converted to odom before sent to the planner), and how we get there.

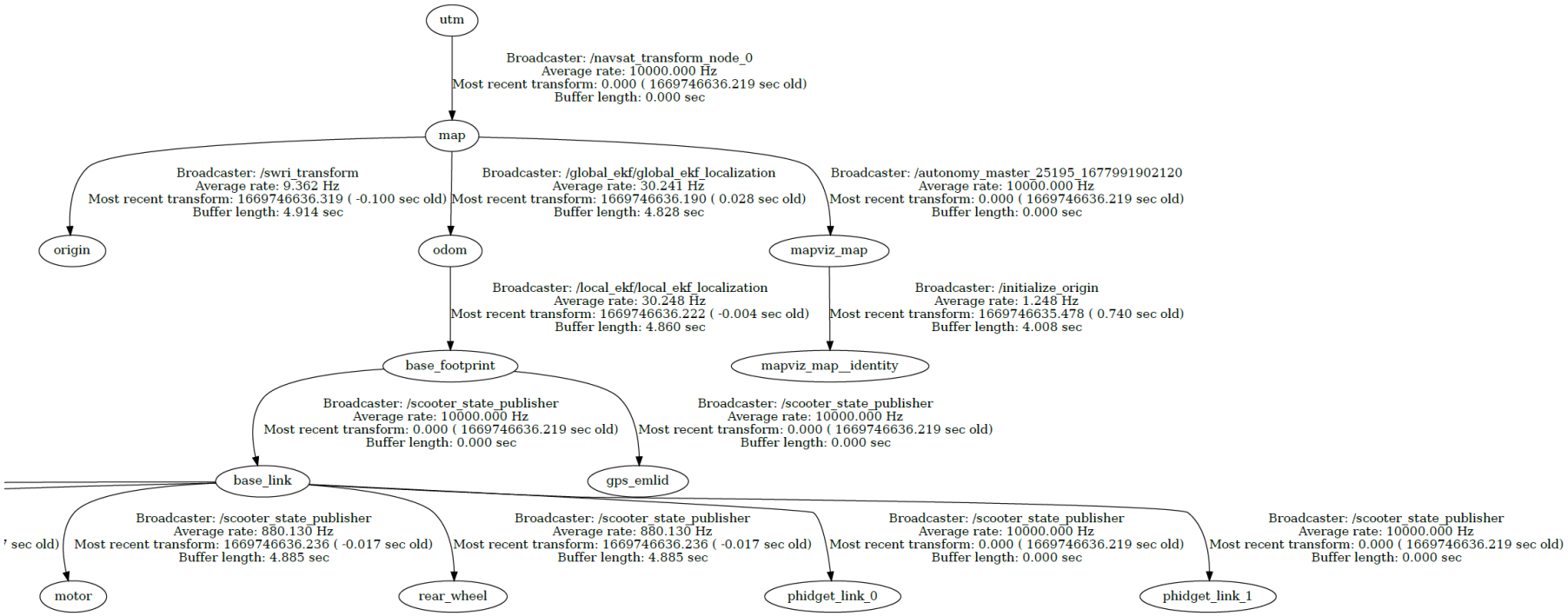

Figure 4.1: [Part A] All coordinate frames for the scooter shown in a TF tree generated using "`rosrun tf view_frames`". Image is split into two pages due to size.

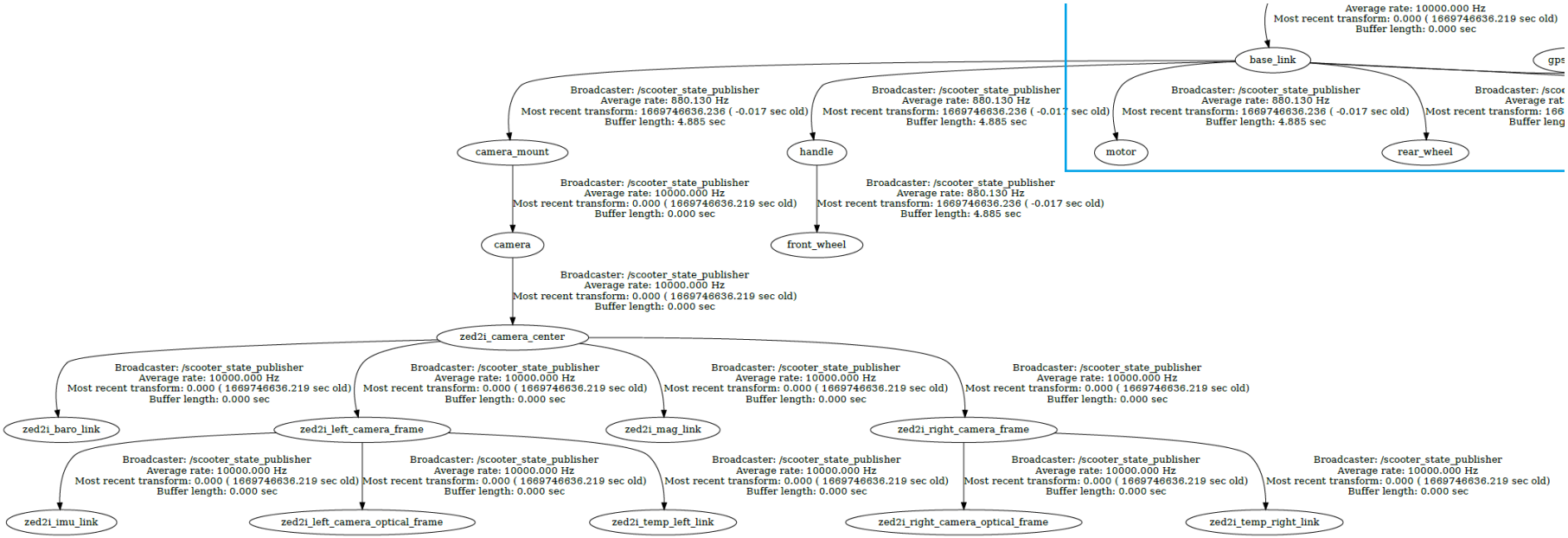

Figure 4.2: [Part B] All coordinate frames for the scooter shown in a TF tree generated using "`rosrun tf view_frames`". Image is split into two pages due to size.

4.1.3.3 Navsat Transform Node

A GPS sensor reports latitude and longitude coordinates. To fuse this into our EKF, we need to convert it to (X, Y) coordinates that are consistent with the filter's states, and robot's world reference frame. The `navsat_transform_node` does this conversion.

4.1.4 Localization Data Flow Overview

Figure 4.3 presents the EKF data flow for our setup. It highlights the measurements used from each sensor by the EKF, the transformations published and the relevant coordinate frames.

4.1.5 Problems with Visual Odometry

We have a camera onboard, so why do we not use Visual Odometry (VO) as part of our localization? This is because of two main issues, (1) Poor tracking when the environment is not feature-rich, and (2) self-shadowing. Methods exist in literature to mitigate both of these issues, at least in some part, but they were not explored in this work.

4.1.5.1 Poor Tracking with Low Features

The ZED 2i comes with its own Software Development Kit (SDK) with support for camera pose tracking. Using the ZED ROS Wrapper, we get the results of this pose tracking on ROS topics `/zed_node/odom` and `/zed_node/pose`. The first one is a pure visual odometry result whereas the latter includes saved spatial information from past runs[5]. In my tests, I have found these to work well in feature-rich environments but the performance is very poor in low

[5]https://www.stereolabs.com/docs/ros/positional-tracking/

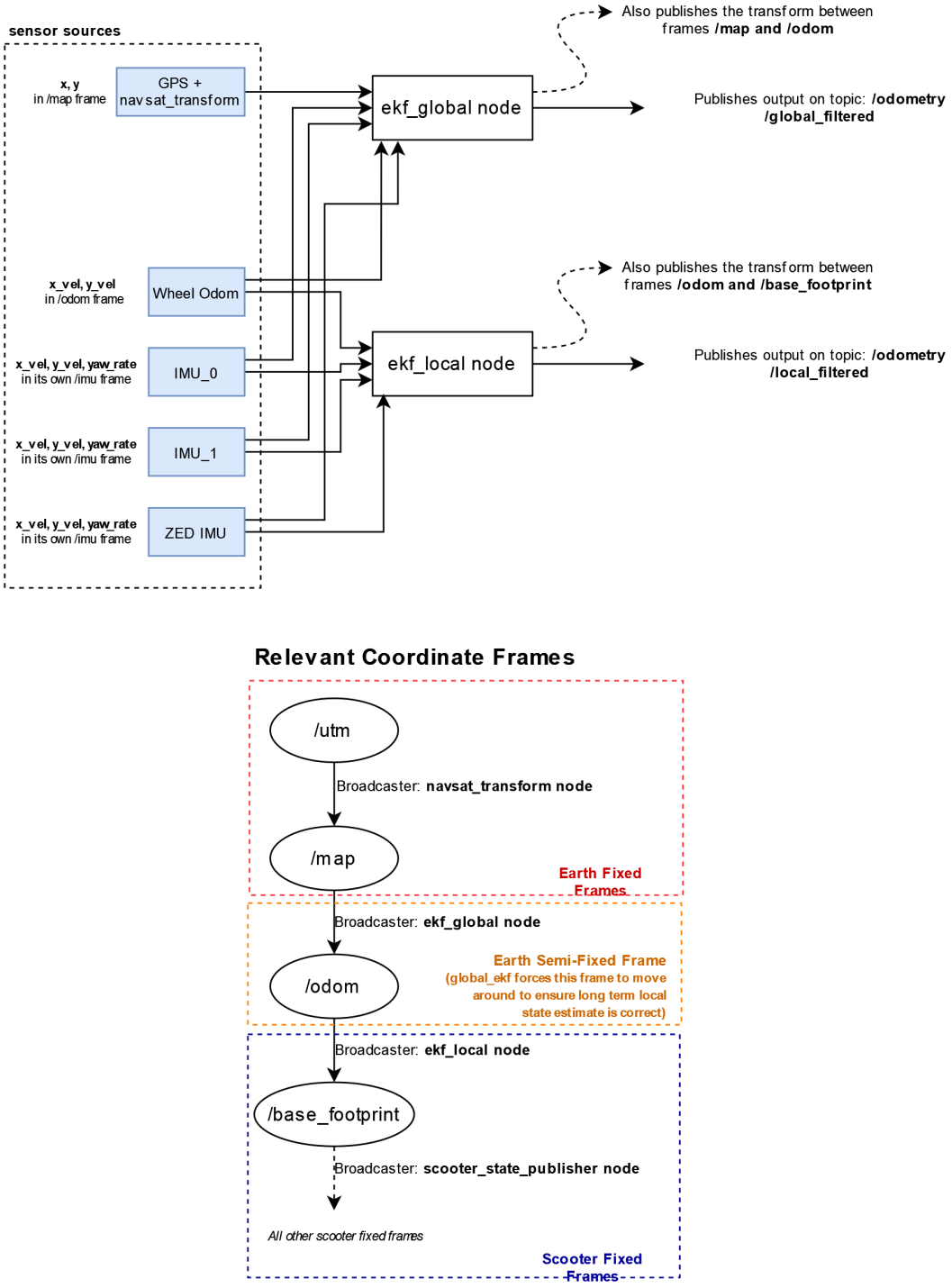

Figure 4.3: An overview of the EKF data flow for localization and relevant coordinate frames.

