March 2019

Autonomous RC Car Platform

Jason Louis Ashton
Worcester Polytechnic Institute

Myles Emmolo Spencer
Worcester Polytechnic Institute

Sean R. Hunt
Worcester Polytechnic Institute

WORCESTER POLYTECHNIC INSTITUTE

MAJOR QUALIFYING PROJECT

Autonomous RC Car Platform

Authors:
Jason ASHTON
Sean HUNT
Myles SPENCER

Advisors:
Prof. Jie FU
Prof. Michael GENNERT

A paper submitted in fulfillment of the
Major Qualifying Project

Control and Intelligent Robotics Lab

March 19, 2019

WORCESTER POLYTECHNIC INSTITUTE

Abstract

Control and Intelligent Robotics Lab

Major Qualifying Project

Autonomous RC Car Platform

by Jason ASHTON, Sean HUNT, Myles SPENCER

This project explores building an autonomous research robot on a 1/10 scale RC car platform. The goals of the project were to build an easy to use system that allowed for the exploration of techniques such as localization, object detection, mapping, and more. The completed robot consists of a self-contained RC car, running on battery power, that uses a camera, lidar, inertial measurement unit, and other sensors to observe the environment. Completed research explored pose estimation based on combining dead reckoning, inertial measurement unit readings, and visual odometry in an Extended Kalman Filter. The result of this project included the RC car and a build guide on replicating the process for future students.

Acknowledgements

We'd like to thank Professor Jie Fu, Professor Michael Gennert, and Abhishek Kulkarni for their continued support and guidance throughout the development of this project.

Contents

List of Figures

List of Tables

List of Abbreviations

CPU	**C**entral **P**rocessing **U**nit
ESC	**E**lectronic **S**peed **C**ontroller
IMU	**I**nertial **M**easurement **U**nit
MQP	**M**ajor **Q**ualifying **P**roject
ORB	**O**riented **F**AST **R**otated **B**RIEF
RC	**R**adio **C**ontrolled
ROS	**R**obotic **O**perating **S**ystem
SIFT	**S**cale **I**nvariant **F**eature **T**ransform
SLAM	**S**imultaneous **L**ocalization **A**nd **M**apping
SURF	**S**peeded **U**p **R**obust **F**eature
WPI	**W**orcester **P**olytechnic **I**nstitute

1 Introduction

1.1 Introduction

Sitting outside in Mountain View, California by a public road, there's a good chance that you may see multiple cars drive by with a wide array of sensors attached to the vehicle. Companies like Waymo, Lyft, Cruise, and Tesla, and universities including Stanford and Carnegie Mellon are all researching self-driving vehicles for use on public roads. The reasons for doing so include eliminating traffic accidents and human harm, economic incentives of lower operating costs, and increased time for leisure or work otherwise spent driving.

1.1.1 Project Statement

The goal of this project is to explore localization and vision techniques, and set the base work for an autonomous car research platform for future students to study upon. The result of this will be the second set of research on this platform at Worcester Polytechnic Institute (WPI), and increased ground work completed for future students.

1.1.2 Summary

With major companies and other institutions interested in this field of research, WPI should also provide this opportunity to students. To reduce the costs and complexity of operating a full scale motor vehicle, we're researching using 1/10th scale radio-controlled cars typically used for hobby racing. This project will become the platform for our research and for future students.

2 Background

2.1 State of the Industry

Being able to travel without a dedicated pilot or driver allows both freedom to travel where you wish and the freedom to spend the time travelling how you would like. This wish has been worked on for thousands of years, with recent attempts becoming ever more sophisticated.

2.1.1 History of Self-Driving Vehicles

While new self-driving car efforts are making headlines, the history of automating the piloting of vehicles is nearly a hundred years old. The first self-propelled vehicles were likely weather vanes hooked into the tiller of a sailboat to keep the sailboat on a steady directions even in heavy winds [1]. A major advancement later came with the first auto-pilot systems on airplanes, which would allow an operator to set a desired heading and altitude. In the 1930's, Wiley Post became the first person to fly solo around the world, with the help of his Sperry Gyroscope autopilot. This autopilot allowed him to both fly and navigate the aircraft at the same time, a job typically done by two pilots.

In the 1930's a wave of interest in self-driving cars excited the public. General Motors displayed plans for an automated highway in its 1939 Futurama ride, which took visitors on a moving seat around a diorama of the future of cities. At the 1939 World's Fair Norman Bel Geddes, an American industrial designer, displayed a highway concept that included trench-like lanes to keep the cars on a certain track. The idea was that the driver would enter on the highway and hook into these tracks, until they exited the highway and took back over themselves.

The rise of computers in the 1960's allowed for the first guidance systems, such as early missile guidance systems. At the same time artificial intelligence

was beginning to boom and show promise for more advance computing algorithms. AI enthusiasts began looking into making cars that could navigate streets on their own. A major milestone came in the 1980's when German pioneer Ernst Dickmanns was able to build a Mercedes van to drive hundreds of highway miles autonomously. The car was capable of a convoy mode and lane changing in addition to being able to drive in a highway lane. This project was part of the European Eureka Prometheus project, the largest R&D project ever in the field of driverless cars. [2].

2.1.2 Current Efforts

The latest self-driving boom started in 2004 when DARPA challenged dozens of teams that were already researching autonomous vehicles to participate in a challenge they would host [1]. This challenge involved driving autonomously through the desert over a 142 mile course in Nevada [3]. The goal of the challenge was to promote the development of technologies that would allow military vehicles to operate autonomously, such as supply convoys. While no team won the 2004 competition, DARPA held another in 2005, with the winning team coming from Stanford. The car was nicknamed "Stanley", and finished first with a time of 6 hours and 53 minutes, winning the $2 million prize. The following year DARPA held the Urban Challenge, staged in a fake city that involved moving traffic, obstacles, and traffic regulations. Carnegie Mellon University won the Urban Challenge and the $2 million prize.

As of May 2018, twenty-two states and the District of Columbia have passed laws relating to autonomous vehicles, while an additional ten state governors have issued executive orders regarding the operation of autonomous vehicles [4]. According to Navigant Research, the top ten companies developing autonomous vehicles, out of nineteen evaluated, are [5]:

- GM
- Waymo
- Daimler-Bosch
- Ford
- Volkswagen Group

- BMW-Intel-FCA
- Aptiv
- Renault-Nissan Alliance
- Volvo-Autoliv-Ericsson-Zenuity
- PSA

These companies were rated on ten criteria that focused on quality, staying power, and future vision.

Waymo, the self-driving car project of Google, has driven over eight million self-driving miles as of September 2018 [6]. Waymo was originally led by Sebastian Thrun, former team lead of the Stanford Stanley project that won the DARPA Grand Challenge. Founded in 2009 at Google X, Waymo has moved from a Toyota Prius platform, to Lexus SUVs, to a purpose built self-driving vehicle in 2014, to the current Chrysler Pacifica minivans. Morgan Stanley recently reported that Waymo could be worth $175 billion, $100 billion more than previously thought by analysts.

2.1.3 F1/10th and Previous MQP

F1/10th is a project founded by University of Pennsylvania to provide a platform and competition for self-driving car research in academia [7]. The competition involves designing, building, and testing a self-driving 1/10th scale RC car, which is capable of speeds in excess of 40mph. The project includes lectures, reading materials, an online teaching kit, and focuses on perception, planning, and control. The system utilizes a Traxxas rally racer, an Inertial Measurement Unit, a lidar, and cameras.

The previous MQP that this project will expand upon used the F1/10 platform as inspiration, with the eventual goal to have the Control and Intelligent Robotics Laboratory participate in the competition. This team focused on adaptive cruise control, trajectory generation, and a trajectory tracking controller [8].

2.2 Localization

Knowing where the robot is located and how it has moved is necessary to be able to map an area, and where the robot is and its path to get there. The process of determining where you are in a known map is referred to as localization. There are many different ways to achieve localization, depending on the sensors available and the capabilities and data available. Many localization algorithms popular for autonomous vehicles involve the use of cameras or lidar, along with an IMU. These sensors, when combined, allow the robot to sense the world around it and where it is going.

2.2.1 Lidar

Lidar sensors create a map of the world around them by scanning with a laser on a rotating platform. While quite expensive, they can have a high sample rate and resolution leading to a robust sensor even in varying environments. These features are quite beneficial, and because of this lidars are one of the most commonly used sensors for localization. The position of features are analyzed in each frame and can then be used to determine distance moved and the current heading. Algorithms such as the iterative closed point (ICP) algorithm can then be used to determine the position in the map. Plain ICP algorithms however do have some issues when sample sizes are large or the data becomes noisy. They can, however, be remedied by using a slightly smarter algorithm such as a feature-based weighted parallel iterative closed point (WP-ICP) algorithm. This algorithm increases speed and accuracy by reducing the number of points needed and dividing features up into either corners or lines to reduce the number of comparisons needed [9].

2.2.2 Camera

Cameras are also a very popular sensor for localization due to the fact that they are very inexpensive and can be used to solve a variety of different problems. Trajectory data such as distance moved and current heading are determined by tracking features between frames. However, due to the complexities of computer vision it is not always possible to use only a camera. Trajectory data is often combined with data from an IMU using a Kalman or other similar filter to

increase accuracy [10]. There are a number of different ways that the position in the map can be determined including feature extraction and analysis as well as image retrieval of images with known locations [11].

2.3 Mapping

One of the requirements to perform localization (as opposed to Simultaneous Localization and Mapping) is that there is a known map. Maps can take many different forms depending on the actual application, and can range from being as simple as a 2D map made of lines to a full 3D scan of an area. There are also a number of ways maps can be created depending on the required accuracy and intended use, ranging from scanning an area with a sensor such as a lidar or camera or manually creating the map in 3D software.

2.3.1 Object Detection

An object detection and recognition algorithm can enable the robot to perform localization and execute tasks based on recognized objects. Object and feature detection is the process of making a local decision at every image point to see if there is an image feature of the given type existing at that point [12]. The process requires some sensor to receive images from its surrounding and an algorithm to analyze the image in order to detect highly distinctive features with reliability. The two main features of the object detection that will be important to an autonomous system will be the speed of detection and the reliability in classifying objects correctly.

2.3.2 Algorithms

There are currently many algorithms that are being used for object detection since it is important for robotics and machine vision. Object detection algorithms rely on identifying key points of an image such as lines, corners, and distinct differences in regions of the images. Certain trained groups of key points are then bound in boxes and followed as multiple images are analyzed [13]. Some of the more popular object detection algorithms that can be implemented are Scale Invariant Feature Transform (SIFT), Speed up Robust Feature (SURF), and Oriented FAST Rotated BRIEF (ORB).

