# An Experimental Platform for Implementing Advanced Algorithms in Scaled Autonomous Cars

---

A Thesis

Presented to

the Faculty of the School of Engineering and Applied Science

University of Virginia

---

In Partial Fulfillment

of the requirements for the Degree

Master of Science (Computer Engineering)

by

Varundev Suresh Babu

May 2018

# Approval Sheet

This thesis is submitted in partial fulfillment of the requirements for the degree of

Master of Science (Computer Engineering)

---

Varundev Suresh Babu

This thesis has been read and approved by the Examining Committee:

---

Madhur Behl, Committee Chair

---

Zongli Lin, Adviser

---

Joanne Dugan, Adviser

Accepted for the School of Engineering and Applied Science:

---

Craig H. Benson, Dean, School of Engineering and Applied Science

May 2018

# Abstract

Many universities across the world are doing breakthrough research in the field of autonomous cars, but only a few universities have the financial and legal means to operate a full-scale autonomous car.

Some institutions have continued to research on autonomous cars using scaled platforms to varying degrees of success, but the question has always remained that these scaled platforms are limited in their capabilities thus limiting their research usefulness.

In this thesis, we present a case for implementing high level computer vision based navigation on scaled autonomous cars to prove that the scaled autonomous car platforms are an effective research tool.

*To family, teachers and friends*

# Acknowledgments

I thank my family for supporting me during all stages of my life and, especially during the past two years when I needed guidance in my academic career.

I thank Prof. Zongli Lin, Prof. Joanne Dugan and Prof. Madhur Behl for guiding me during the course of this project and, this project would have been significantly more difficult without their valuable suggestions and intellectual inputs.

Special thanks are due for the F1tenth.org team, both at the University of Virginia and the University of Pennsylvania for their work on the autonomous race-car platform which is the basis for this project.

I would finally like to thank the School of Engineering and, especially the LinkLab for allowing me to work on this project in their facilities.

Varundev Suresh Babu

University of Virginia

# Contents

# List of Figures

# Chapter 1

# Introduction

In the current world of academic research on autonomous vehicles, using a scaled system to replicate performance and dynamics has become a popular option for universities across the world [2, 3], including the Korean Institute of Advanced Science and Technology, the University of Tokyo, and the University of Western Australia. This revolution was started in North America at the Massachusetts Institute of Technology's RACECAR group and the University of Pennsylvania mLAB group, and both use similar approaches to develop the hardware part of this system, including the chassis, drive computer, and sensor suite; this approach will be referred to as the 'Racecar Approach'. One of the primary drawbacks of the Racecar Approach is that it relies upon the use of point-clouds, which are generated by a combination of LIDAR and depth-sensors. Our team has been developing a method to adapt the Racecar Approach in order to develop a monocular vision-based algorithm that helps the car navigate a test track, which is bounded by lane markings not unlike those found on highways. The goal of this thesis is to expand the capability of the Racecar Approach by developing a computer vision-based approach for understanding the environment instead of relying on sensors that provide only point-cloud information.This expansive method increases the intelligence of the car's system, which is able to derive more information from the environment instead of merely treating nearby objects as obstacles to avoid.

This thesis is divided into 3 parts: (1) an exploration of the current state of academic and industrial research, including shortcomings, (2) a proposed method and its advantages, and (3) the results and conclusions of our tests.

## 1.1 Motivation & Problem Statement

Autonomous vehicles, including self-driving cars, are the next frontier of academic & industrial research [4]. Companies like Alphabet's Waymo and Tesla have invested heavily in the field of autonomous vehicles hoping to become industry leaders in the near future. The natural question here is: why would well established corporations take risks amounting to several hundreds of billions of dollars on this technology now? A self-driving car is not a new concept, with first recorded attempts since early 20th century [1]. The very first automation is the invention of auto-ignition systems. Some of the early attempts at making an autonomous car included the invention of the automatic transmission and, cruise control, although serious attempts of realizing cruise control was made towards the later end of the 20th century. The definition of an autonomous car is an open ended question, and there are levels of autonomy (to be discussed later), but it is commonly agreed that a significant component of an autonomous car is its ability to navigate itself in a global frame of reference. The steady pace at which navigation aides have evolved helped propel this idea into reality, especially with the introduction of the satellite navigation systems that utilize the theory of relativity for accurate positioning of objects with reasonably finite dimensions. A major problem with satellite navigation is that it is primarily intended for ordnance delivery systems, and as such the controlling governing body will always provide highly accurate channels only to law enforcement agencies [5].