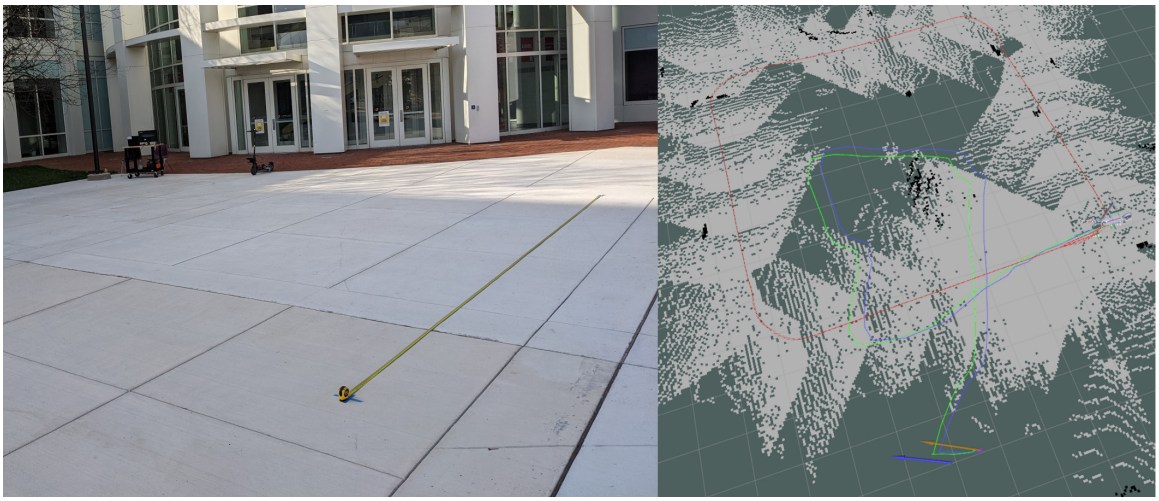

Figure 4.4: The scooter is made to follow an 8x8m square in Kim Plaza, a low feature environment. The figure on the right shows the output of our local EKF (Wheels and IMU) in red, and the ZED's `odom` and `pose` outputs in blue and yellow/green. The local EKF output follows the square well, but the ZED's tracking suffers.

feature environments. Figure 4.4 shows an example of this. Further, the covariance reported by the camera in these low feature environments did not reflect its inaccuracy, and therefore degrades the performance of our EKF. I have observed better results without it included.

4.1.5.2 Self-shadowing

Self-shadowing is the situation when the robot's shadow falls into the camera field of view, and the robot sees its own shadow is a static feature on the ground. This degrades performance of visual odometry as the robot uses its own shadow as reference and thinks that it moved less than it actually did. This was seen during a test of ORB SLAM2 on the scooter as shown in Figure 4.5. Other than this issue, ORB SLAM2 provides excellent results. ORB SLAM2 also includes a Visual Inertial Odometry (VIO) node which includes information from IMUs in pose tracking, and might provide better results in this situation. Due to runtime errors, I was unable to get the VIO node running on our setup and could only test the VO node. An improved method, ORB

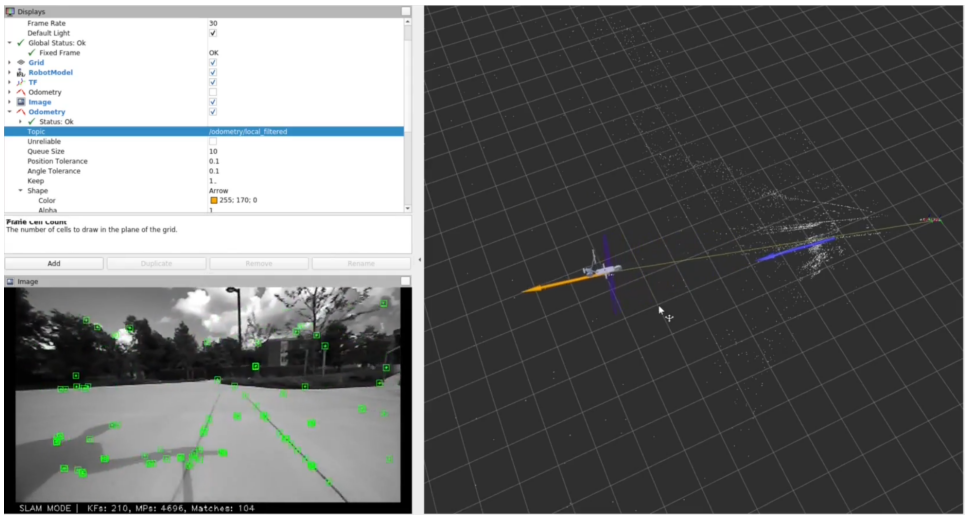

Figure 4.5: Robot pose estimated by ORB SLAM2 (blue) suffers and falls behind the local EKF estimate (yellow) as the robot (and my) shadow is seen in the image stream.

SLAM3 is also available, but I could not build and run it on the Jetson.

To avoid self-shadowing, a possible method could be to blur out a polygon containing the robot's shadow from the image feed. This polygon could be a function of time of day and the scooter's heading. Perhaps, based on these parameters, we could modify tracking covariance or entirely exclude visual tracking from our EKF. There are also other methods in literature[30] to avoid self-shadowing. These methods have not been implemented in this work.

4.1.6 Initialization

Since the EKF is not guaranteed to converge, we have to ensure that it is initialized as close as possible to the scooter's true state. An approximate initial position can be known from GPS sensor readings before starting up the EKF, but this is not reliable as the scooter often starts up in a scooter parking area that are along the edges of buildings. These buildings affect the quality of GPS measurements.

Further, an even bigger challenge is knowing the absolute initial orientation w.r.t. the earth,

because magnetometers(compass) are notoriously unreliable.

4.1.6.1 Problem with Magnetometers

Magnetometers work by detecting the relatively weak magnetic field of the earth, but are easy affected by hard-iron (magnetic field produced by motors or high current wires) and soft-iron (metals such as the chassis of the scooter can distort magnetic fields) errors [31]. It is possible to calibrate against soft-iron errors, but not as much against hard-iron errors which change dynamically. I have tried calibrating the magnetometers on the ZED and Phidget IMUs by following recommended steps[6,7] and physically rotating the scooter along all axes in 3D, but I still had up to 30 degrees error in orientation.

Our scooter does have three separate magnetometers on board (1 inside the ZED, 2 from the IMUs), so a method to obtain good absolute orientation should be possible with some methods suggested in literature [32–34]. But we do not explore these in this thesis, and instead, resort to other methods to get initial absolute position and orientation.

4.1.6.2 April Tags as World Reference Anchors

Once the scooter starts up, the idea is to have it recognize a unique feature in the environment which would help it localize itself. These unique features, which we call world reference anchors, are fixed at specific global locations. For ease of implementation, we use AprilTags [35, 36] as anchors in this thesis, but this can be extended in the future to any similar visual feature matching system.

[6]ZED2i Magnetometer Calibration Steps: https://www.stereolabs.com/docs/sensors/magnetometer/
[7]Phidget Spatial Magnetometer Calibration: https://www.phidgets.com/docs/Magnetometer_Primer

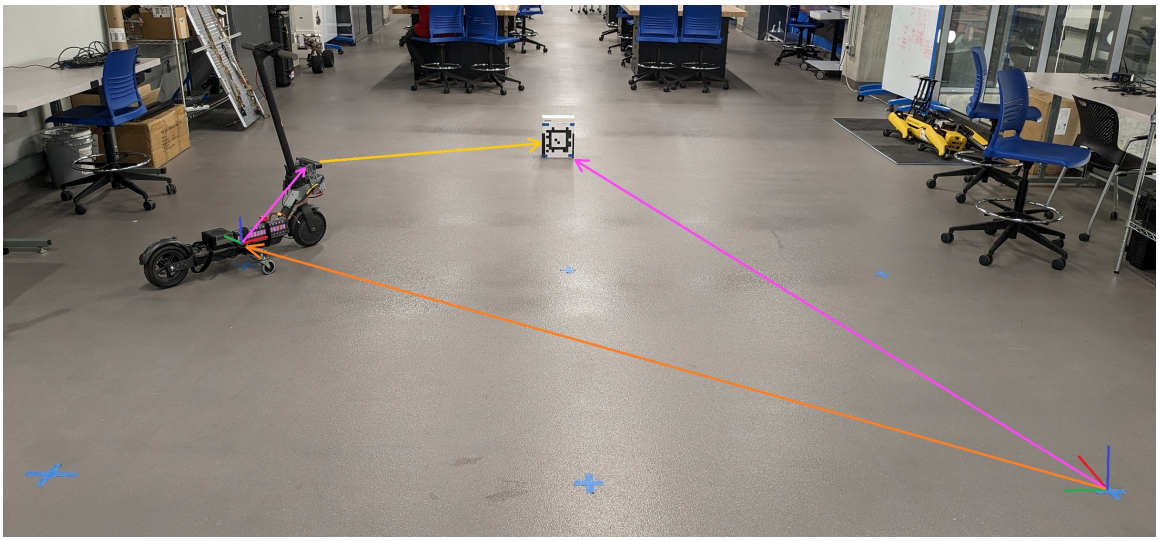

Figure 4.6: A picture showing our initialization test setup. Representative coordinate frames are drawn for `map` (at the bottom right) and `base_link` (on the scooter). The camera–tag pose (yellow) is observed and the base–camera and map–tag pose are known. We then compute the map–base transformation (orange) which is the initial position of the scooter in this setup.

AprilTags are a visual fiducial system, and can be detected in real-time with low compute requirements. Further, there is a ROS package available, called `apriltag_ros`[8], which takes in camera images and outputs the detected tags along with their pose[9] w.r.t. the camera. We use this as our reference to initialize the scooter's pose.

4.1.6.3 Initialization using AprilTags

Figure 4.6 shows the basic initialization problem and our test setup. The test setup consists of a grid marked with blue Xs on the floor with known position of the AprilTag. The `map` frame is assumed to be at one of these points, say at the bottom right corner, as shown in the figure. The goal is to compute the position of the scooter (orange) using known information (position of the tag and base–camera transformation, purple) and observed information (camera–tag pose,

[8] `apriltag_ros` wiki page: https://github.com/AprilRobotics/apriltag_ros
[9] The term pose refers to a combination of position and orientation

yellow). With ${}^{x}T^{y}$ representing a transformation from frame x to frame y, this can be easily done using properties of a rigid transformation matrix[37]:

$$[{}^{map}T^{base}] = [{}^{map}T^{tag}][{}^{tag}T^{cam}][{}^{cam}T^{base}] \tag{4.1}$$

and so,

$$[{}^{map}T^{base}] = [{}^{map}T^{tag}][{}^{cam}T^{tag}]^{-1}[{}^{base}T^{cam}]^{-1} \tag{4.2}$$

where each transformation on the right hand side is either known or observed.