SIFT

First introduced in 1999 by Lowe, a researcher at University of British Columbia, the SIFT algorithm is an approach that transforms the image data into scale invariant coordinates relative to local features [14]. There are four main steps to the algorithm. The first detects a scale space extrema using the difference of Gaussian function to find potential points that are invariant to scale and orientation. The second step the algorithm refines the key points based on measures of their stability and determines their location and scale. Third, an orientation is assigned to each keypoint based on the local images gradient directions in order to perform all future operations relative to this local orientation. Then finally the local image gradients are measured around each keypoint to create a descriptor that represents the shapes distortions and change in illuminations. These descriptors are highly distinctive which allows for the probability to be matched to a feature from a database to be much greater.

SURF

Similar to SIFT, SURF is also a scale and rotation invariant feature detection algorithm that is sometimes seen as an approximation of SIFT. It performs faster without reducing he quality of the detected points by to much. SURF uses similar general steps for object detection, however the specifics of how each one is executed varies. The "approximation" comparison is made since SURF simply approximates the difference of Gaussians and by using integral images for image convolution which can be done in parallel for different scales [12]. In SURF the key points that are then used to build a descriptor are found through a BLOB detector, which is based on a Hessian matrix. The descriptor makes matches based on whether or not features have the same type of contrast compared to there backgrounds.

ORB

The Oriented FAST and Rotated BRIEF is a system built off of both of those features, FAST - a keypoint detector and BRIEF - a descriptor. It focuses on low cost and efficient computations with the same high reliability in object detection required for each algorithm. The ORB system takes the advantages and basis of the two underlying systems and builds on to them to fit the needs for a more

cohesive process. Unlike the previous two algorithms, the keypoint detector does not include an orientation operator on the belief that it is computationally demanding and a centroid operator, which is used, gives a single dominant result as opposed to multiple values on a single keypoint [15]. The BRIEF feature descriptor uses binary tests between pixels in images to train classification sets, that over time will be able to return signatures for any arbitrary keypoint throughout the image.

2.3.3 Image Transformations

Since our system will be attached to a mobile robot there will need to be consideration of how well the vision algorithm is able to detect changes between image polling. Some of these transformations could include scaling of an image, rotations of an image, and images with varying intensities. The scaling of an image will occur when the sensor is moving forward or backward through a frame, as it moves backwards the image that was previously analyzed will be in the next frame as a smaller scaled image and vice-versa for moving forward. Rotations of images will occur when the vehicle and camera sensor on-board is turning. And thirdly, varying intensities of images need to be considered since the environment the robot will be operating in is unknown, which could involve various levels of lighting which would either increase or decrease the intensity of the polled image respectively.

2.3.4 Semantic Object Detection

While humans can observe and give meaning to thousands of things they see everyday, robots are limited to the things they are trained to see which. In order to give the objects detected meaning robots need to be trained to know what objects and materials look like as well as possibly how to interact with them. In order to speed up this training, Semantic Hierarchies can be formed. These semantic hierarchies consist of objects and there discriminative details so groups of objects can be formed [16]. An example of a semantic hierarchy could be a car which can be broken down into motor vehicles, which in turn could be vehicles, then man-made object then physical object. This allows for context to be formed from the objects detected so the rest of the system can react to certain categories of objects instead of every individual object.

2.4 Semantic SLAM

While traditional SLAM techniques use geometric features, using semantic labels may allow for better loop-closure and pose optimization [17]. A paper presented to the 2017 IEEE international Conference on Robotics and Automation details a method for combining low-level geometric features and semantic labels to perform more efficient SLAM on both indoor and outdoor datasets. The goal of the paper was to address metric and semantic SLAM problems, using object recognition and semantically-labeled landmarks to address data association (matching sensor observations to map landmarks) and loop closure (recognizing previously-visited locations).

The sensor package used consisted of an inertial measurement unit and a single monocular camera. Geometric point measurements come from feature-detection algorithms such as SIFT, SURF, ORB, etc. while the semantic information comes from an object recognition model such as a DPM detector. The paper uses expectation-maximization to robustly handle the semantic data association. The result of the paper was evidence that semantic features can improve localization performance and loop closure, while only slightly affecting performance.

2.5 Applications of Semantic SLAM

Since Semantic SLAM is still an emerging field of technology, it is not actively integrated in a wide variety of areas, however the possibilities are immense. The benefits of having semantic detection on top of traditional SLAM is that it gives context to the world it is mapping and exploring. Instead of simply knowing where things are, it is possible through this technique to know what things are as well. Possible applications for Semantic SLAM include the commonly applied autonomous car, assistive and everyday household robots, as well military surveillance applications.

2.5.1 Cars

Most driverless car systems use some various forms of SLAM to localize itself on the road as well as mapping the surrounding area to learn more about how to interact with it. The specific benefits of the semantics allows for the car to interact differently with the objects it detects in its path or around it. As an

example, if there is an object in the road in the path of the car, the ability to detect whether it is a branch or a person will allow it to either continue on its path or evade it respectively.

2.5.2 Assistive Robotics

The use of semantics allows for robots navigating and mapping a room to interact with its surroundings better since it knows what the object in front of it actually is. This allows applications such as retrieving food or cups from areas and returning it to a person. It could allow cleaning robots to detect the surfaces it is on to clean accordingly. If it detects a mess on a carpet it will be able to adjust its cleaning settings to vacuum instead of mop for example.

3 Methodology

3.1 Objectives

To accommodate the unpredictable nature of long-term projects, we have split up our objectives into milestones, with each milestone increasing in complexity while building upon the previous.

3.1.1 Minimum Objectives

Description	**Target**
Working Car Platform	Ability to read all sensors and control onboard motors
ROS Environment	Working ROS environment w/ tools
Geometric Feature Detection	Implement SIFT, SURF, ORB, etc.
Localization	Ability to localize within a known map

TABLE 3.1: Minimum Objectives

3.1.2 Target Objectives

Description	**Target**
Object Detection	Ability to recognize our target objects
SLAM Implemented	Localization and mapping in an unknown map
Path Generation	Plan a path from object of interest to location of interest
Exploration	Ability to drive and explore autonomously

TABLE 3.2: Target Objectives

3.1.3 Advanced Objectives

Description	**Target**
Multiple Environments	Ability to perform objectives in more than one environment
Multiple Stages of Plans	Ability to interact with environment in complex ways

TABLE 3.3: Advanced Objectives

3.2 Requirements

The following sections describe the requirements that are set both externally and internally on the project. A functional requirement describes how a certain system is expected to perform while a non-functional requirement describes how the system is expected to be designed or work. Each requirement should be

testable and concise enough to describe a distinct function or aspect of the system.

3.2.1 Functional Requirements

Title	**Description**
Autonomy	Once activated, the system shall execute its tasks without human intervention
Data Logging	Data shall be logged in a way that allows for virtual playback and testing
Manual Controls	An operator shall be able to remotely navigate the vehicle
Data Visualization	While operating, the system shall visualize in a meaningful way what the robot is perceiving

TABLE 3.4: Functional Requirements

3.2.2 Non-Functional Requirements

Title	**Description**
Incremental Build Process	The system shall allow for its components to be built incrementally
Modularity	The software systems shall allow for testing of alternative components easily
Documentation	The results of the project shall be well documented to allow future students to further develop the system
Battery Life	The system shall be able to run for at least 30 minutes

TABLE 3.5: Non-Functional Requirements

3.3 System Timeline

The following sections provide an overview of the general schedule of this Major Qualifying Project, based on Worcester Polytechnic Institute's quarter-based calendar.

3.3.1 A Term

Timeframe	**Objective**
September 1	Understand Project
September 8	Statement of Work
September 15	Review Literature
September 22	Background
September 29	System Design; Sensor Mounting Plate, Wheel Encoders
October 6	Methodology, Draft submitted; Car Built
October 11	Final Proposal submitted

TABLE 3.6: A Term Schedule

3.3.2 B Term

Timeframe	**Objective**
October 27th	Set up ROS, install existing packages to establish basic functionality
November 3rd	Continue to refine system using existing packages
November 20th	Map a test environment, sample localization packages
December 1st	Sample SLAM algorithm packages
December 14th	Implement object detection

TABLE 3.7: B Term Schedule

3.3.3 C Term

Timeframe	**Objective**
January 19	Begin implementing Semantic SLAM
February 2	Refine Semantic SLAM
February 9	Results
February 23	Revise draft, misc. sections
March 1	Finish and Submit final paper

TABLE 3.8: C Term Schedule

4 System Design

The following section describes the considerations and approach of this project. The project begins with an analysis of stakeholders and needs, which influences the design and direction. Following that is an overview of the different physical and software components used.

4.1 Stakeholders and Needs Analysis

Table 4.1 lists the stakeholders of this project. Each stakeholder has their associated descriptions, involvement, and needs. A stakeholder was identified as any individual, group or entity that has an interest in the outcome of a system design project. Table structures and content are modeled based on the writings in Systems Engineering for Capstone Projects by Professor Fred Looft [18].

Title	**Description / Role**	**Involvement**	**Needs**
Students	Developers of Project	Directly Involved	System is feasible to test and develop on. Project is operates in a safe environment
Advisors	Advise and Grade Project	Directly Involved	Weakly updates on project progression. Project should meet Advisor's standards
WPI	Sponsoring Organization provides Space for Project	Graduation Requirement	Project should meet graduation requirements for an MQP Project
RBE/CS Departments	Provides Funding for Project	MQP Requirements	Project should meet requirements for an RBE and CS MQP Project
Future WPI Students	Continue the project and future autonomous car projects	Not Involved	Well documented code and project specifics. Modular structure to be built upon or removed from.

TABLE 4.1: Stakeholders

4.2 Robot Platform

To be able to navigate a variety of different terrains, the platform the robot is built on needed to be reliable, performant, and provide the space we needed to mount sensors and hardware. The platform chosen was a hobby radio controlled electric car from Traxxas, the Slash 4x4 Platinum Edition. This car features all of the electronics mounted low in the chassis with a large body, which left plenty of room for our own hardware. The primary interaction with the RC car will be through the electronic speed controller and the steering servo. The fundamental electronic components of the RC car are:

- Battery
- Electronic Speed Controller
- Steering Servo
- Motor

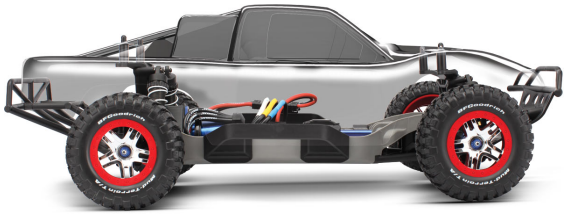


FIGURE 4.1: RC Car [19]

4.2.1 Features

The Slash 4x4 features a "performance-optimized low-CG [center of gravity] chassis" [19]. This places the battery and electronics as low as possible for higher performance. The rest of the space under the body is used for hardware attached to custom-built mounting plates. The large tires and high-travel suspension allow us to use the car in a variety of terrains, both indoor and outdoor. The body style also allows us to use a high power 18.4v 4S2P 8000mAh battery for long life.