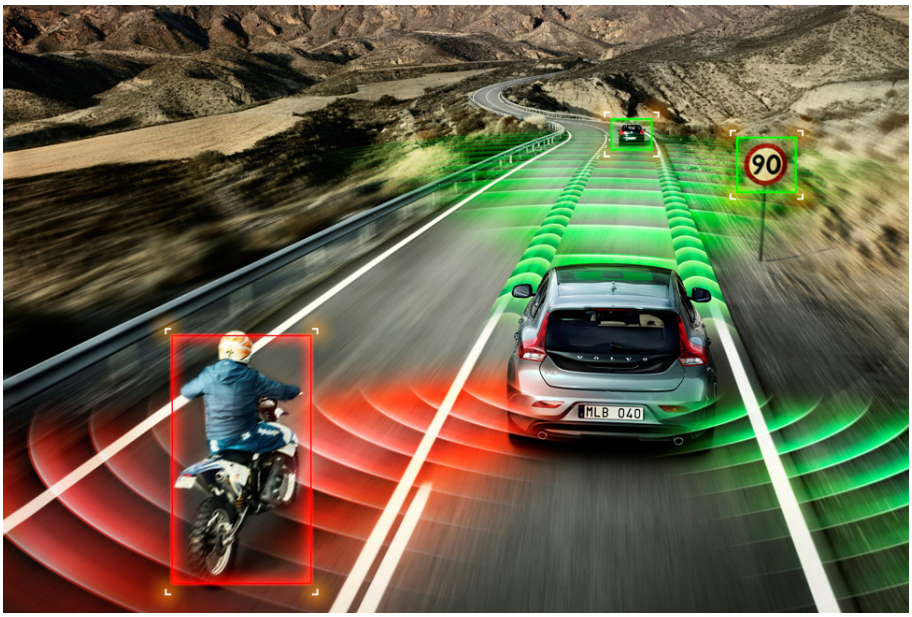


Figure 1.1: An autonomous car (green waves indicate safe travel zones, and red waves indicate danger)

The other major drawback of satellite navigation is that the guided vehicle has no additional information, especially about its surroundings. In technicality, this situation is called localization, and when technology

was made available for low level localization scanners, the early concepts of self-driving cars, especially those like Carnegie-Mellon's NavLab [6] provided in-depth perspective required to make progress towards a truly autonomous car.

Today, there exist mature technologies to help automate several subsystems in a car, like adaptive cruise control. The aim of our thesis project is to extend the capabilities of framework developed by the University of Pennsylvania's mLab scaled autonomous car by designing a single monocular camera-based navigation system that enables the car to travel in its lane, similar to a car driving on a highway. Our primary motivation to develop this system is to bridge the gap between a full scale self-driving car and the Racecar Framework to allow researchers across universities to test their algorithms in real life. While we agree agree that the thesis Racecar cannot be a replacement for a full scale self-driving car, our approach will enable other researchers in this field to obtain results with more reliability than that offered by pure simulation only results. The Racecear in our thesis project uses a single monocular camera as its only environment perception sensor. This single fixed Field of View (FOV) camera is always pointed forward like a dashboard camera and the underlying algorithm filters the image frame, identifies lane markings and requests the drive computer to navigate the track using PID control.


Figure 1.2: CMU NavLab 1: An early self-driving car

## 1.2 Current Systems & Their Shortcomings

At full scale, a system similar to the one described by the Racecar Approach is implemented in all of Tesla Motors production cars, but with many more cameras and other sensors. Some other companies, like Waymo and Uber depend heavily on a network of sensors to make the car aware of its surroundings. A method known as sensor fusion is applied to get the best possible information about the vehicle's surroundings and address the cost and technical trade-offs of individual sensors. Sensor fusion is combining sensory data or data derived from disparate sources such that the resulting fused information has less uncertainty than would be possible when these sources were used individually [7]. The term uncertainty reduction in this case can mean more accurate, more complete, or more dependable, or refer to the result of an emerging view, such as stereoscopic vision (calculation of depth information by combining two-dimensional images from two cameras at slightly different viewpoints). It is common to find most or all of the following sensors:

| Sensor | Data | Cost | Performance |
|---|---|---|---|
| RGB Camera | Images | Inexpensive | Unreliable during fog, mist & rain etc. |
| PIR Camera [8] | Heat signature | Expensive | Easily saturated |
| Lidar [9] | Point-cloud | Very Expensive | Highly dependent on lighting conditions |
| Radar | Point-cloud | Inexpensive | Unreliable in urban areas |
| RTK GPS [10] | Relative position | Expensive | Requires large open spaces |