We then extend this idea outdoors, where when a tag is placed at a fixed location, two things are associated with it. (1) The latitude, longitude and heading of the tag is known, we call this `tag_llh`, and (2) The latitude, longitude and heading of the desired `map` frame location is set, we call this `map_llh`. The second part is useful because setting `map` at a fixed location on the earth is helpful to use it as a reference when restoring any other previously stored databases. For example, we might have a pre-recorded map that we want to restore and use, or we might want to create plots of multiple robot missions. It also allows `navsat_transform_node` to convert GPS measurements precisely. Having `map` at a predictable, known location allows us to confidently use it as a reference for the mission.

With `tag_llh` and `map_llh` known, initialization follows the steps shown in Figure 4.7. The three steps are:

1. We receive the ID and pose of the tag w.r.t. the camera from `apriltag_ros`, therefore $[{}^{cam}T^{tag}]$ is known.

2. From the detected tag ID, we check param file `april_tags.yaml` to retrieve `tag_llh`

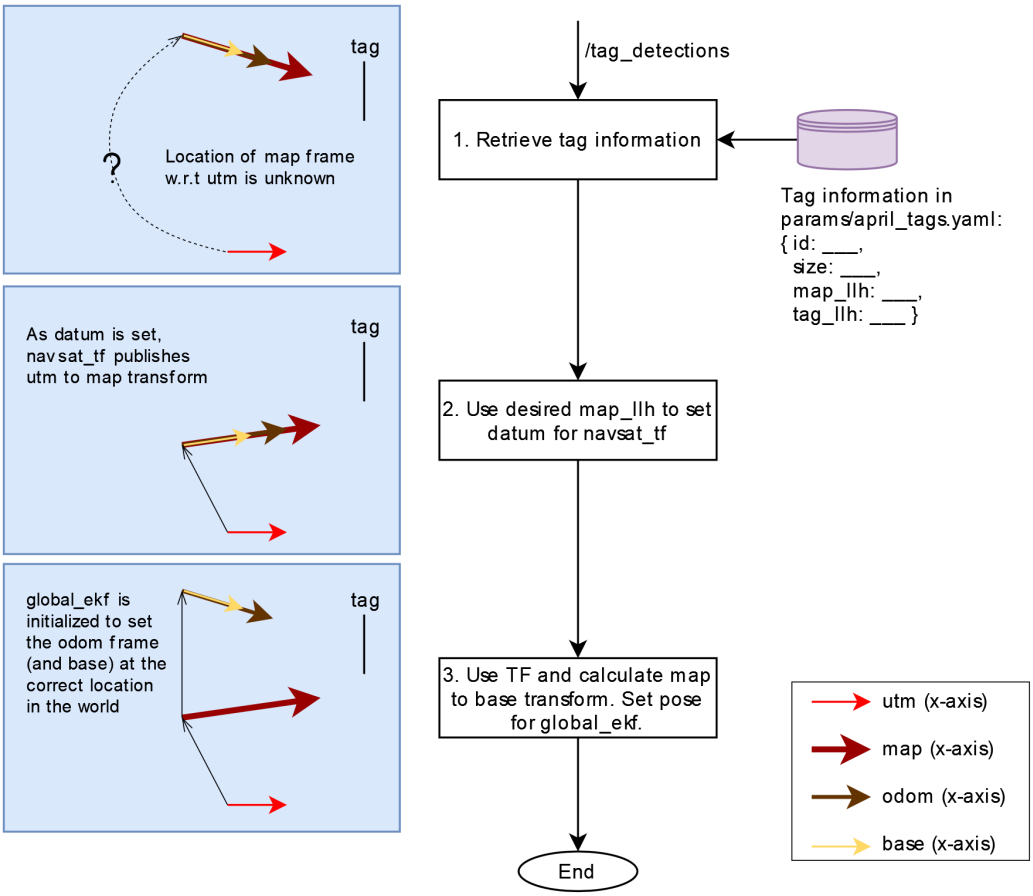

Figure 4.7: A flow diagram showing the steps taken to initialize coordinate frames correctly using an April Tag as world reference.

and `map_llh`. `map_llh` information is sent to `navsat_transform_node` using its `set_datum`[10] ROS service. `navsat_transform_node` will then create and publish the `utm` to `map` transform.

3. Finally we:

   (a) Use `tag_llh` to get $[{}^{utm}T^{tag}]$.

   (b) Get $[{}^{utm}T^{map}]$ which is published by `navsat_transform_node`.

   (c) $[{}^{base}T^{tag}] = [{}^{base}T^{cam}][{}^{cam}T^{tag}]$

   (d) $[{}^{map}T^{base}] = [{}^{utm}T^{map}]^{-1}[{}^{utm}T^{tag}][{}^{base}T^{tag}]^{-1}$

   (e) We now have the pose of the robot in `map`, and this will be set as the initial pose for the global EKF using the `set_pose`[11] ROS service.

This procedure is performed in the `init_from_tag_node` in our autonomy stack. The scooter and state estimation nodes are now initialized correctly, and can proceed with the mission.

4.2 Planning and Navigation

4.2.1 Move Base

`move_base`[12] is a major component of the ROS navigation stack, and provides an interface for accepting goals, path planning, and sending movement commands to the robot. An overview of `move_base` is shown in Figure 4.8.

[10] http://docs.ros.org/en/noetic/api/robot_localization/html/integrating_gps.html
[11] http://docs.ros.org/en/noetic/api/robot_localization/html/state_estimation_nodes.html#services
[12] http://wiki.ros.org/move_base

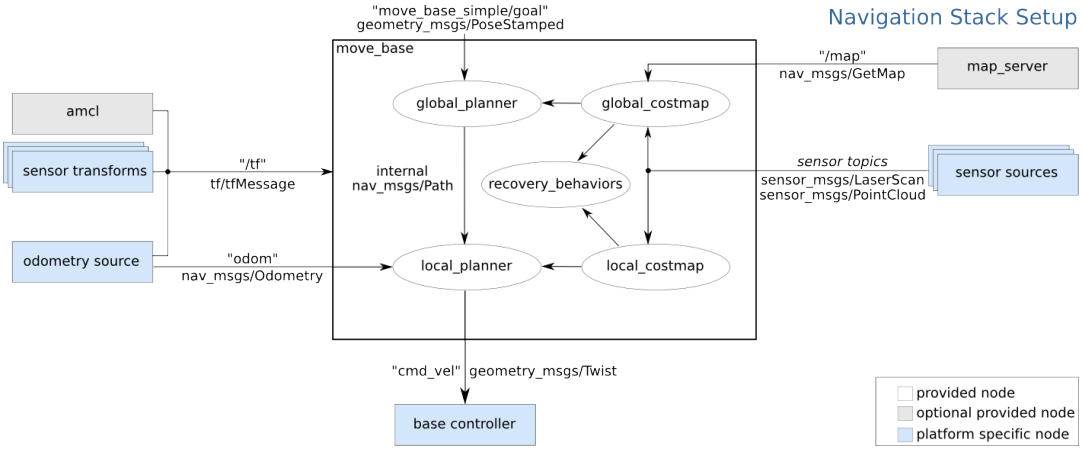

Figure 4.8: A high level view of move_base and its interaction with other components. Source: move_base wiki[12]

The general idea is to use two levels of path planners—local and global. The global planner accepts a goal, and will then create a rough plan from the start to the goal (which could be quite far), without considering the obstacles which show up as the robot is moving through. For the planners, obstacles and free space are just marked areas in a 2D costmap[13]. We may not know beforehand all obstacles which can show up during the mission (e.g. pedestrians), but *may* have a general idea about the region (e.g. buildings, roads). The global planner passes this global plan to the local planner, which then creates a local plan, considering real-time information from the robot's sensors to carefully plan around objects in the immediate vicinity of the robot. The local planner generates movement commands (in the topic `cmd_vel`) that are then sent to the robot's `kinematics_node`[14], which converts them to required motor movements.

4.2.1.1 Passing Goals to Move Base

Our desired goal is a latitude-longitude (LL) coordinate on earth, but we cannot send this to `move_base` directly. This is because the global planner is only concerned with taking a costmap

[13] http://wiki.ros.org/costmap_2d

[14] We talked about the `kinematics_node` in Chapter 2. This is a node specific to our robot and implementation.

(the global costmap) and finding the best path toward the goal *in* this costmap. Therefore the goal must be in the global costmap's frame of reference.

Recall from subsubsection 4.1.3.1 that the `map` to `base` transformation may include abrupt jumps due to the inclusion of GPS. So if we create the global costmap in `map`, the goal could move around (and does indeed move quite a bit) w.r.t. the scooter, as the scooter is moving toward the goal, and the global EKF tries to correct the scooter's position on earth. With this setup, as the goal moves around, the local planner constantly tries to create a new plan, and performance is slow and sluggish.

If we instead, create the global costmap in `odom`, the goal remains fixed w.r.t the scooter as the local EKF does not experience any jumps. Planning is smooth and stable. The downside to this method is that, once the goal is assigned, and the scooter is moving toward it, any corrections to the scooter's pose in `map` done by the global EKF would not affect the scooter. This means that for longer goals, the goal that the scooter achieves would involve significant error (depending on error in the local EKF) as compared to the desired LL.

To minimize this error while keeping global costmap in `odom`, we pre-plan our route to the desired end LL into small, incremental LL waypoints. Each waypoint is only about 5 to 10 meters away, ensuring that we include the latest corrections done by global EKF every time a new waypoint is assigned as a goal to the global planner.

An alternative explanation to the ideas in this subsubsection are provided here[15].

[15] https://answers.ros.org/question/265534/using-teb_local_planner-with-robot_localization/

4.2.1.2 Global Planner for the Scooter

For our setup at the time of writing, we do not use any pre-recorded map or database of the environments we run the scooter in. So the global costmap is really just an empty map with everything marked as free space. This makes the global planner effectively useless, as it just plans a straight line path to any given goal. Still, to maintain move_base's data flow, we configure `move_base` to use `navfn`[16] as the global planner, which uses Dijkstra's algorithm as the navigation function.

4.2.1.3 Local Planner for the Scooter

The local planner plans the fine movements for the scooter, and should consider its kinematic constraints. We do not want the local planner to plan paths that are impossible for the scooter to achieve, for example, to move sideways or turn on the spot. To achieve this, we use the Time Elastic Band (TEB) local planner[17] [38]. TEB allows us to configure the robot's kinematic limits, including parameters like wheelbase, minimum turning radius, and the robot's footprint. We also set the paramter `cmd_angle_instead_rotvel` to true, which tells the planner to directly output the required steering angle, instead of the default `move_base` configuration that outputs angular velocities.