4.3 On-board Computer

A computer is needed on-car to run ROS, communicate with sensors, provide GPU power, and more. This project is powered by an NVIDIA Jetson, a small and compact computer that provides some special features that make it a good fit for this project, including low power usage and CUDA cores.

4.3.1 NVIDIA Jetson

NVIDIA describes the Jetson as "an AI supercomputer on a module". The Jetson is power-efficient, which allows us to power it off a battery for long periods of time. Despite this, it has the power to run power-intensive tasks like neural networks, receiving lidar and camera information, and controlling the car. The Jetson TX2 comes with an NVIDIA Pascal GPU with 256 CUDA cores, which allows us to use the CUDA-dependent ZED Camera. It also contains an HMTP Dual Denver ARM CPU, 8GB of LPDDR4 RAM, and 32GB of on-board storage.

4.3.2 Connect Tech Orbitty Carrier

While the Jetson development board that the TX2 sits in provides full size ports and is good for developing software, it is too large to mount on a small car and is not intended for deployment. Since the chip is self-contained, it can be removed from the development board and placed in a different “carrier”. The carrier provides power to the chip, as well as providing access to a range of ports including ethernet, two USB ports, HDMI, MicroSD, expansion headers, and buttons for power, resetting the board, and recovery mode. The Orbitty carrier also conveniently provides mounting holes, which allowed for semi-permanently mount the Jetson/Orbitty package.

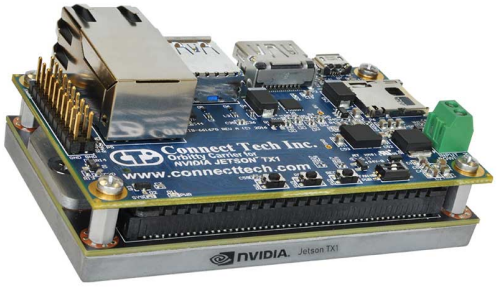


FIGURE 4.2: Jetson in Orbitty Carrier [20]

Using the Orbitty requires some careful consideration of multiple aspects. To power the board, it requires DC power between +9V and +14V. While there is a theoretical maximum power draw of 21 watts, Connect Tech provides measurements which include a stress test of the system, with keyboard, mouse, camera, and HDMI, that used 8.5 watts at 12 volts. This is considered in the wiring, described in section 5.3.

Another requirement to using the Orbitty carrier is flashing some special software provided by Connect Tech. This is done while setting up the Jetson, and is explained in another section. Without this, the system will boot but the peripherals will not work.

4.4 Sensors

Various sensors added to robots allow them to perceive and interact with the world in various ways. The sensors we added allow for different methods of vision, and interpretation of how fast we're moving. These sensors were chosen based on cost, size, and software support.

4.4.1 Stereo Camera

The camera on board the car is a StereoLabs ZED stereo camera. This camera module has two cameras within it, that when combined using stereo vision provide an RGB-D signal (comprised of a standard RGB image with a depth component) [21]. This camera features high frame rates of 30fps+, a wide field of view of 110 degrees, and depth perception up to 20 meters. The combination of

these will allow for six degrees of freedom positional tracking as well as spatial mapping.

4.4.2 Lidar

The robot will contain a small 2D lidar to assist with depth measurements. The robot uses the Slamtec RPLIDAR A2, a "low cost 360 degree laser range scanner" [22]. The lidar is used to create scans of the environment using a rotating laser. This data can be used to map the environment and localize within it. This particular lidar can scan within a 6 meter range at 10hz (600rpm). This results in a 0.9 degree resolution, with 400 points in a complete 360 degree scan.

4.4.3 Microcontroller

To interact directly with voltage-based sensors (such as hall-effect sensors) requires a microcontroller attached to the main computer. This is accomplished using an Arduino Pro Mini, which is a 5 volt 16MHz microcontroller in the Arduino lineup.

4.4.4 Wheel Encoder

For the robot to accurately understand how far it has moved for our localization equations requires precisely tracking how many times the wheels have turned. To do this, we built a wheel encoder comprised of magnets inside each wheel, and a hall-effect sensor to measure the magnetic strength.

The hall-effect sensor changes its voltage when near a magnet. By placing the magnets equidistant from each other around a wheel, we are able to measure the speed of the wheel based on the pulses in voltage provided by the hall effect sensor. The four sensors will be plugged into an Arduino Pro Mini microcontroller development board, which will read the analog voltages and provide a digital signal to the main processing board.

4.4.5 Inertial Measurement Unit (IMU)

he IMU used on the setup on this car is the SparkFun 9DOF Razor IMU M0 which contains an accelerometer, for linear acceleration in the XYZ frame, a gyroscope, for angular velocity about the XYZ axes, and a magnetometer, for angular orientation about the XYZ axes. The sensor has a usb connection for easy data streaming and power, as well as a port for a microSD card if there is a necessity for logging data. This port was not used on this project as the data could be logged through ROS instead.

4.5 Software Structure

The software of the system is what ties the machine together and allows it to operate. In a robot of this complexity there are many moving parts, and the system must be designed in a way that allows for expansion and development of different parts simultaneously. Our system will be built upon the popular Robot Operating System, or ROS [23]. While it has the term "operating system" in its name, it is not an operating system like Microsoft Windows or Canonical's Ubuntu is. Rather, ROS provides many tools, libraries, and other software to build and manage a robot.

4.5.1 Modularity

One of the important parts of this project is being able to test our code quickly without having to spend time in setup getting the rest of the system to work. ROS aids this by being designed to be "as distributed and modular as possible" [24]. In addition to the provided software, ROS also has an extensive community that builds pre-built packages that can be dropped into a robot. This will allow us to focus on the aspects of the task that we're concerned with, and find well-written pre-built packages from the community to fill in the rest of the pieces.

An important aspect of modularity is being able to swap out individual software components quickly and without much additional work, to be able to test and compare, and receive feedback on the state of development. ROS provides a low-level messaging system that makes this possible [24]. For example, if we have Task A sending messages to Task B, we can capture these messages in ROS and play them back later, allowing us to test Task B without needing Task A

(Task A might be capturing data in a certain environment, for example). In addition, if we're comparing Task B1 to Task B2, we can quickly subscribe to Task A's messages on both Task B1 and Task B2, and compare the data later. This disjoint task ideology connected by messages is the core of ROS and allows for faster, more modular development.

5 Vehicle Assembly

This section outlines the physical construction of the RC car robot. This included modifying purchased components, building new sensors, and wiring the car.

5.1 RC Car Modifications

Table 5.1 details the main components of the RC car, what they do, and what modification will be required to work as the robotic platform.

Vehicle Component	**Functionality**	**Modification Required**
Chassis	The chassis is used to connect the individual components of the car, but will also be used to house our research components. We will be building platforms on the chassis to connect our sensors.	Building platforms to be mounted to the chassis.
Wheels	The wheels will have a 3D-printed housing for the magnets necessary for the wheel encoders. This allows us to accurately measure how much the wheels have rotated and inherently the distance travelled.	3D printed magnet holder glued to the inside wall.
Suspension	Connecting the wheels to the chassis, the suspension allows for a smoother, more consistent ride. It will also be home to the hall-effect sensor for the wheel encoder assembly.	Hall-effect sensor attached with adhesive to the arm of the suspension.
Electronic Speed Controller	An electronic speed controller (ESC) modulates the power to the motors from the battery. It will be controlled by our electronics.	Tapping into the sensor input of the ESC.
Battery	Powers the vehicle.	None
Body	The body keeps the internals safe from any collisions.	Cutting out pieces to allow our sensors to reach the outside world.

TABLE 5.1: Base Vehicle Components

5.2 Sensor Mounts

A variety of different sensors mounts were built for the car. Most of these mounts were 3D printed and bolted to the acrylic plate.

5.2.1 Mounting Plate

The Slash 4x4 provides four mounting pins for the body of the car. These mounting pins bolt to the rest of the car, and a piece of acrylic, modeled in CAD and laser cut, is attached using these bolts (see Figure 5.1). This sheet of acrylic is used as a base to mount other components such as the lidar, ZED stereo camera, IMU, NVIDIA Jetson, and USB hub. The design of the body of the car allows for sufficient space to mount the sensors to this acrylic piece without interference. The mounting plate also needs to allow access to the lower platform to remove and replace the battery as needed. The mounting plate covers only part of the bottom section of the chassis to allow for this.

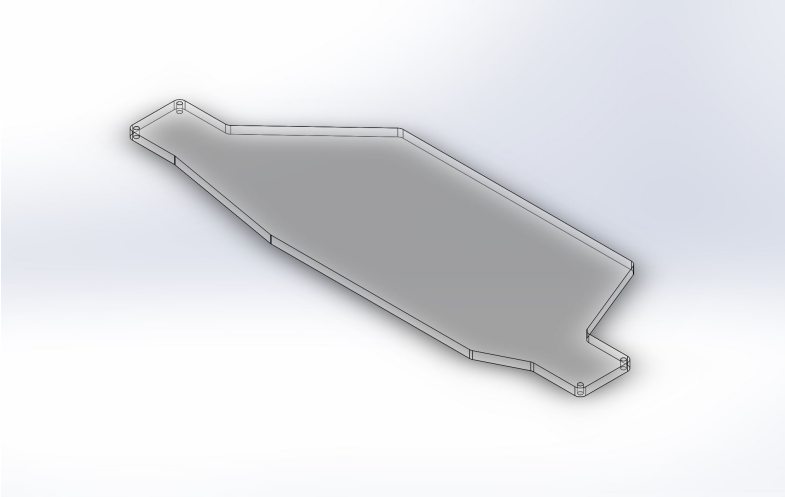


FIGURE 5.1: Electronics Mounting Plate

5.2.2 Cutting the Body

One of the logistical requirements influencing the design is to maintain as much as the outer shell of the body as possible. This is so that it can protect the inner components from damage and maintain the general look and appeal of the car. However, cutting the body shell to allow different sensors to the outside world is necessary. This includes a hole for the ZED stereo camera to see through the

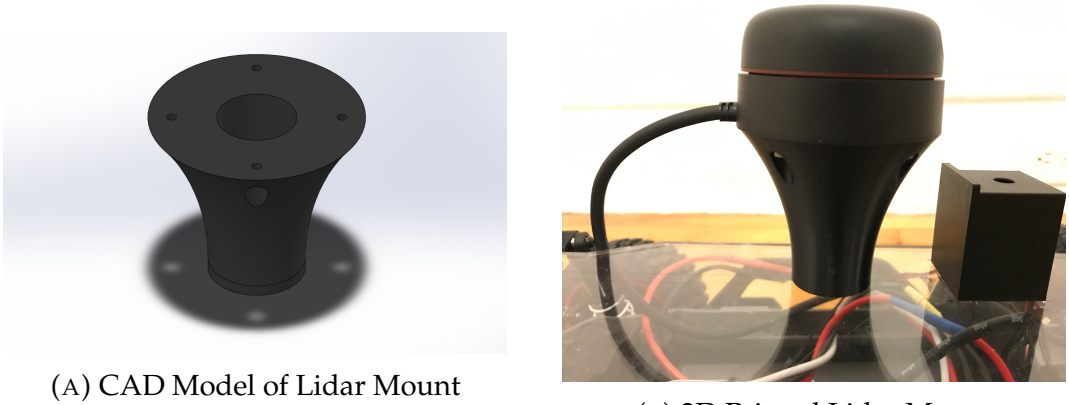
body and a hole to allow the lidar to extend out the top roof of the body. The areas are measured and cut out with the cutting wheel of a rotary tool. Figure 5.2a shows the hole opening and how the Zed stereo camera is installed to see out the front windshield of the body. Figure 5.2b shows how the lidar is mounted so that the wire of the sensor still remains on the inside of the body, however the rotating light emitting and detection sensor remains above the frame of the body.