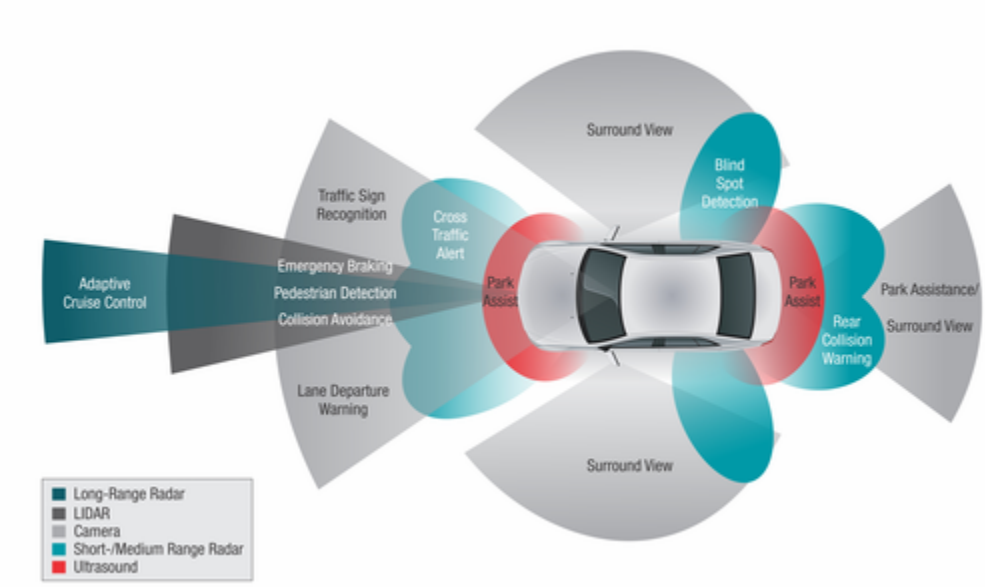


Figure 1.3: Sensor Fusion for an autonomous car

By carefully observing the characteristics of each type of sensor used in self-driving cars, it is evident that each type of sensor has its own set of advantages and disadvantages which is what originally led to the study on sensor fusion. Research on self-driving cars is contemporary and will continue for the foreseeable future as advances in sensor technology drives further advancements in safety and reliability of these machines [12].

One of the major drawbacks for universities is that self-driving car platforms are very expensive at full-scale to procure and government regulations prohibit live testing, forcing many researchers to depend on simulation for verifying their designs. While simulation tools are very robust, they can not accurately capture the dynamics of the system [13], nor the dynamics of the operating environment.

One way around this problem is to observe the system in a scaled environment with reasonable scaling factors that assist in accurately capturing the system's dynamics and enable research teams to test and verify their designs on a live model. While this cannot be considered a replacement for full scale testing, it is still more reliable that simulation alone. Work on the scaled platforms began in the Massachusetts Institute of Technology and spread throughout the universities in the world, and the design used in this thesis is partly inspired by the original work done by the MIT RACECAR group [2].

Some modification that can be seen in our system include the use of a flight controller instead of a generic AVR (an Atmel design) based micro-controller, and this allows use to reduce the additional hardware required to collect critical navigation data including odometry, pose and it has a very computationally efficient implementation of the Extended Kalman Filter (EKF) [14]. To reduce computational overhead on the Cortex co-processors on board, our model is designed to run entirely on the GPGPU.

Currently, most of the scaled experimental platforms at other institutions depend on a planar scanning LIDAR and/or a 3D point-cloud generator such as a stereo-camera. Their underlying software stack is designed to scan the environment to identify obstacles to avoid and, they employ different navigation algorithms to achieve this to varying levels of efficiency. By using the Racecar approach system out-of-the-box, and under default configuration, the current approach is to transfer data over the wireless network to a powerful desktop computer for computationally intensive processing, which then sends back the commands to the on-board computer for execution. This process can extremely inefficient because:

1. It requires near zero network transfer latency and high network bandwidth. Systems capable of supporting the network requirements usually communicate at 2.4GHz or 5GHz, and both these frequencies have poor penetration, thereby drastically reducing the range [15]. This, coupled with the FCC requirement to limit radiated power to less than 100mW produces a maximum range of 30 meters on most commercial network infrastructures.

2. Offloading computation is not scalable, because it implies that a computing cluster should provide up to 99.99% up-time, and at this scale, such reliability can not be offered, nor expected [16]. However, this is not an insurmountable problem for full scale cars which have more space, power and, control computers like the NVIDIA Pegasus [17].