4.2.2 Preparing Waypoints

At this time, waypoints for any route are manually generated using Google Maps. Right clicking at any point in Google Maps allows us to copy its coordinates. We do this for several

[16] http://wiki.ros.org/navfn
[17] http://wiki.ros.org/teb_local_planner

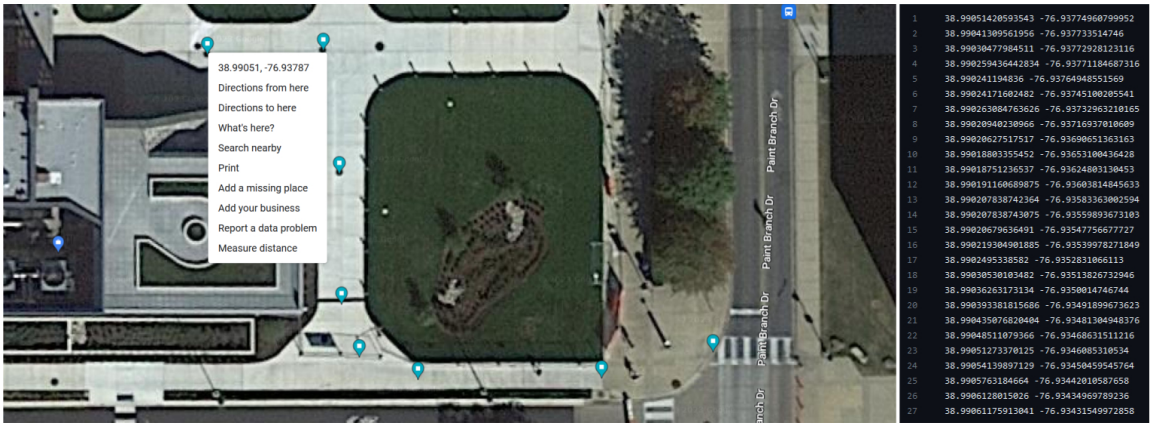

Figure 4.9: Screenshots showing the manual generation of waypoints from Google Maps into a text file which is later read by the autonomy stack. I also save the waypoints as labels (blue pins) for reference.

points along the desired route and store it in a text file as seen in Figure 4.9.

4.2.3 Publishing Waypoints

Waypoints are published using the `gps_waypoint_node` from the `outdoor_waypoint_nav` package[18], which reads the previously prepared list of waypoints, converts each into the `odom` frame, and sends it to `move_base`. The node also monitors the distance between the scooter and the goal, and sends the next waypoint as soon as the scooter is within 2 meters of the goal. This is done because `move_base` performs corrective maneuvers to reach within a given goal's tolerance, and takes a few seconds to verify that the goal is reached[19]. We do not want these pauses for each waypoint, and switching to the next waypoint avoids this behavior. A flow diagram shown in Figure 4.10 illustrates this method.

[18]We use a modified version of Nick Charron's package available here: https://github.com/nickcharron/waypoint_nav/tree/master/outdoor_waypoint_nav

[19]This behaviour can be seen in one of our test videos here: https://www.youtube.com/watch?v=0MFSLxyukCQ

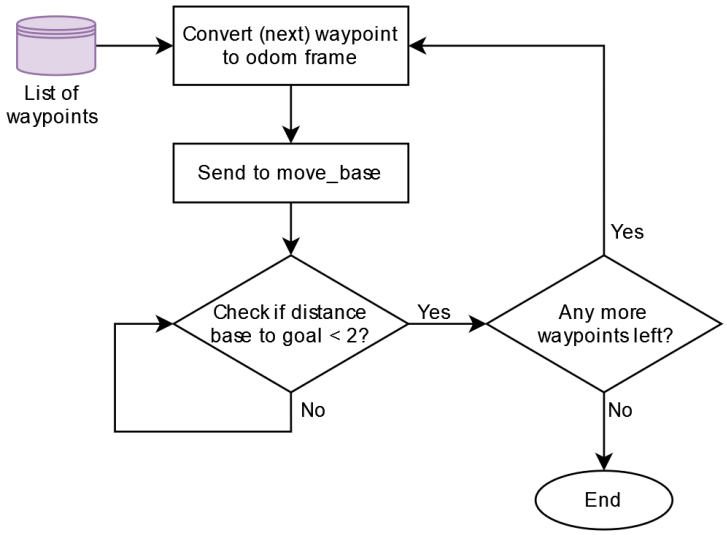

Figure 4.10: A flow diagram of the outdoor waypoint navigation node, reading and publishing the generated list of waypoints.

4.2.4 An Overview of the Autonomous Navigation Process

An overall view of the navigation stack is shown in Figure 4.11. Limited by time, I have not tuned all modules in this process to the best that they could be, and I believe we could do better with more time invested into careful configuration and isolated testing. The modules themselves with certainly be improved and changed as ReZoom evolves.

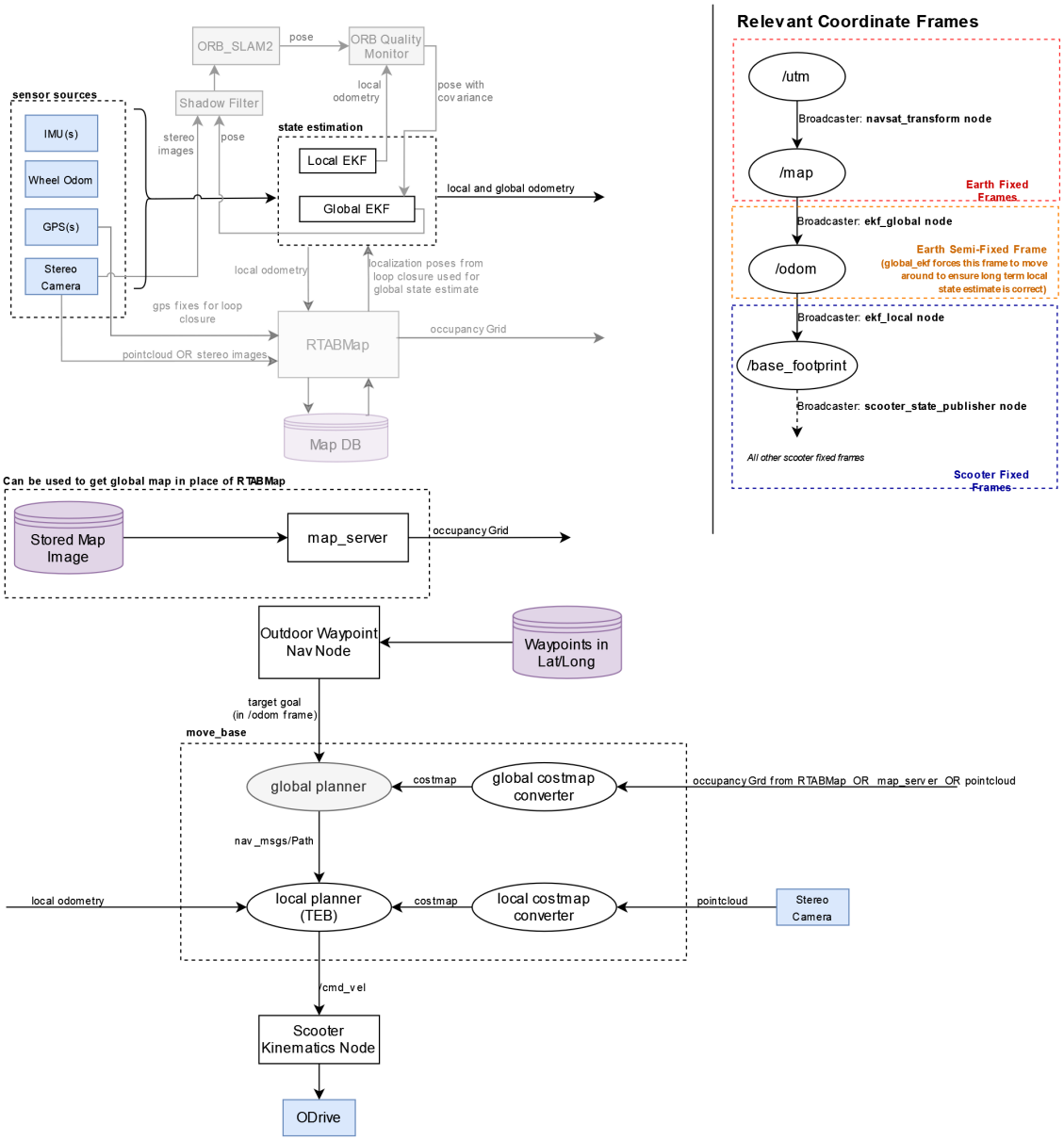

Figure 4.11: An overview of the overall autonomy data flow for the scooter's localization and navigation, along with relevant coordinate frames. Grayed out nodes for ORB_SLAM and RTABMap are not presently in use.

Chapter 5: Autonomy Experimental Results

This chapter presents localization results, local planning results and finally combines them to demonstrate autonomous waypoint navigation.

5.1 Evaluating Localization Performance

Before moving on to autonomously navigating the scooter, we must verify that the localization performance is satisfactory. Any errors in localization directly affect the self-driving performance, as navigation heavily relies on the coordinate transformations generated by the localization module.

5.1.1 Methodology to Evaluate Localization

Testing the local EKF is fairly straightforward, and can be performed even indoors with sufficient space. We draw a known figure, such as an 8x8 meter square, and have the scooter go around the square and check the error upon loop closure (Similar to Figure 4.4). Multiple loops can be performed to test robustness.

But for the global EKF, it is important to know if the scooter is correctly geo-localized. To check this, we have created a plotting script using `matplotlib` in Python3 to overlay results of the scooter's EKF performance over a screenshot of Google Maps. The script also allows us

Figure 5.1: Selecting ground truth points which are easily identifiable on Google Maps, allowing us to get their LL.

to plot ground truth points, and plot the error between the EKF output and ground truth. Ground truth points are visually identifiable features on Google Maps, such as manhole covers shown in Figure 5.1, which allows us to get their LL coordinates.

We then record a `rosbag`[1] collecting the output of the global EKF while pushing the scooter over a pre-determined path, manually. While collecting the `rosbag`, we hold the scooter over these ground truth points for a few seconds. This gives a time reference when plotting error—the scooter is at a ground truth point when its velocity is zero.

5.1.2 Localization Performance Results

The following four Figures 5.3 through 5.6 show the results of this test on the University of Maryland Campus. As shown in Figure 5.2, the scooter starts at a location facing a known

[1] http://wiki.ros.org/rosbag

Figure 5.2: The scooter is started facing an AprilTag placed at a location which is identifiable on Google Maps, and therefore at a known global pose.

AprilTag for each of these tests. This helps the scooter initialize its initial pose. The scooter is then pushed along, and the plots include output of the global EKF (yellow) and its covariance (black), along with raw GPS sensor inputs (green), ground truth points (blue +'s), and measured distance error at each ground truth point (marked in red).