(A) Mounted Camera (B) Mounted Lidar

FIGURE 5.2: Sensors in Body

5.2.3 Lidar

The lidar sensor mounts ideal requirements included being located centrally on the platform so that it there is an even reading in relation to the car no matter which orientation it is facing. In addition the lidar needs to be outside of the body and unobstructed to give a clear view of the area around the car. The design for the lidar mount places it centrally on the car mounting plate and high enough that it goes through a hole in the shell. The design and resulting 3D printed sensor mount can be seen in Figures 5.3a and 5.3b.



(A) CAD Model of Lidar Mount

(B) 3D Printed Lidar Mount

FIGURE 5.3: Lidar Mount

After mounting the lidar it can be tested to ensure that it is receiving clean data and is not obstructed by the frame of the car. Figure 5.4 shows the results of this data collection. The sensor returns a 2D map representation of the distances to objects. It returns the angle and corresponding distance to the nearest object at this angle.

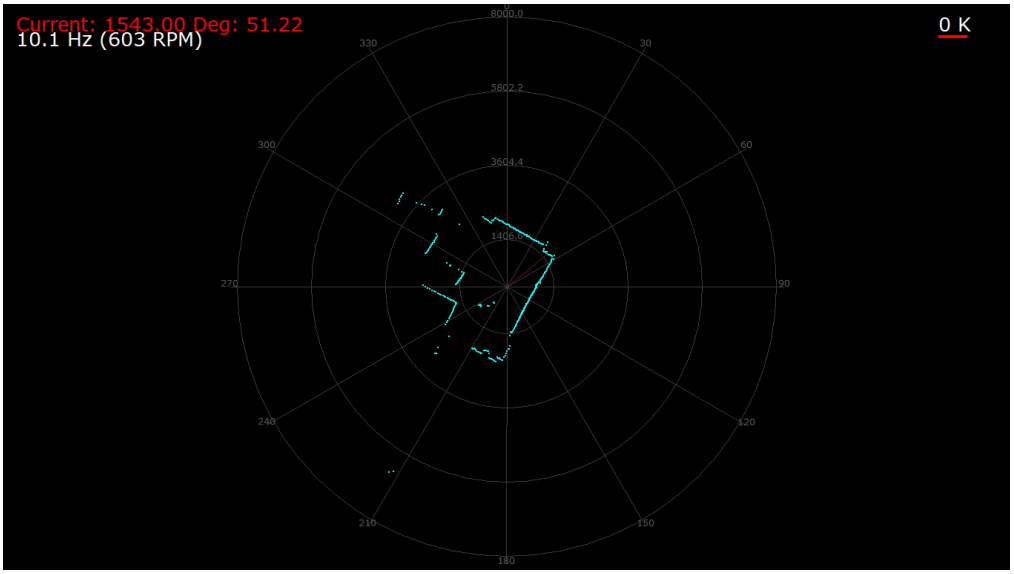
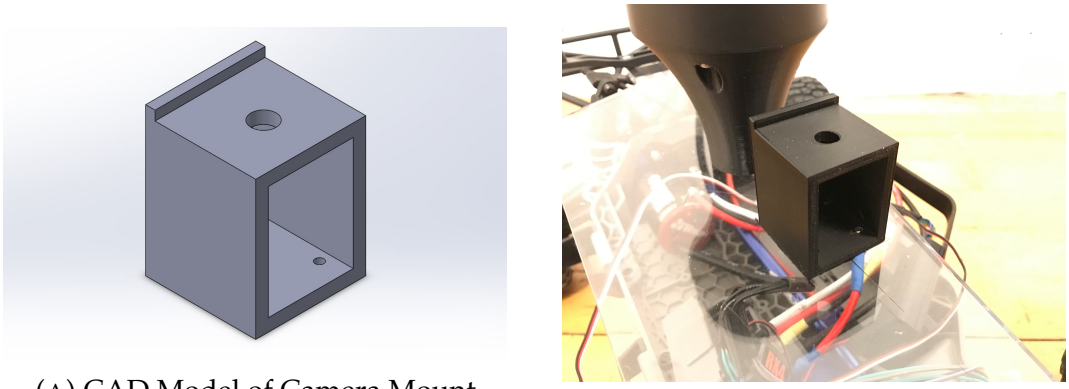


FIGURE 5.4: Lidar Data Map

5.2.4 ZED Stereo Camera

The ZED stereo camera also required being able to see outside of the vehicle's shell. It also needed to be towards the front and high enough to see over the

hood of the car and give a first-person front view from the perspective of the vehicle. The ZED stereo camera also needs to be mounted so that it does not obstruct the 360 degree view of the lidar. Similar to the lidar, the ZED stereo camera needs to be mounted securely so that it will not shift at all during use. On the model for the sensor mount, a small back wall is added to prevent a rotation around the single bottom mounting bolt. In Figure 5.5a and 5.5b the CAD model and associated 3D printed sensor mount can be seen.



(A) CAD Model of Camera Mount

(B) 3D Printed Camera Mount

FIGURE 5.5: ZED Camera Mount

The ZED stereo camera comes with an executable image viewer in the downloadable SDK. To ensure the proper visual clearance, this image viewer can be used to check what the camera can see. The resulting images from the camera can be seen in Figure 5.6. The nearest ground location that can be seen by the car is about a foot away from the front bumper. Therefore when testing the car can achieve data collection up to this distance away from any object before it risks the danger of collision.

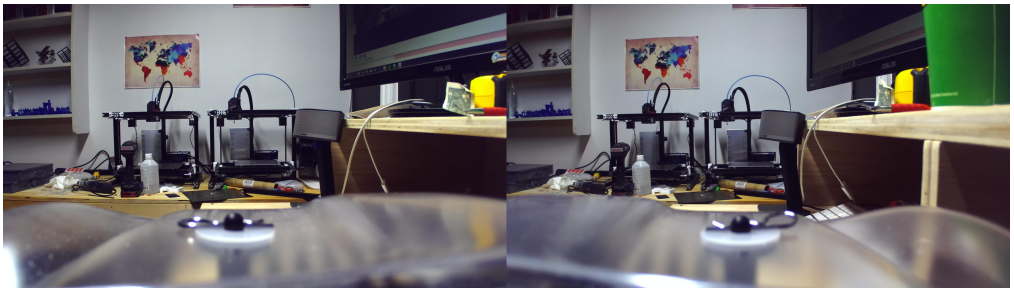


FIGURE 5.6: View from ZED Camera

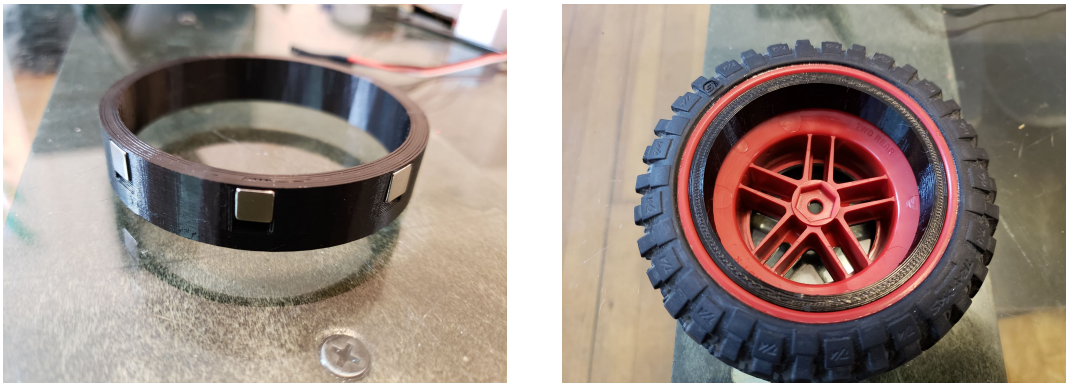
5.2.5 Encoders

To build a dead-reckoning odometry system (which predicts where the robot is based on our driving input) the robot needed to keep track of the amount the wheels have turned, and the steering angle. The steer angle comes from the input to the steering servos, which are motors that provided turning based on specific degree measurements. Since the wheel motors are controlled by voltage, there is no built-in way of telling how many times the wheel has turned.

To address this, the robot requires wheel encoders, which are custom-built for this project. These sensors keep track of the number of revolutions the wheel has made. Knowing this, the software multiplies the number of revolutions by the wheel circumference to get an estimate of how far the robot has traveled.

Magnets

To calculate how far the wheels have turned, there needs to be an external reference, which in this case is provided by magnets. Eight magnets are installed inside each wheel using a custom 3D printed mount that the magnets press fit into. This allows for a consistent and known spacing between each magnet. A necessary consideration when installing the magnets is that the polarities of the magnets needs to be consistent.



(A) Magnet Wheel Hub (B) Magnet Wheel Hub Mounted

FIGURE 5.7: Wheel Magnets

Hall Effect Sensors

A hall effect sensor is an analog sensor that detects the presence and strength of a magnetic field. They are used on the robot as a digital sensor to detect the presence of each magnet as the wheels rotate. Two hall effect sensors are mounted on the arm of the suspension axis of the back right wheel using a custom 3D printed mount. The second hall-effect sensor is used for directional detection.