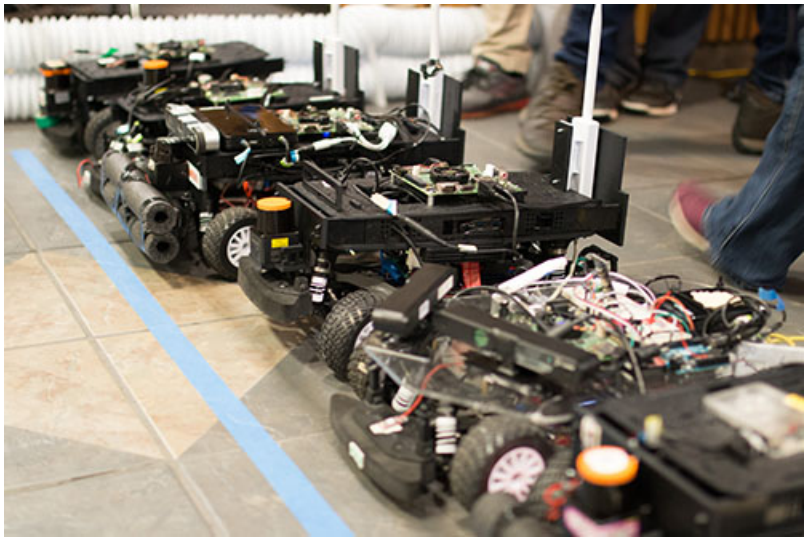

Figure 1.4: A few scaled autonomous car examples

## 1.3 Proposed Solution & Success Measures

In this thesis we propose a simpler solution to address the problems described at the end of the last section: Optimize the system to take full advantage of the on-board GPGPU (General Purpose Graphical Processing Unit) and test its performance by implementing a lane following scaled Radio Control (RC) car and compare it to the same problem implemented in a system that offloads computing over network and observe the following:

1. Subjectively distinguish the vehicle's ability to stay in the lane, especially at turns, to look for characteristics such as smoothness, PID behaviour and lane crossover rates. Ex: A good PID gain tuning will result in faster turns and quicker adjustments.

2. Compare the system performance as measured by resource utilization between a car that offloads computation to a remote computer against that of a car that performs the computation online using the Jetson and use available metrics to observe contrasting results.

# Chapter 2

# Experimental Setup

In this chapter, we explore the details of our hardware-based and software-based navigation stack. As explained in chapter 1, we borrow heavily from the University of Pennsylvania's design, with some critical changes to enable higher level of performance and reliability of the autonomous drive system, including the addition of a re-purposed and unconventional flight controller system and a software based telemetry, tracking and control (TTC) system.

## 2.1 The Hardware & It's Dynamics

The hardware used in this thesis is broadly divided into chassis system and the drive electronics. The chassis system includes the car itself and the drive electronics act as the implementation platform for the algorithms that enable the car to move autonomously. We chose a black-box design approach for modelling our entire hardware system where the various subsystems in our design architecture acted completely independent of each other and at the same time synchronized their operations to achieve a common goal.

The chassis is a generic 1:10 scale RC car, and we chose the Traxxas Rally car for the following advantages:

1. Single servo motor and single traction motor, thereby greatly simplifying the Input/Output (I/O) to two channels and a low latency control link to both actuators through a 22ms radio control system

2. Servo motor with self-contained Proportional-Derivative-Integral (PID) control for powering the Ackermann steering mechanism [18], and a high powered DC motor controller by an Electronic Speed Controller (ESC) that also provides DC power to the servo motor.

3. Four wheel drive setup provides precision control to even a heavy setup, with rapid positive and negative acceleration capabilities.