These results are after many hours of testing different sensor input combinations and tuning internal EKF covariances. After multiple tests over the course of months, we have observed that the global EKF relies heavily on good GPS measurements, regardless of tuning the

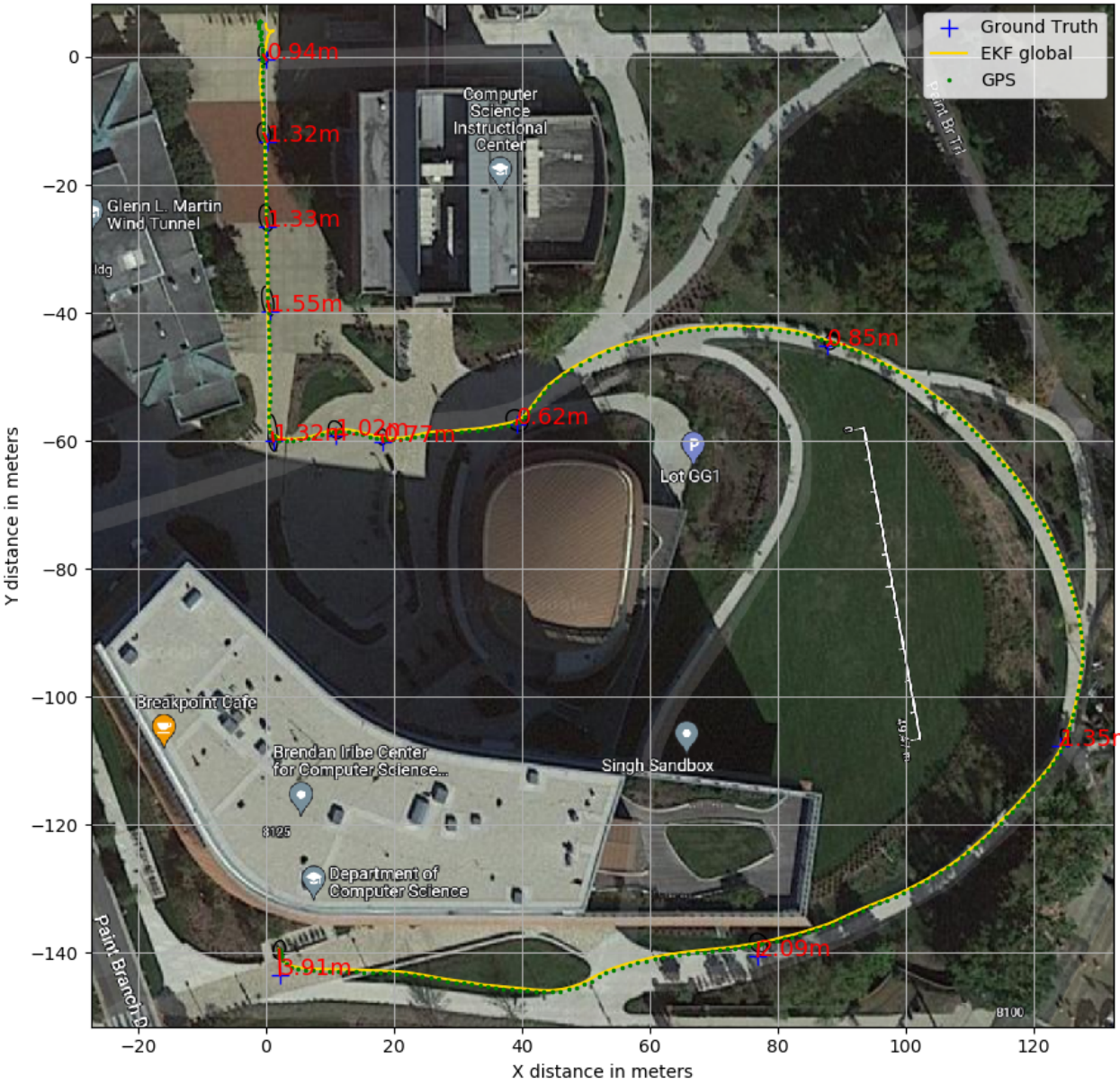

Figure 5.3: [CSI to Iribe] The plot shows results of pushing the scooter manually along a known path to evaluate EKF performance - Path length 375 meters. Black ellipses represent the covariance ellipse of the global EKF at the ground truth point, and the text in red is the distance error.

Figure 5.4: [IDF to CSI] The plot shows results of pushing the scooter manually along a known path to evaluate EKF performance - Path length 620 meters. Black ellipses represent the covariance ellipse of the global EKF at the ground truth point, and the text in red is the distance error.

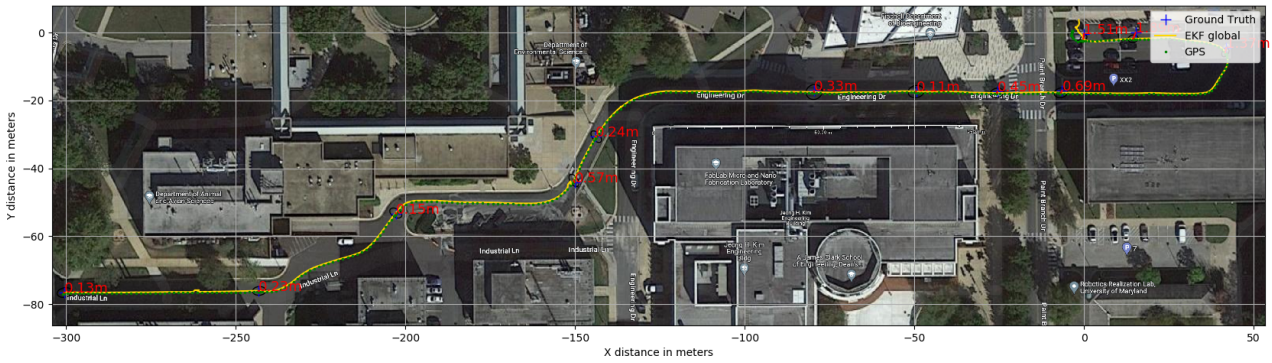

Figure 5.5: [ERB to Regents Dr.] The plot shows results of pushing the scooter manually along a known path to evaluate EKF performance - Path length 440 meters. Black ellipses represent the covariance ellipse of the global EKF at the ground truth point, and the text in red is the distance error.

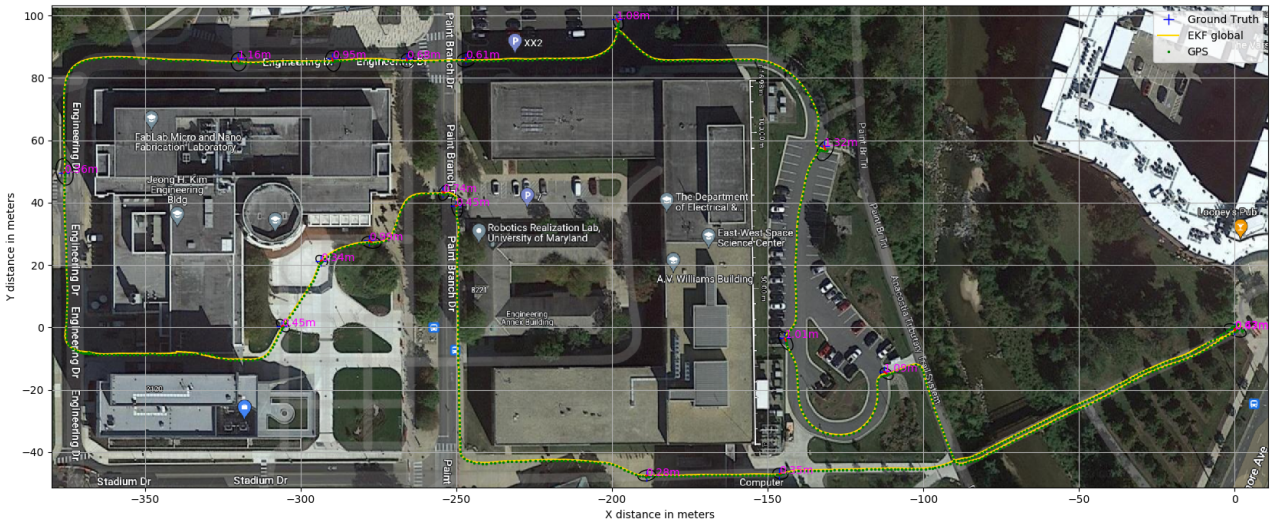

Figure 5.6: [Looneys Loop] The plot shows results of pushing the scooter manually along a known path to evaluate EKF performance - Path length 1150 meters. Black ellipses represent the covariance ellipse of the global EKF at the ground truth point, and the text in magenta is the distance error.

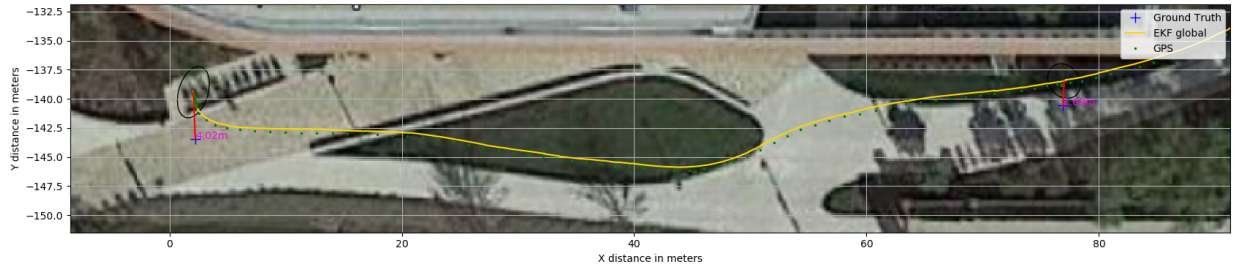

Figure 5.7: A zoomed in view of Figure 5.3 is shown. The close proximity to a large building introduces error in GPS measurements, causing the global EKF to be off by about 4 meters at the end.

`process_noise_covariance` and `initial_estimate_covariance`. If GPS reports incorrect values, global EKF slowly converges to these erroneous values. This can also observed when the scooter is closer to a large building which seemingly causes an error in GPS readings (Zoomed view shown in Figure 5.7).