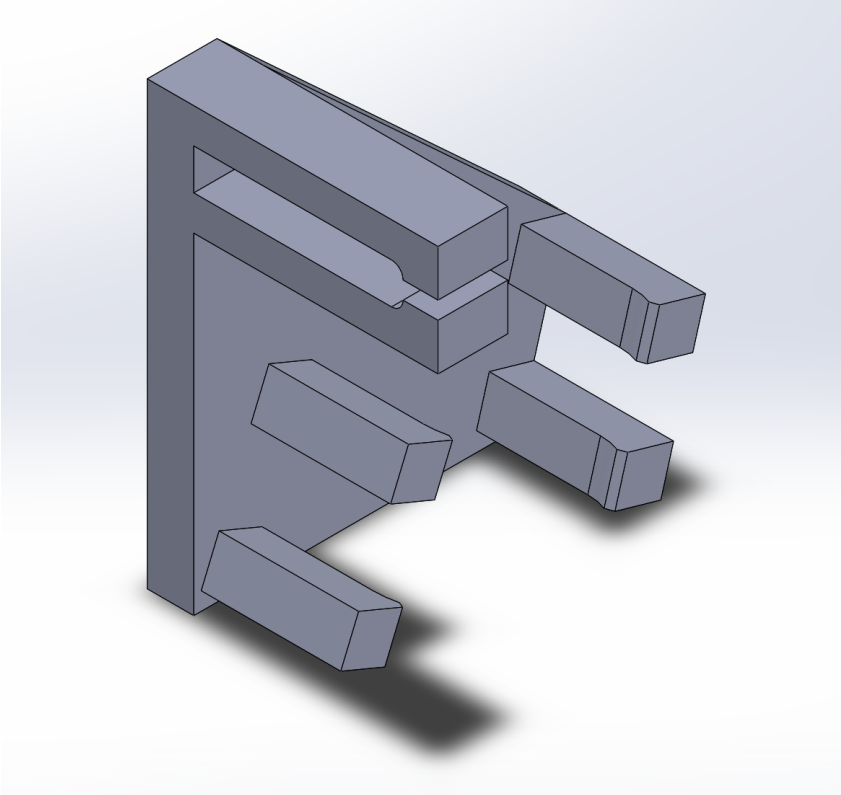


FIGURE 5.8: Back Wheel Encoder Mount

The hall-effect sensor mount allows for consistent reading of the magnet's presence. Once mounted, the sensors were tested by slowly moving the car forward so that the wheels rotate at a consistent speed.

5.3 Wiring

To power the car from a battery requires a wiring harness that creates different voltages and can carry the necessary power to all of the devices. A major consideration was to be able to power the car off of a dedicated power supply to avoid

relying only on the battery. To accomplish this, the wiring harness is designed to accept either a laptop power supply or the LiPo battery, both of which are terminated in EC5 style connectors. There is a separate plug for the drive system and the rest of the computing electronics, which can be chained together. When the battery is connected, the harness should be fully connected to power the entire car. When only using the power supply, it is plugged directly into the computing electronics plug, bypassing the power-heavy drive system.

The computing electronics require different voltages for different components. The NVIDIA Jetson takes an operating range between 9-15 volts DC. To compensate for the different voltages of input power (battery vs. power supply), a 12 volt regulator is present that is rated for the power requirements of the Jetson. The rest of the devices required 5 volts DC, which is the rated voltage of USB. Many of the devices were powered directly from their USB connections. However, normal USB ports on a computer, including the Jetson, are rated at 500mA. To power the connected devices required up to 1.5A. To supply this, a USB hub with separate power supply was used. The USB hub was rated at a maximum of 2A, which is sufficient for our sensors. The power input of the USB hub was connected to a 5V regulator which would be connected to the power in. This allowed for the rest of our sensors to be powered off the USB ports.

Some devices needed to be adapted to use the USB port for power. For example, the lidar used a separate data and power plug. However, the power plug was also 5v. For ease of use, the power plug is adapted to use USB, so it now has a data USB port and a power USB port, both of which are powered into the USB hub.

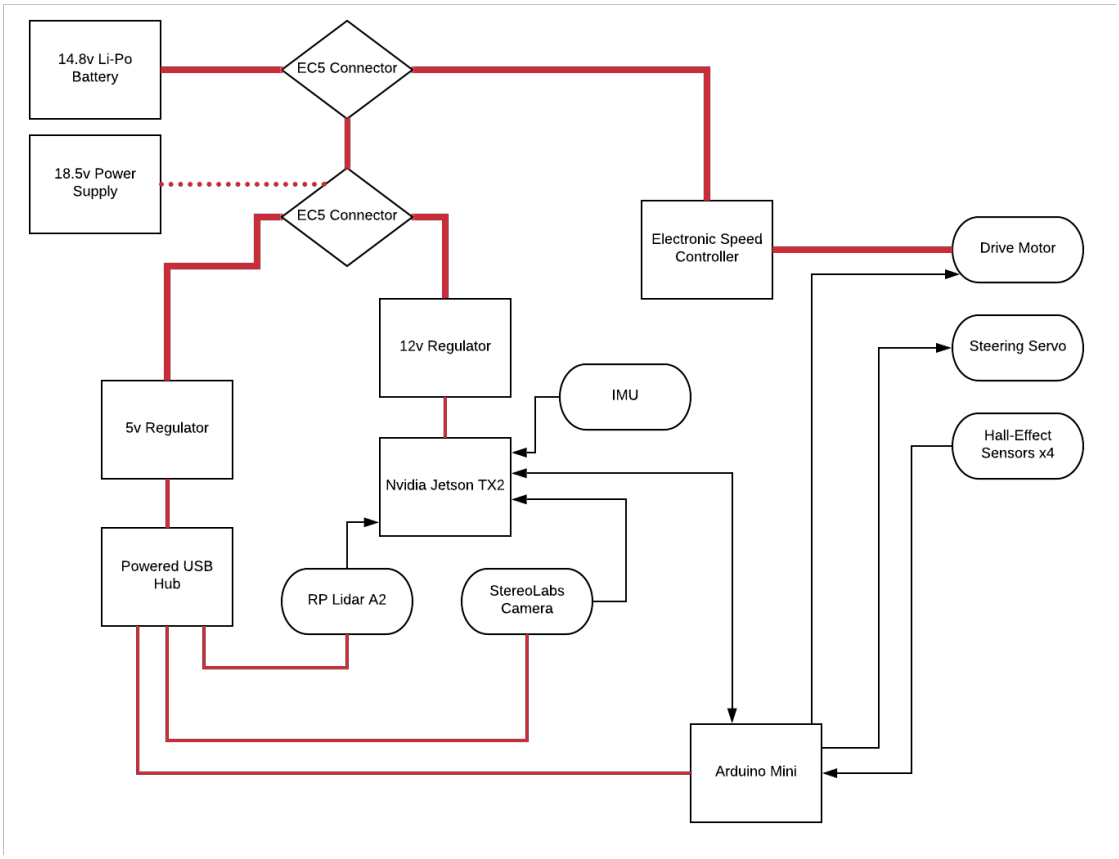


FIGURE 5.9: Electrical Diagram

6 Vehicle Software

Installing the necessary software to run the RC car requires some configuration, as outlined below. Throughout this section the `racecar-mqp` project is mentioned, which is the provided code repository for the project.

6.1 Flashing the Jetson

To begin using the Jetson requires flashing the boot loader and kernel onto the chip using a host computer. The process for doing this is outlined in the Jetson TX2 Developer Kit User Guide [25]. The operation involves booting the Jetson in recovery mode while connected over USB to a host computer running a flavor of Linux. To use the Orbitty Carrier requires adding a Board Support Package supplied by Connect Tech [26]. To install this extra software requires adding files into the NVIDIA install folder before flashing. Once this is completed, the Jetson is flashed using the NVIDIA install script.

The second step is running the Jetpack software as supplied by NVIDIA. This software installs a Ubuntu image, necessary drivers and tools, and third-party software.

6.2 Installing ROS

The Robot Operating System (ROS) is a software framework that encompasses the project and provides the foundation for almost all of the software that is used on the project. Despite its name, ROS is not a traditional "operating system" like Windows or macOS or GNU/Linux. It is a C++ and Python tool package that contains a software framework, command line tools, compiler, and an ecosystem of third party packages [23].

Installing ROS is the first step towards building the car. Even if using the provided project, ROS will still need to be installed onto the system. The version

chosen is important for compatibility reasons. This project uses ROS Kinetic, which is targeted at Ubuntu 16.04 LTS (Long Term Support). This version of Ubuntu will be supported until 2021, at which the system should use a different version of Ubuntu with LTS and its associated ROS installation.

The installation process is outlined on the ROS website wiki [A.1]. A full desktop install is required. The only change to the process is under *1.6 Environment Setup*, where the path of the `racecar-mqp` project (or other ROS project used) is added to the robot's `.bashrc`, and not the default ROS location. An example is below:

```
$ Echo ‘‘source /path/to/devel/setup.bash’’ >> ~/.bashrc
$ Source ~/.bashrc
```

6.3 Git & Version Control

Version control allows multiple people to work on the car, provides backups for the code, and improves the development experience. Committing the project is generally similar to other projects. To eliminate unnecessary files being committed, the project borrowed a .gitignore file from GitHub's gitignore example repo [A.4].

Another difference in this project is the use of the git module feature. This project uses a number of different projects for the ROS community. Each of these is typically held in its own git repository, which we clone into our `src` folder. To maintain each of these projects' own repositories, a git feature is used called *modules*. This allows for a git project that contains multiple sub-projects, which each are their own git repository. When cloning in a new project, it is necessary to add an entry referencing it to the `.gitmodules` file in the root of the project. This allows git to register that folder as a specific git repository.

This changes the workflow when working with these folders. First, when performing a fresh clone of the `racecar-mqp` project, these folders won't be populated with files. Git needs to recursively pull each of these projects. This is accomplished by running

```
git module init && git module update.
```

Another point of consideration is that generally, it is not desirable to change files in these projects. When a change is made, it is considered a change of that

projects repository, not the main `racecar-mqp` repository. To work around this and be able to launch these nodes, any configuration changes or launch files are placed outside in the main project. For example, place launch files in the `racecar-mqp/launch` folder and any configuration files in the `racecar-mqp/config`.

6.4 Driving the Car

Depending on the system being tested, it may be necessary to drive the car around manually. The controller chosen was a standard Microsoft Xbox controller, because of its relatively good support in Linux. In standard versions of Ubuntu, the controller is recognized automatically as a gamepad input. However, the version that is installed on the Jetson does not do this. To enable support, a third-party piece of software is used to initialize it called XboxDRV [A.3].

A node was written to translate the button presses of the Xbox controller to drive commands. The left and right triggers were chosen to drive the motors in forward and reverse, and an analog stick for steering. A scale was added to convert the scale of the axes from the Xbox controller to what was read by the Arduino controlling the motors.

6.5 IMU

The IMU comes preprogrammed with example Arduino firmware, however in order to translate the sensor data as a ROS topic, a separate ROS package was used [A.2]. The package used is developed by Kristof Robot and requires initial Arduino firmware to be installed and then calibration steps to be completed.