The results of all four tests are then plotted in Figure 5.8. The combined tests show a mean of 0.9683 meters and a standard deviation of 0.6707 meters for distance error. It has been

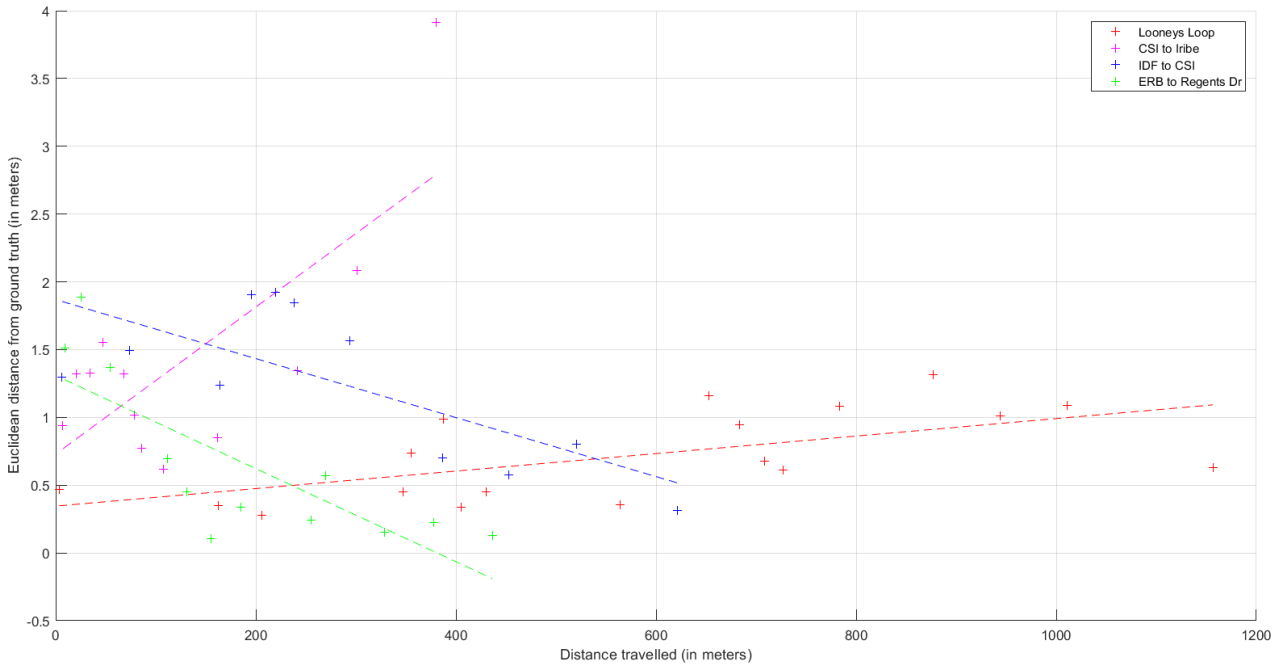

Figure 5.8: Distance error from ground truth vs distance travelled by the scooter in meters. Linear regression is performed for each run to show the trend in distance error.

observed that the error remains bounded unless a significant error in GPS fixes are reported.

These results are still fairly good considering that no visual odometry or perception algorithm is included, and the EKF relies only on wheel encoders, IMUs and GPS.

5.2 Local Planning Results

As discussed in 4.2.1.2, the global planner for our setup is simple, and plans a straight line path to the goal in most cases. However, the local planner (TEB) must account for the robot's kinematics and it's therefore interesting to see the kind of path it plans. Videos demonstrating the results of local planning and obstacle avoidance can be seen here[2,3].

The TEB local planner is able to plan smooth trajectories, and generates commands re-

[2]Local Planning Results: https://youtu.be/O717ezibP8A
[3]Local Planner Avoiding Obstacles: https://youtu.be/razoHR4mLqM

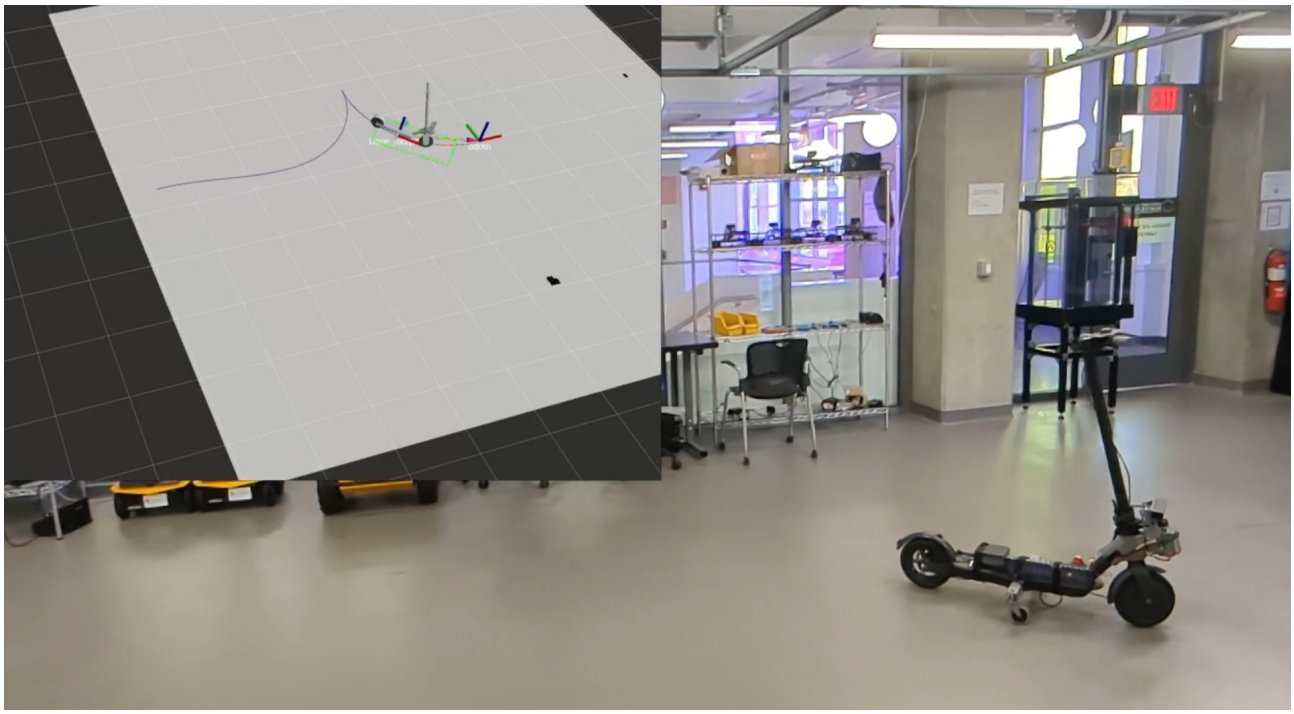

Figure 5.9: An example of TEB local planner's planned path (in blue), and the scooter following it (red curve shows local EKF output). See full video.

quired for the drive wheel velocity and steering angle. It also gracefully avoids any obstacles along the path, although the limits for avoiding dynamic obstacles has not been evaluated.

5.3 Autonomous Waypoint Navigation

5.3.1 Methodology for Autonomous Waypoint Navigation

After localization, we now enable `move_base` and send it pre-collected waypoints using the `outdoor_waypoint_nav` package. In this case, it is difficult to get ground truth for the path that the scooter takes. We can only qualitatively evaluate how well the scooter follows a route by plotting the shortest distance from each waypoint to the gobal EKF curve. But since the waypoint publisher switches to the next waypoint as soon as the scooter is close to the current waypoint, we cannot expect this distance to be zero.

Three successful missions were performed—two along the CSI to Iribe route and one along the ERB to Regents Dr. route. A video of the first mission is shown.

5.3.2 Results of Autonomous Waypoint Navigation

For the first mission, the scooter starts off at the same location as shown in Figure 5.2, and then given waypoints along the path as shown in Figure 5.3. The result of this autonomous mission can be seen here[4]. Figures 5.10, 5.11 and 5.12 show plots for all three successful missions.

However, obstacle avoidance is performed solely using depth information from the ZED2i with a height threshold. So the scooter cannot distinguish between surfaces of similar height, and would happily drive into grass if it were cut short as seen in Figure 5.14. This form of "blind" localization is not capable of driving the scooter reliably along sidewalks or narrow trails, and more sophisticated methods need to be investigated.

The AprilTag placed at the destination parking lot is intended to be used for (near) future development of precise parking maneuvers using the tag as reference.

[4]Result of the scooter's autonomous mission from CSI to Iribe: https://youtu.be/WBt8li1MTVY

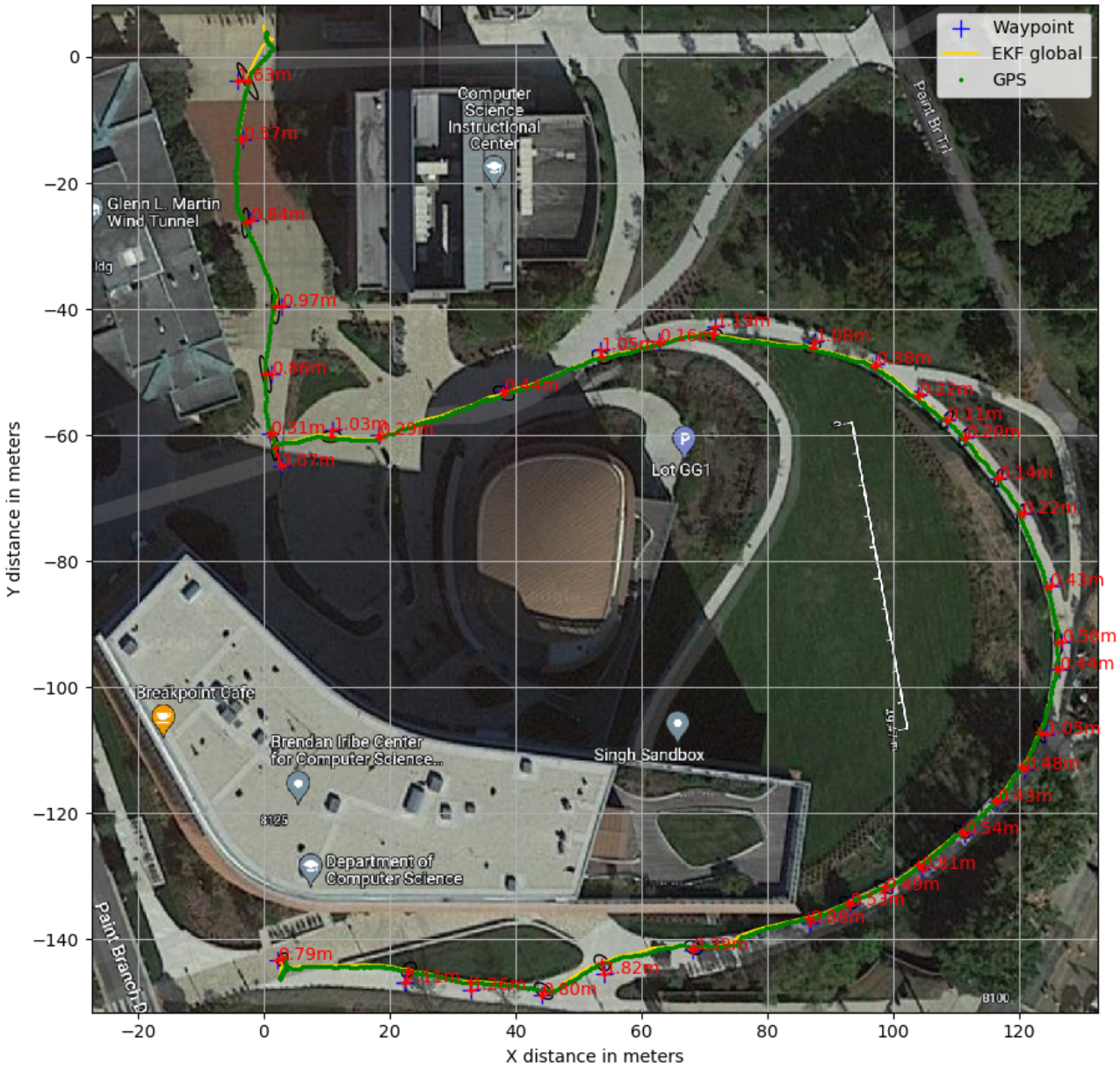

Figure 5.10: [CSI to Iribe 1] The plot shows results of the scooter moving autonomously for the given waypoints - Path length 375 meters. The text in red is the shortest distance to each waypoint, and the black ellipses represent the covariance ellipse of the global EKF at that point.

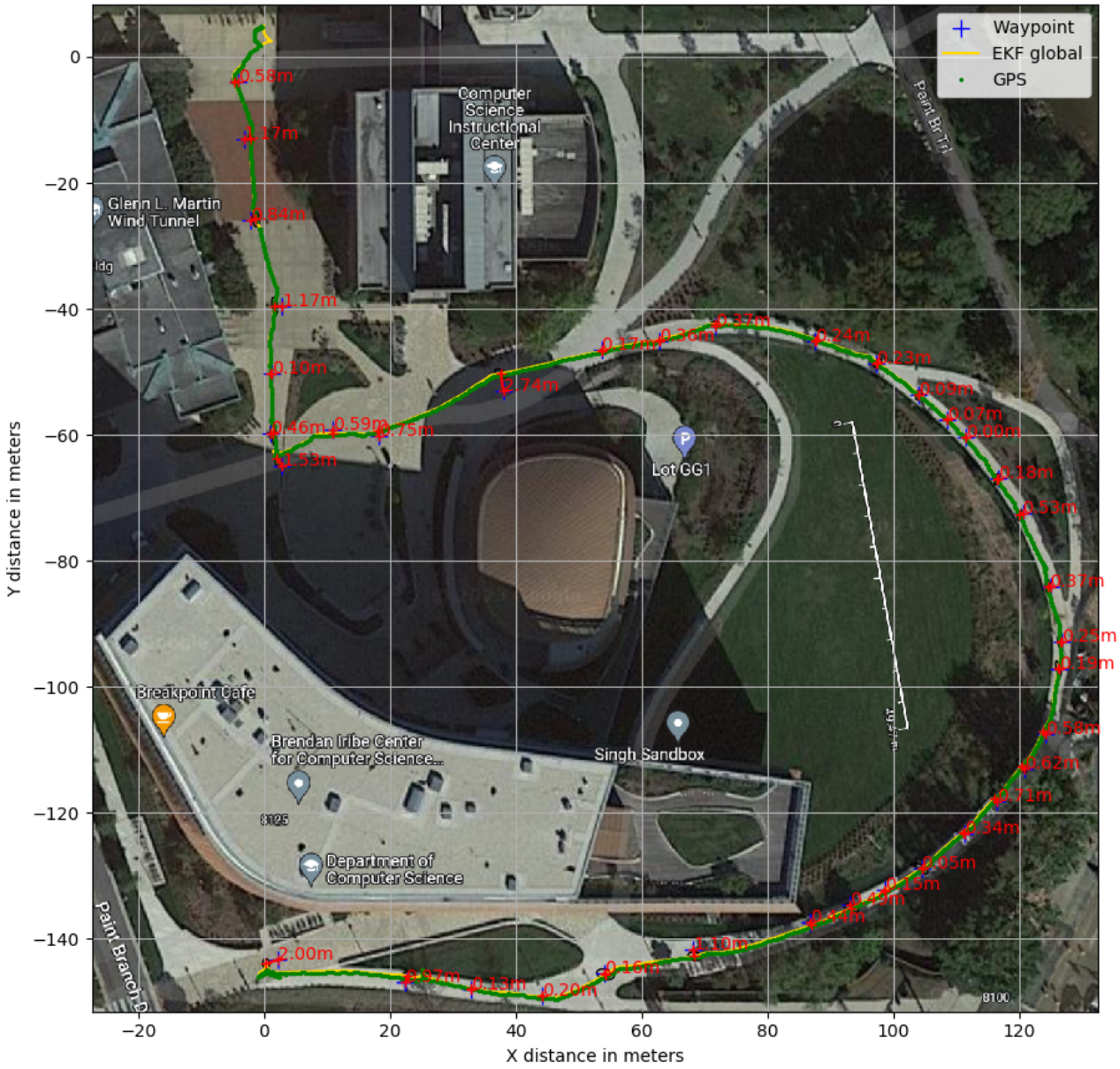

Figure 5.11: [CSI to Iribe 2] The plot shows results of the scooter moving autonomously for the given waypoints - Path length 375 meters. The text in red is the shortest distance to each waypoint, and the black ellipses represent the covariance ellipse of the global EKF at that point.

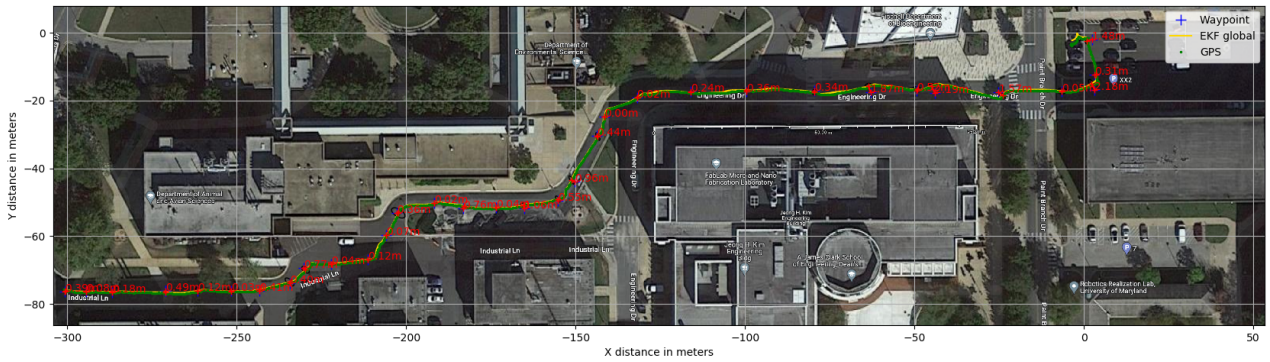

Figure 5.12: [ERB to Regents Dr.] The plot shows results of the scooter moving autonomously for the given waypoints - Path length 375 meters. The text in red is the shortest distance to each waypoint, and the black ellipses represent the covariance ellipse of the global EKF at that point.

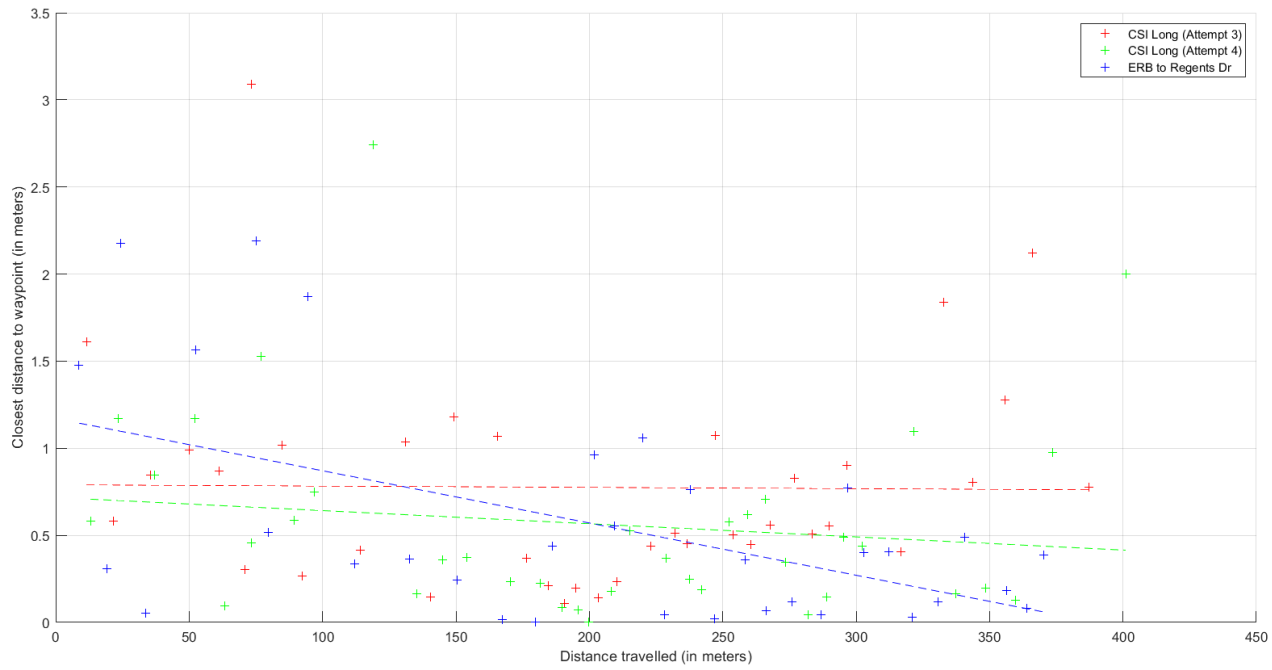

Figure 5.13: Shortest distance from waypoint vs distance travelled by the scooter in meters for all three autonomous missions. Mean distance 0.6353, standard deviation 0.6093 meters. Linear regression is performed for each run to show the trend in distance error.

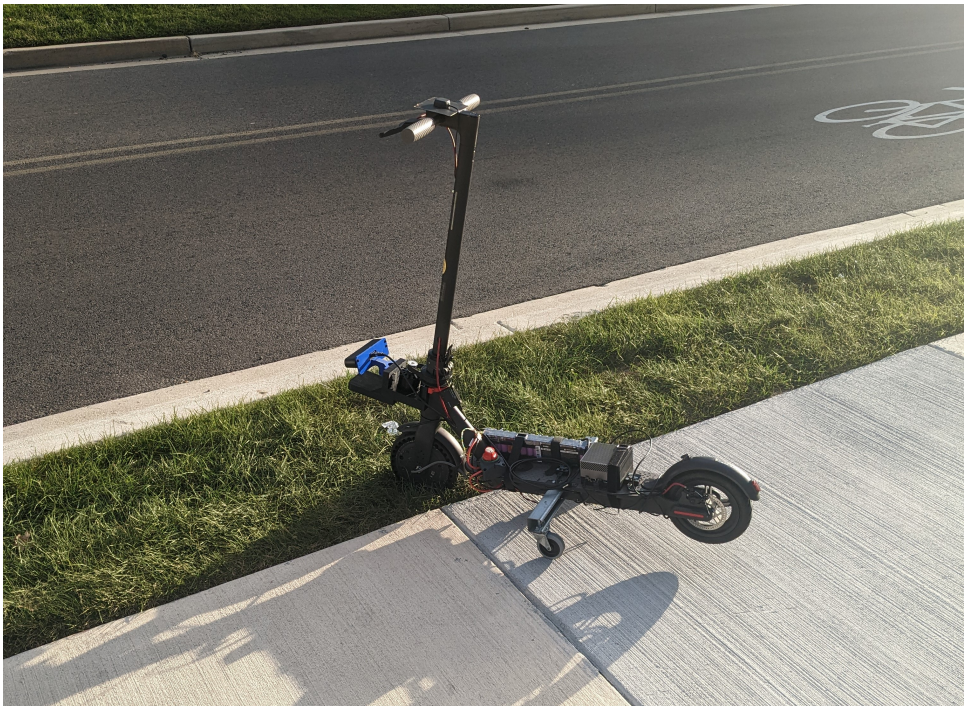

Figure 5.14: The scooter drives into grass and get stuck during an autonomous mission.