From this a `sensor_msgs/imu` data typed topic is formed which contains a Header which is necessary to set the frame_id of the IMU. Its orientation reported from the Magnetometer set as a `geometry_msgs/Quaternion`, its angular velocity reported from the gyroscope set as a `geometry_msgs/Vector3`, and its linear acceleration reported from the accelerometer set as a `geometry_msgs/Vector3`. Along with this the covariances for the three datasets is also available as an array of length 9 that is row major about X Y Z. The coordinate frames used by the Razor_AHRS firmware is different from what is physically printed on the board as well as what ROS defines as the standard coordinate frames in REP-103. The Razor_AHRS firmware uses the following coordinate frame:

- X axis points forward (in the direction you plug the USB in)
- Y axis points to the right
- Z axis points downward

The REP-103 standard uses the following for its right hand coordinate standard:

- X axis points forward (in the direction you plug the USB in)
- Y axis points to the left
- Z axis points upward

The `razor_imu_9dof` node handles this transformation when it creates the `/imu` topic. Because of this the transformation about the X axis does not need to be done by the user, however it is important to keep in mind as the data read from an Arduino serial monitor using the firmware will differ from the data read from the created ROS topic. Due to the way the car is structured and the way the IMU is mounted this standard frame does not align with the REP-103 standard frame fixed to the base link of the car, where the X axis is pointing in the forward direction of the car. To adjust for this a static transformation is created that is π radians about the Z axis. This is covered further in section 6.9.

6.5.1 IMU Calibration

The ROS wiki for the `razor_imu_9DOF` package linked before has step by step instructions on the calibration of the IMU. This section will include specific instructions that may be unclear or that should be enforced. The accelerometer, the gyroscope and magnetometer need to be calibrated in these steps to provide an accurate measurement. Before calibration the sensor data will be virtually unusable as some axes of measurement may not be correctly aligned. The calibration will align the measurements along with the assigned coordinate axes as well as compensate for any wrongly scaled sensor. Since this package is external from this porject and is installed as a module, we do not want to make changes directly in the cloned directory. Instead of creating the `my_razor.yaml` file inside the `config` folder of the `razor_imu_9DOF`, it is recommended to create the file inside a separate `config` folder in the base project repository. In an IMU launch file created to run the node, a `.yaml` file is passed in as a `rosparam`. This line of the

launch file should reference the `.yaml` file containing the calibration values. An important note with launch files is that the `file` parameter of `rosparam` needs to be an absolute path to where the file is located.

6.6 ZED Camera

The Stereo Labs ZED Camera features two cameras that are used together to create three-dimensional stereo vision. Use of this module requires the ZED SDK [A.3]. This SDK requires that the computer feature NVIDIA CUDA cores, found on NVIDIA graphics cards. Without this, the SDK will not be installed. The Jetson features 256 CUDA cores and can install the SDK.

StereoLabs provides a ROS package for use with the ZED Camera. This package features the capability to create point clouds, Odometry, and more.

6.7 Lidar

To use the lidar the only setup required is the installation of the `rplidar_ros` package which is installed by cloning the repository [A.2] into the project's `src` directory. As this is a third party package it is important to remember to include it as a submodule, and any changes made to the source code be done outside of the cloned repository.

6.8 Encoders

Collecting data from the encoders and turning that into an estimation of how far the robot has traveled is essential for the odometry system. This project uses two encoders mounted on a single wheel, which will give both direction and the number of rotations of the wheel. This system works by triggering a "callback" each time either encoder changes state. When a magnet is over an encoder, it pulls it low, and when it is not, it is pulled high. Based on whether we are going from low to high or vice-versa, the robot can calculate how far it has moved and the direction the wheel is turning. The calculation for how far the wheel has moved comes from the circumference of the wheel. These distances need to be segmented by the arclength between magnets and the arclength over magnets, on the outside edge of the wheels.

6.9 Transforms

In robotics many different coordinate frames, known simply as frames in ROS, must be used for different components work together. These coordinate frames allow the transformation of relative distances and locations into another frame's point of view. For example, if you had a distance reading from a lidar, the robot might need to know where that point is in the greater world. In this instance the software can translate the lidar reading into the world coordinate frame. To translate between different frames transforms must be created. Transforms are the `x, y z, roll, pitch, yaw` difference between two different frames. ROS uses a library called `tf2` to keep track of all transforms so that all nodes have access to them and they only need to be defined once. There are two different types of transforms, static transforms and dynamic transforms.

6.9.1 Static Transforms

Static transforms are between frames that are fixed and don't change, such as transforms from sensors on the car to a common frame. This common frame is known as `base_link` and is located in the center of the rear axle. All the sensor frames have a static transform to the `base_link` frame, allowing the robot to "tie in" each sensor to a single reference point. Static transform are set as XML in the launch file using the `static_transform_publisher` [A.4] in the `tf2_ros` package and should be put before any node that may require that transform. The static transforms to `base_link` for this project can be seen in Section 6.1.

6.9.2 Dynamic Transforms

Dynamic transforms change over time, such as the cars position compared to where it first began. The `odom` frame is initialized at the cars initial position, a transform is then dynamically published, by the `robot_pose_ekf` node, from the `base_link` frame to the `odom` frame (i.e. the position of the car compared to where it first started). Another dynamic transform is between the `map` frame and the `odom` frame, this however is only published if using the Vicon system (which uses the `map` frame) or when running localization software as otherwise the car does not know where in the `map` frame the `odom` frame is.

6.9.3 Standard Frames for Mobile Robots

ROS standards documents REP 103 and REP 105 define a preferred standard for coordinate frames for mobile robots along with a standard orientation. Most external packages follow and expect these frames to be set so it is important to follow them. At any time a coordinate frame should only have one parent but can have multiple children frames. This allows for multiple maps to allow for transitions between them if datasets become too large, allows for multiple robots, `base_link` frames, within a single map, and multiple sensors connected to a single robot, which is the most significant to most projects. The defined standard coordinate frames can be seen in Figure 6.1

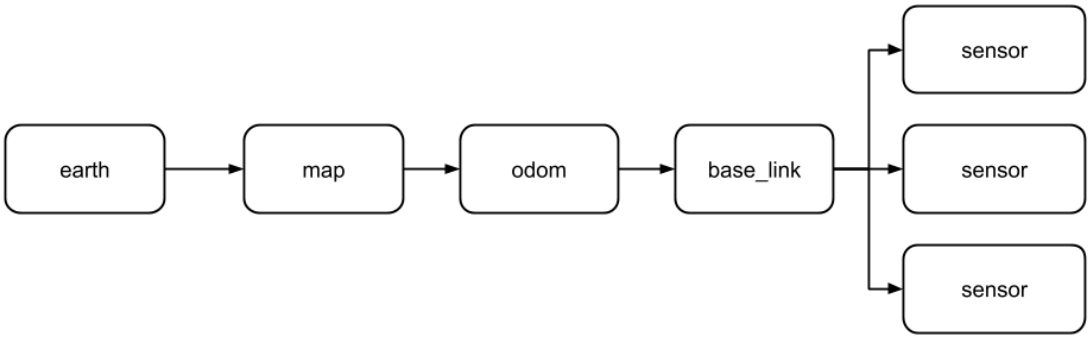


FIGURE 6.1: REP 105 Standard Coordinate Frames

6.10 Odometry

The `odom` node is a custom written node to generate dead reckoning odometry data based on the wheel encoders and the set angle of the front wheels. The node subscribes to the `/drive/steer` topic for the angle of the front wheels and the `/encoder/velocity` topic for the wheel velocity. The change in position of the car over time is calculated using the bicycle model [A.4] which approximates the dynamics of the car. Then, using the time since the last measurement and the previous location of the car a transform from the `base_link` frame and the `odom` frame can be calculated. The transform is not published as a transformation however, it is instead publish as an `Odometry` message on `/bicycle/odom` to be fed into an extended Kalman Filter, as described in 6.11. The node also publishes its current transform as a `PoseWithCovarianceStamped` on `/bicycle/pose`

so that it can be easily turned into a path for visualization and debugging purposes.

6.11 EKF

An EKF or Extended Kalman Filter [A.4] is an algorithm for fusing unreliable data from multiple sensors and producing a more accurate result. On the car the `robot_pose_ekf` package is used to accomplish this. It is configured using the `launch/robot_pose_ekf.launch` file and subscribes to the dead reckoning odometry (`/bicycle/odom`), the IMU (`/imu`), and the visual odometry produced by the ZED camera (`/zed/odom`). The topic names are remapped in the launch file because the node expects them on `/odom`, `/imu_data`, and `/vo` respectively. This node is what publishes the transform from the `base_link` frame to the `odom` frame. The node also published the current transform as a `PoseWithCovarianceStamped` on `/robot_pose_ekf/odom_combined` which can be useful for visualization and debugging.

7 Vehicle Workflow

When using the car it is necessary to have a monitor, mouse, and keyboard available to be able to interact with the car's onboard computer. For the most part the Jetson behaves like a normal computer, and much of this project was developed while using the Jetson running on the car.

7.1 Turning The Car On

There are two ways the car can be powered, depending how the car will be functioning. If the intention is to drive the robot it must be powered by the battery, but for regular development and testing of electronics other than the battery the car can be plugged into the wall. There is one connector that connects to all of the electronics, and two that attach to the motor for it to be attached in parallel. The power source can be plugged directly into the electronics connector. If the motor is also to be attached, it has two plugs, that effectively places it between the power source and the rest of the electronics.

Once the car is plugged into a power source the power button on the Jetson (located underneath the Jetson near the Lidar) can be pushed in order to start the boot sequence. The boot sequence will appear on the monitor in a short time.

7.2 Running The Code

The code for this project is contained in the `/src/racecar-mqp` directory. Running the entire project involves running many different nodes on top of `roscore`, which is the base software of the rest of the system. Different configurations can be contained in launch files, which are launched with `roslaunch`. This project contains a `racecar.launch` file which starts the different sensors and pieces of software with one command `roslaunch racecar.launch`. The only major piece

of software needed externally is the Xbox controller. This is run using software called `xboxdrv`. On the project's car this is run as a daemon on startup.

7.3 Setting Up Your Laptop For Remote Control

ROS contains the ability to run different nodes on different computers to create a distributed network. The first steps to setting up a laptop for remote control is to ensure that you have ROS installed on the remote computer. Then connecting the racecar to the remote laptop is the same as connecting multiple systems together over ROS for any project. While SSH'd into the car or connected through HDMI, view the IP address of the car using the `ifconfig` command in Linux. In another terminal open up the `/.bashrc` file and change the `ROS_MASTER_URI` to be the IP address of the car followed by the port, which by default is 11311. As an example the line should look like the following:

```
export ROS_MASTER_URI = https://(ipaddress):11311
```

While still in the `/.bashrc` file of the car change the `ROS_HOSTNAME` to be the same value, without the port. Updating the `/.bashrc` file on the car before trying to do further work. From the remote laptop the same process needs to be done, but the `ROS_MASTER_URI` needs to be set to the IP address of wherever the master node is being run, in this case the master node is run on the car. Again ensure to source the `/.bashrc` file before doing further work. The ROS master node needs to be running before trying to connect to run any nodes from the remote computer. At this point nodes can be run from a remote laptop and topics can be echoed as well as viewed in visualization tools such as RVIZ. It is recommended to create a secondary CMAKE file that can be run on the remote laptops that may not be able to have all of the module dependencies installed that will be solely run on the car. For example, the ZED SDK requires CUDA to be installed as previously mentioned. This however requires NVIDIA drivers which may not always be possible on remote laptops.

7.4 Writing Code For The Car

The general structure for developing software on the car is similar to how most ROS based systems are developed, with packages the contain nodes and our

run in some combination in a launch file. Packages are created with the intent of having a general purpose that can contain ROS nodes, data sets, configuration files. The intent of separating the code into packages is to create an easy to consume and highly reusable environment. For example, a useful package may be a dead reckoning odometry package that contains nodes for calculating the movement of the car and a configuration file for the specific dimensions of the car wheels. A package can be reliant on other packages to publish certain data, but its functionality should not be dependent on something other than a published topic. This allows for this dead reckoning odometry package to be easily switch with another odometry package without having to change the rest of the software. However both of these odometry packages could be reliant on a third package the publishes IMU data. The modularity provided by ROS allows for the freedom of developing software without understanding the full underlying subsystems and the ins and outs of every package involved. Each of the sensors were set up and installed so that you can run there corresponding packages and subscribe to the data published.

In the ROS basic tutorials [A.1] it describes how to create a basic ROS package in either Python or C++ and specify its specific dependencies. Within a package you can create nodes that function similar to writing code for any other projects, scripts for Python and more object orientated based code for C++. It is important to follow the information on creating the `CMakeLists.txt` and `package.xml` so that catkin will work correctly. It is possible to have compilation errors in the code but when building the catkin package is built, the developed code may not be included, so the errors will not be caught.

8 Results

8.1 Objectives and Requirements

Tables 3.1 to 3.5 discuss the initial objectives and requirements that were set for the end goal of the project.

In terms of objectives, each of the minimum targets were achieved. The platform was fully constructed and installed with sensors that were not only necessary for this project but leave room for future projects to research more in depth on each of the components. The system also contains the documentation required to read all the data from these sensors and can be controlled through multiple computers through its ROS environment. Early stages of geometric feature detection were implemented. Objects were decided based on some real life possible objects an autonomous car may encounter in a real life scenario. In the case of the small scaled car, similarly small scaled objects were chosen to detect.

Each of the target objectives were recognized as over ambitious as the terms progressed, due to the amount of initial work required to set up a platform that would allow future research to be done on. Therefore these objectives became possible future directions for work to be done on the car. These are to be further discussed in Chapter 9.

Tables 3.4 and 3.5 discuss the requirements necessary to be considered a successful platform. Each of these requirements both functional and non-functional were met and are vital to future development on the car platform. The operation of the car, logging and visualization of data are all complete. The benefits of being able to remotely control the car and log the data is that tests can be done within the VICON, which is seen as the ground truth and then brought to different areas to work on the car instead of having to test on the VICON system each time a change is made to the platform. The logging and playback of data is done through recording topics in ROS bags. In terms of non-functional requirements, as mentioned previously throughout the report the modularity and incremental build process is an important portion of the project since it allows

for future developers to proceed as they wish. While testing over long periods of times, the battery would last easily up to two hours at a time before it would need to be recharged. A low voltage battery alarm was installed in the electrical system so that the car would notify the users when one of the cells drops below 3.3V. It is equally as important to not let the battery to drop to low in voltage as it is to not overcharge the battery. The electrical system was designed so that all of the systems except for the motor can be operated off of a wall outlet supplied cable. So data playback and various tests dealing with the lidar and camera can be done without the worry of battery charge levels.

8.2 Encoders

The magnet wheel hubs, as seen in Figure 5.7a, has room for eight magnets to be installed. However since the width of magnet and the width of the area that does not contain a magnet is not equal, every time the hall effect senses a change in magnetic field it can not be assumed that an eight of the wheel turn has been completed. While turning the wheel at a constant rate the changes between high and low logged by the hall effect sensor can be seen. The results of the test can be found in Figure 8.1.

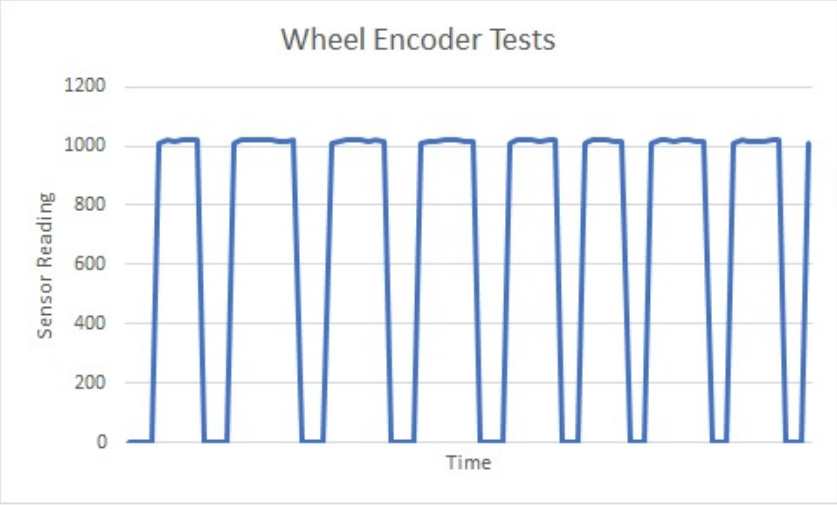


FIGURE 8.1: Wheel Encoder Test Results

The amount of time the sensor reads high, when the hall effect sensor does not detect a magnet, is longer than the time it detects a magnet. This was expected and accounted for in the calculation of wheel velocity by adding the arc

radius of the non magnet area when the hall effect sensor attached to the Arduino shifted from high to low and then added the arc radius of the magnet when it shifted from low to high.

The velocity was calculated based on the time taken between state changes and the arc radius previously mentioned. The time calculation was done on the Arduino as opposed to in ROS so that it was as close to the state changes as possible so that there would be little time discrepancy from the time it takes to transmit state change messages. Also included in the velocity calculation was a time out that would reduce the value of the velocity to exactly 0.0 so that drift would not occur. If the car was moving and then came to a quick stop the velocity without this timeout only updated on a state change. The value for the timeout is directly proportional to the previous change in time between states. Therefore the faster the car is moving the faster the time out is, and the slower the car is moving the longer the timeout is. This allows for very slow motion of the car as well as removes the limitations on how fast the car can brake.

In order to determine the direction the car one wheel needed to have two hall effect sensors attached to it, since one sensor simply reads transitions between high and low values regardless of the direction the wheel is turning. Therefore there are two hall effect sensors attached to the back right wheel relative to the forward moving direction of the car. There are interrupts for both sensors however only one of them checks and updates the wheel velocity while the other one only checks the direction of wheel motion. If the sensor that checks for direction reads the same state as the other encoder then the wheel is rotating in reverse relative to the car, if the direction encoder is on alternating states then the car is moving forward. There is a simple positive and negative factor that is then multiplied to the velocity calculation.

8.3 Odometry

The odmetetry uses a number of sensors to achieve accurate results. The base of the system is a custom written encoder based dead reckoning system. The dead reckoning system combines the distance the car has travelled and the direction the steering was pointed to estimate where the robot is at any point in time. The ROS node that was written to compute this is based on the bicycle model, combining the steering input and the wheel encoders. The end result of this was

a system that was accurate in the distance travelled, but the steering angle was prone to drift. This is likely because the model used does not take into account any slip in the wheels and suffers from imprecise measurements. However, it provides a starting point to incorporate other measurements in the Extended Kalman Filter.

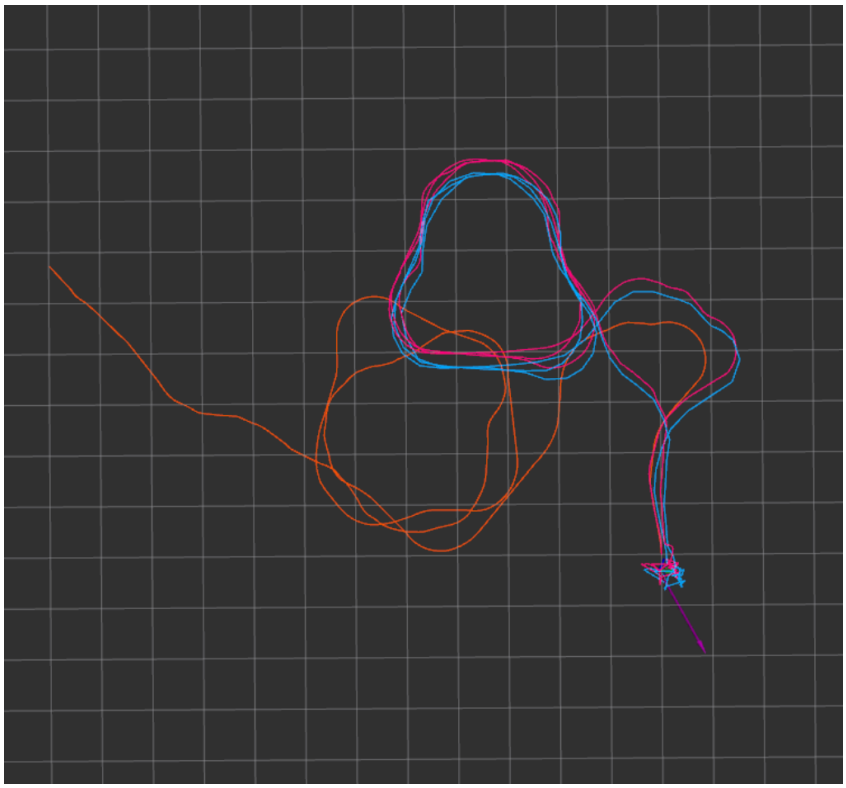


FIGURE 8.2: Odometry Results

Red is Dead Reckoning, Blue is Camera, Pink is EKF

The IMU gives an orientation that is used to help improve the dead reckoning odometry. The ZED stereo camera ROS package also publishes a visual odometry which can be used to improve accuracy. All the odometries and orientations are all combined by passing them to an Extended Kalman Filter which fuses them all together, based on covariance matrices provided by each, to create a much more accurate combined odometry. A visualization of the different odometries can be seen in Figure 8.2

9 Conclusion and Future Projects

A small scale RC Car platform was successfully developed that allows for future students to continue work on the car as well was documented with the possibility to create multiple platforms if desired. First a suitable hardware platform was set up, then the establishment of software tools and paradigms, and finally our own research. It is possible for future students to pull out specific components of the base system to be developed on further.

The state of the project at the time of this writing leaves a lot of opportunity for future projects and research. Using the assembled robot or constructing another using this guide provides a solid foundation for research into robotics topics such as localization, mapping, computer vision, machine learning, and more.

Since modularity was one of the main factors in the development of this platform there is room to pick a certain component of the car and improve on that without parallel components stopping its development. If a team wants to develop solely with the stereo camera to work on an object detection and tracking package this can be done without interference with how an odometry package was developed.

9.1 Examples

The following is a list of recommended projects that could be completed based on the current state of the car with relation to what sensors and fields are involved. Most were derived from the target objectives that were not completed for this particular project.