Chapter 6: Conclusion

6.1 Summary of Contributions

This thesis presents the development process and design decisions taken to develop an autonomous electric scooter, starting from OEM scooter hardware. Three such scooters were developed, each building upon its ancestor. Alongside hardware, the software development and integration also followed an incremental and modular approach, with each functionality separately tested and debugged. The localization and navigation framework for the scooter is presented, along with results demonstrating its success. However, without the use of visual guidance, localization outputs poor estimates when GPS is erroneous. The effect of this is seen during autonomous navigation, and the scooter requires a sufficiently wide path for travel. Further, it cannot distinguish surfaces like sidewalk and grass when they are nearly the same height. Using ROS helped in building a modular software framework, and nodes such as `init_from_tag` could certainly be used in other autonomous robots. Moreover, this thesis is intended as a development guide for a future developer at ReZoom or similar research.

6.2 Future Work

On the way to achieving complete autonomy, there are a lot of challenges left to investigate. Some of the most relevant topics for immediate research are:

- Modifications to the `init_from_tag` node should allow for the scooter to precisely park itself at a parking lot. This will complete the final piece of the autonomous mission structure: initialize–travel–park.

- Presently, waypoints are manually generated, but our work on generation of waypoints using historical data[39] would allow us to generate them automatically. Though the Python code has been developed, a ROS wrapper needs to be implemented to allow integration into our autonomy stack.

- Some form of image segmentation needs to be included to strictly avoid terrain such as grass. This could also help the scooter stay better localized, for example, by helping it stay at the center of the sidewalk.

Bibliography

[1] O. EPA, *What if we kept our cars parked for trips less than one mile?* en, Collections and Lists, 2016. [Online]. Available: https://www.epa.gov/greenvehicles/what-if-we-kept-our-cars-parked-trips-less-one-mile.

[2] T. Nouvian, "In paris referendum, 89% of voters back a ban on electric scooters," en-US, *The New York Times*, 2023, ISSN: 0362-4331. [Online]. Available: https://www.nytimes.com/2023/04/03/world/europe/paris-electric-scooters-ban.html.

[3] P. Wenzelburger and F. Allgower, "A first step towards an autonomously driving e-scooter," en, p. 4.

[4] R. Soloperto, P. Wenzelburger, D. Meister, D. Scheuble, V. S. M. Breidohr, and F. Allgöwer, "A control framework for autonomous e-scooters," en, in *IFAC-PapersOnLine*, ser. 16th IFAC Symposium on Control in Transportation Systems CTS 2021, vol. 54, 2021, 252–258. DOI: 10.1016/j.ifacol.2021.06.030. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2405896321004675.

[5] en, 2020. [Online]. Available: https://sfist.com/2020/01/08/city-aims-to-shut-down-rogue-scooter-company/.

[6] K. Hyatt, *Go x is unleashing 100 self-driving scooters on a large georgia business park*, en, 2020. [Online]. Available: https://www.cnet.com/roadshow/news/go-x-peachtree-corners-self-driving-scooter-test-fleet/.

[7] E. Gorgan, *Go x, tortoise launch world's only fleet of self-driving scooters*, en, 2020. [Online]. Available: https://www.autoevolution.com/news/go-x-tortoise-launch-worlds-only-fleet-of-self-driving-scooters-143924.html.

[8] *Introduction to ros*. [Online]. Available: http://wiki.ros.org/ROS/Introduction.

[9] *Standard units of measure and coordinate conventions*. [Online]. Available: https://www.ros.org/reps/rep-0103.html.

[10] *Coordinate frames for mobile platforms*. [Online]. Available: https://ros.org/reps/rep-0105.html.

[11] *What do the different north arrows on a usgs topographic map mean? — u.s. geological survey*. [Online]. Available: https://www.usgs.gov/faqs/what-do-different-north-arrows-usgs-topographic-map-mean.

[12] en. [Online]. Available: https://docs.revrobotics.com/duo-build/mecanum-drivetrain-kit-mecanum-drivetrain/mecanum-wheel-setup-and-behavior.

[13] [Online]. Available: https://groups.csail.mit.edu/drl/courses/cs54-2001s/diffdrive.html.

[14] P. Polack, F. Altché, B. d'Andréa Novel, and A. de La Fortelle, "The kinematic bicycle model: A consistent model for planning feasible trajectories for autonomous vehicles?" In *2017 IEEE Intelligent Vehicles Symposium (IV)*, 2017, 812–818. DOI: 10.1109/IVS.2017.7995816.

[15] *The kinematic bicycle model*. [Online]. Available: https://thomasfermi.github.io/Algorithms-for-Automated-Driving/Control/BicycleModel.html.

[16] S. Crowe, *Intel issues end-of-life notice for realsense lidar*, en-US, 2021. [Online]. Available: https://www.therobotreport.com/intel-issues-end-of-life-notice-realsense-lidar/.

[17] J. Seo, H. Lee, G.-I. Jee, and C. Park, "Lever arm compensation for gps/ins/odometer integrated system," *International Journal of Control, Automation and Systems*, vol. 4, 2006.

[18] Y. H. Cho, W. J. Park, and C. G. Park, "Novel methods of mitigating lever arm effect in redundant imu," *IEEE Sensors Journal*, vol. 21, no. 7, 9465–9474, 2021, ISSN: 1558-1748. DOI: 10.1109/JSEN.2021.3054945.

[19] T. Kos, I. Markezic, and J. Pokrajcic, "Effects of multipath reception on gps positioning performance," in *Proceedings ELMAR-2010*, 2010, 399–402.

[20] J. Leonard and H. Durrant-Whyte, "Mobile robot localization by tracking geometric beacons," *IEEE Transactions on Robotics and Automation*, vol. 7, no. 3, 376–382, 1991, ISSN: 2374-958X. DOI: 10.1109/70.88147.

[21] R. E. Kalman, "A new approach to linear filtering and prediction problems," en, *Journal of Basic Engineering*, vol. 82, no. 1, 35–45, 1960, ISSN: 0021-9223. DOI: 10.1115/1.3662552.

[22] E. Hendricks, O. Jannerup, and P. H. Sørensen, "Optimal observers: Kalman filters," en, in *Linear Systems Control: Deterministic and Stochastic Methods*, E. Hendricks, O. Jannerup, and P. H. Sørensen, Eds. Berlin, Heidelberg: Springer, 2008, 431–491, ISBN: 978-3-540-78486-9. DOI: 10.1007/978-3-540-78486-9_7. [Online]. Available: https://doi.org/10.1007/978-3-540-78486-9_7.

[23] R. G. Brown and P. Y. C. Hwang, *Introduction to Random Signals and Applied Kalman Filtering*, en. Wiley, 1992, Google-Books-ID: 6f5SAAAAMAAJ, ISBN: 978-0-471-52573-8.

[24] Mathworks, *Understanding kalman filters*, en. [Online]. Available: https://www.mathworks.com/videos/series/understanding-kalman-filters.html.

[25] E. Wan and R. Van Der Merwe, "The unscented kalman filter for nonlinear estimation," in *Proceedings of the IEEE 2000 Adaptive Systems for Signal Processing, Communications, and Control Symposium (Cat. No.00EX373)*, 2000, 153–158. DOI: 10.1109/ASSPCC.2000.882463.

[26] F. Dellaert, D. Fox, W. Burgard, and S. Thrun, "Monte carlo localization for mobile robots," in *Proceedings 1999 IEEE International Conference on Robotics and Automation (Cat. No.99CH36288C)*, vol. 2, 1999, 1322–1328 vol.2. DOI: 10.1109/ROBOT.1999.772544.

[27] T. Moore and D. Stouch, "A generalized extended kalman filter implementation for the robot operating system," en, in *Intelligent Autonomous Systems 13*, E. Menegatti, N. Michael, K. Berns, and H. Yamaguchi, Eds., ser. Advances in Intelligent Systems and Computing, vol. 302, Cham: Springer International Publishing, 2016, 335–348, ISBN: 978-3-319-08337-7. DOI: 10.1007/978-3-319-08338-4_25. [Online]. Available: https://link.springer.com/10.1007/978-3-319-08338-4_25.

[28] methylDragon, *Sensor fusion in ros*, 2023. [Online]. Available: https://github.com/methylDragon/ros-sensor-fusion-tutorial.

[29] Lorenz, *The ros robot localization package*, 2019. [Online]. Available: https://kapernikov.com/the-ros-robot_localization-package/.

[30] N. Seegmiller and D. Wettergreen, "Optical flow odometry with robustness to self-shadowing," in *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, IEEE, 2011, pp. 613–618.

[31] Mathworks, *Magnetometer calibration - matlab simulink*. [Online]. Available: https://www.mathworks.com/help/fusion/ug/magnetometer-calibration.html.

[32] W. Koo, S. Sung, and Y. J. Lee, "Error calibration of magnetometer using nonlinear integrated filter model with inertial sensors," *IEEE Transactions on Magnetics*, vol. 45, no. 6, 2740–2743, 2009, ISSN: 1941-0069. DOI: 10.1109/TMAG.2009.2020542.

[33] D Gebre-Egziabher, G. H. Elkaim, J. D. Powell, and B. W. Parkinson, "A non-linear, two-step estimation algorithm for calibrating solid-state strapdown magnetometers," en,

[34] A. R. Spielvogel and L. L. Whitcomb, "A stable adaptive observer for hard-iron and soft-iron bias calibration and compensation for two-axis magnetometers: Theory and experimental evaluation," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, 1295–1302, 2020, ISSN: 2377-3766. DOI: 10.1109/LRA.2020.2967308.

[35] E. Olson, "Apriltag: A robust and flexible visual fiducial system," in *2011 IEEE International Conference on Robotics and Automation*, 2011, 3400–3407. DOI: 10.1109/ICRA.2011.5979561.

[36] J. Wang and E. Olson, "Apriltag 2: Efficient and robust fiducial detection," in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2016, 4193–4198. DOI: 10.1109/IROS.2016.7759617.

[37] [Online]. Available: http://motion.cs.illinois.edu/RoboticSystems/CoordinateTransformations.html.

[38] C. Rösmann, F. Hoffmann, and T. Bertram, "Kinodynamic trajectory optimization and control for car-like robots," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2017, 5681–5686. DOI: 10.1109/IROS.2017.8206458.

[39] V. Agrawal, S. Poojari, and D. Paley, "Graph-based global path planning for an autonomous electric scooter using historical ride data," in *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2023, [Submitted].