9.1.1 Improved Odometry

The odometry system in this project would not be sufficient for a precision application or driving over great distances. This could be further improved, possibly

with more accurate encoders or increased external sensors such as GPS. Another option is to create a Kalman Filter or other particle filter from scratch.

9.1.2 SLAM

There are packages in the ROS ecosystem that can perform SLAM with different sensors, or as a project SLAM could be implemented from scratch. This could combine different types of sensors, including the depth map created by the ZED, the lidar scans, or other sensors added to the car such as radar.

9.1.3 Path Planning and Trajectory Generation

Once a map is generated of the environment around the robot, it may be desirable to create a path from one location to another. For example, the situation might be to interact with the environment in a certain way and move about it from one point to another. This requires creating a path for the robot to travel along. This project would need to be able to create paths between the robot and a point of interest, and constrain the path to movements that the robot can follow.

9.1.4 Object Detection

The NVIDIA Jetson provides computing power and tools for training machine learning models. This could be used for developing object detection through the onboard cameras. The semantic SLAM mentioned in the background works on geometric feature detection, but this could also theoretically be performed with other object detection methods. Other uses would be to identify objects of interest and be able to "interact" with them in some way. This approach may be useful for the path planning project.

A Helpful Links

A.1 Tutorials

1. Installing ROS on Ubuntu
2. Creating a ROS Package

A.2 Package Documentation

1. IMU ROS Package
2. ZED ROS
3. Lidar ROS Package
4. Robot Pose EKF

A.3 SDKs and Drivers

1. Xbox Controller Driver
2. ZED Stereo Camera SDK

A.4 Other Resources

1. Bicycle Model Explanation
2. Extended Kalman Filter Description
3. ROS TF2 Static Transform Publisher Documentation
4. Gitignore Example

B Bill of Materials

RC Robotic Car Bill of Materials

This list is everything we used to build our car. Most parts can be substituted if needed. Use as a reference and research each part.

Category	Manufacturer	Name	Part No.	Description	Link	Price	Notes
RC Car	Traxxas	Slash 4x4 Platinum	6804R	RC Car, incl. motor, chassis, wheel, shell	https://tinyurl.com/y44xwv6	$429.99	Features plenty of room under shell to house electronics
	Zippy	Flightmax Batteries	906700308-0	18.4v 8400mah 4s2p LiPo battery	https://tinyurl.com/y7lh3jcz	$87.17	Can use other LiPo battery if compatible
	Power Peak	D7 AC/DC Charging Station	HRC44249	LiPo Charger	https://tinyurl.com/y6c2r3bj	$259.99	Any compatible LiPo Charger will work
	Venom	Low Voltage Monitor	644	Low Voltage Monitor	https://tinyurl.com/yxrb68xl	$10.99	Essential to avoid over-discharging batteries
Electronics	NVIDIA	Jetson TX2 Development Kit		Computer w/ Development Board	https://tinyurl.com/ycmwbwx8	$599.99	May elect to purchase only the processor
	Connect Tech	Orbitty Carrier	ASG003	Carrier for TX2 Processor	https://tinyurl.com/y93sslg9	$174.00	
	StereoLabs	ZED Stereo Camera		Stereo Camera w/ ROS support	https://tinyurl.com/yx6pk5n	$449.00	
	SlamTec	RPLIDAR-A2		Two-dimensional Lidar	https://tinyurl.com/y243v74u	$319.00	A newer version is available (A3)
	Arduino	Nano	A000005	Microcontroller for Sensors	https://tinyurl.com/y9yo5hns	$22.00	Other microcontroller can work if it has at least two interrupts
	SparkFun	9DoF Razor IMU M0	SEN-14001	USB IMU	https://tinyurl.com/j4rw7j	$35.95	
	Adafruit	Hall Effect Sensor	US5881LUA	Magnet-sensor for wheel encoder (2 required)	https://tinyurl.com/y386wf8a	$4.00	
Electrical	Generic	Laptop Power Supply		Used to power electronics when testing	https://tinyurl.com/yy7hk4am	$15.84	12v or higher, 3A or higher. Any PSU will work
	Generic	EC5 Connectors		Connects to LiPo	https://tinyurl.com/y5bqmndh	$4.99	
	DROK	DC Voltage Buck Converter	LM2596	Adjustable Voltage Regulator (2 required)	https://tinyurl.com/y4cxoxgt	$19.98	This adjustable voltage regulator supports up to 3A and has screw terminals
	Generic	USB to DC		Adapter for Lidar Power			
Other	TP-Link	N300 Travel Router	TL-WR802N	Acts as access point	https://tinyurl.com/yxz3ldtk	$27.63	USB WiFi adapters are an option but require compiling from source for the Tegra Processor
	Logitech	MK345 Keyboard Mouse	920-006481	Wireless KB/Mouse combo w/ one USB receiver	https://tinyurl.com/y38fn4zl	$32.49	Any KB/Mouse will work
					Total:	**$2,493.01**	

Bibliography

[1] M. Weber, *Where to? A History of Autonomous Vehicles*, 2014.

[2] EUREKA, *PROMETHEUS - Programme for a european traffic system with highest efficiency and unprecedented safety*, 1987. [Online]. Available: https://www.eurekanetwork.org/project/id/45http://www.eurekanetwork.org/project/id/45.

[3] DARPA, *The DARPA Grand Challenge: Ten Years Later*, 2014. [Online]. Available: https://www.darpa.mil/news-events/2014-03-13http://www.darpa.mil/NewsEvents/Releases/2014/03/13.aspx.

[4] J. Karsten and D. West, *The state of self-driving car laws across the U.S.* 2018. [Online]. Available: https://www.brookings.edu/blog/techtank/2018/05/01/the-state-of-self-driving-car-laws-across-the-u-s/.

[5] Navigant Research, "Navigant Research Leaderboard: Automated Driving Vehicles", Tech. Rep., 2018, p. 64. [Online]. Available: https://www.navigantresearch.com/reports/navigant-research-leaderboard-automated-driving-vehicleshttps://www.navigantresearch.com/research/navigant-research-leaderboard-automated-driving-vehicles.

[6] M. DeBord, *Google Car project history - Business Insider*, 2018. [Online]. Available: https://www.businessinsider.com/google-car-project-history-2018-8.

[7] M. Behl, H. Abbas, and R. Mangharam, *F110 Autonomous Race Cars*, 2016. [Online]. Available: https://mlab-upenn.github.io/f110/.

[8] J. Chen, A. Huynh, Z. Jiang, and Z. Wu, "Self-driving on 1/10 racer car",

[9] Y. T. Wang, C. C. Peng, A. A. Ravankar, and A. Ravankar, "A single LiDAR-based feature fusion indoor localization algorithm", *Sensors (Switzerland)*, vol. 18, no. 4, 2018, ISSN: 14248220. DOI: 10.3390/s18041294.

[10] J. A. Hesch, D. G. Kottas, S. L. Bowman, and S. I. Roumeliotis, "Camera-IMU-based localization: Observability analysis and consistency improvement", *International Journal of Robotics Research*, vol. 33, no. 1, pp. 182–201, 2014, ISSN: 02783649. DOI: 10.1177/0278364913509675.

[11] I.-r. System, W. Monte, C. Localization, J. Wolf, W. Burgard, and H. Burkhardt, "Robust Vision-Based Localization by Combining", vol. 21, no. 2, pp. 208–216, 2005.

[12] E. Karami, S. Prasad, and M. Shehata, "Image Matching Using SIFT , SURF , BRIEF and ORB : Performance Comparison for Distorted Images Image Matching Using SIFT , SURF , BRIEF and ORB : Performance Comparison for Distorted Images", no. February 2016, 2015. DOI: 10.13140/RG.2.1.1558.3762.

[13] S. Gidaris and N. Komodakis, "Object detection via a multi-region & semantic segmentation-aware CNN model", *arXiv:1505.01749v3 [cs]*, vol. 2015 Inter, no. 1, pp. 1134–1142, 2015, ISSN: 15505499. DOI: 10.1109/ICCV.2015.135. [Online]. Available: http://arxiv.org/abs/1505.01749v3.

[14] D. G. Lowe, "Distinctive image features from scale invariant keypoints", *International Journal of Computer Vision*, vol. 60, pp. 91–11 020 042, 2004, ISSN: 0920-5691. DOI: http://dx.doi.org/10.1023/B:VISI.0000029664.99615.94. [Online]. Available: http://portal.acm.org/citation.cfm?id=996342.

[15] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, "ORB: An efficient alternative to SIFT or SURF", *Proceedings of the IEEE International Conference on Computer Vision*, pp. 2564–2571, 2011, ISSN: 1550-5499. DOI: 10.1109/ICCV.2011.6126544.

[16] M. Marszalek and C. Schmid, "Semantic Hierarchies for Visual Object Recognition", *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2007, ISSN: 10636919. DOI: 10.1109/CVPR.2007.383272.

[17] S. L. Bowman, N. Atanasov, K. Daniilidis, and G. J. Pappas, "Probabilistic data association for semantic SLAM", in *Proceedings - IEEE International Conference on Robotics and Automation*, 2017, pp. 1722–1729, ISBN: 9781509046331. DOI: 10.1109/ICRA.2017.7989203.

[18] F. J. Looft, "Systems Engineering and Capstone Projects", no. August, 2016.

[19] Traxxas, *Slash 4X4 Platinum: 1/10 Scale 4WD Electric Short Course Truck with Low CG chassis | Traxxas*. [Online]. Available: https://traxxas.com/products/models/electric/6804Rslash4x4platinum?t=overview.

[20] Connect Tech Inc., *Orbitty Carrier for NVIDIA® Jetson™ TX2/TX2i/TX1 - Connect Tech Inc.* [Online]. Available: http://connecttech.com/product/orbitty-carrier-for-nvidia-jetson-tx2-tx1/.

[21] Stereolabs, *ZED SDK for Jetson | Stereolabs*. [Online]. Available: https://www.stereolabs.com/developers/nvidia-jetson/.

[22] SLAMTEC, *RPLIDAR A2 Introduction and Datasheet*, 2016. [Online]. Available: https://download.slamtec.com/api/download/rplidar-a2m4-datasheet/1.0?lang=en.

[23] ROS.org, *ROS.org | About ROS*. [Online]. Available: https://www.ros.org/about-ros/http://www.ros.org/about-ros/.

[24] ——, *ROS.org | Is ROS for Me?*, 2015. [Online]. Available: https://www.ros.org/is-ros-for-me/http://www.ros.org/is-ros-for-me/.

[25] NVIDIA Corporation, *Jetson TX2 Developer Kit*, 2017.

[26] Connect Tech Inc., *NVIDIA® Jetson™ TX2/TX2i/TX1 Solution Support - Connect Tech Inc.* [Online]. Available: http://connecttech.com/support/resource-center/nvidia-jetson-tx2-tx1-product-support/.