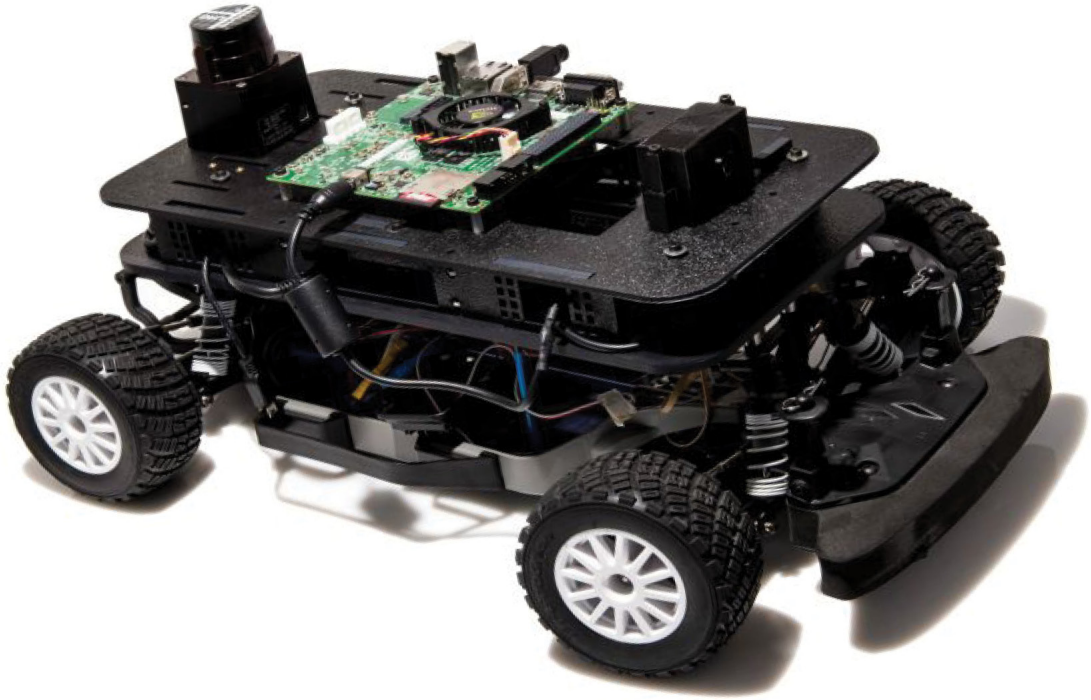

Figure 2.1: The RC Car chassis

The chassis is impressive in it's ability to robustly respond to commands, relatively independent of the terrain in which it is operating, but it was primarily intended for hobbyists who often push its physical abilities to the limits using the radio system provided. The remote control of the vehicle is dependent on a complex visual feedback of the operator, so issues such as obstacle avoidance, driving in a straight line and driving at a near-constant speed can be easily achieved if the vehicle's state is correctly estimated and/or recorded. To understand this better, imagine that the RC car is controlled manually by a human operator; the car's motion is refined by information the human operator gets by actively looking at the car (visual feedback) and predicting the trajectory of the car from the current position to the next position.

To obtain critical information about the vehicle at all times, we have used an open-source controller known as Pixhawk. Developed at the Swiss Federal Institute of Technology, Pixhawk is a mature adaptation of Ardupilot, an Arduino based autonomous vehicle controller. Pixhawk wins over Ardupilot in both hardware capability and its navigation software. The software stack, called PX4, rests on a Linux based Real Time Operating System (RTOS) and provides the following features:

| Component Base | Type | Function |
|---|---|---|
| MPU9250 | Inertial Sensor | Extended Kalman Filtering |
| PCA9685 | I2C Actuator | PWM State Machine |
| STM32F427 | Processor | I/O, Serial UART |
| ICM20608 | Inertial Sensor | Odometry |


Figure 2.2: The Pixhawk Controller

It should be noted here that the Extended Kalman Filtering (EKF) is preimplemented on the flight controller and is widely supported by the open-source community, so we did not have to work more on it to ensure operational reliability. The vehicle base responds to commands through messages passed using Pulse Width Modulation (PWM) [19]. PWM, in our context, is the art of encoding steering and throttle position, by mapping these values to the approximate duty cycle of the signal. If the signal duty cycle if 10%, this makes the traction motor turn at 10% of it's original power, or the servo motor to point at 18 degrees

from rest. An additional high accuracy inertial sensor is used to obtain linear and angular velocities and the system's odometry in all 3 dimensions, which is critical information for higher level control. The main processor which hosts the RTOS is also responsible for serial communications via the Universal Asynchronous Receiver Transmitter (UART) [20] modem through a serial cable or Universal Serial Bus (USB) cable.

The controller is connected to a more powerful on-board computer, the NVIDIA Jetson TK1. The Jetson is a single board computer with unique features including a 4-core Cortex A15 and 192-core NVIDIA Pascal GPU. By using GPU based libraries, this computer is capable of 160 Billion Floating Point Operations per Second (FLOPS) [21] with single precision. Coupled with DDR3 memory bandwidth, the Jetson is just capable enough of executing Computer Vision intensive tasks that we plan on implementing in this thesis.

The system is assembled using custom 3D printed support structures using PLA plastic that is light and strong enough for this task. The priority was given for protecting the on-board computer and the lithium batteries. The finalpart of the system is the telemetry and network infrastructure. We have used an Access Point (AP) router that provides wireless link between the on-board computer and the remote control computer.

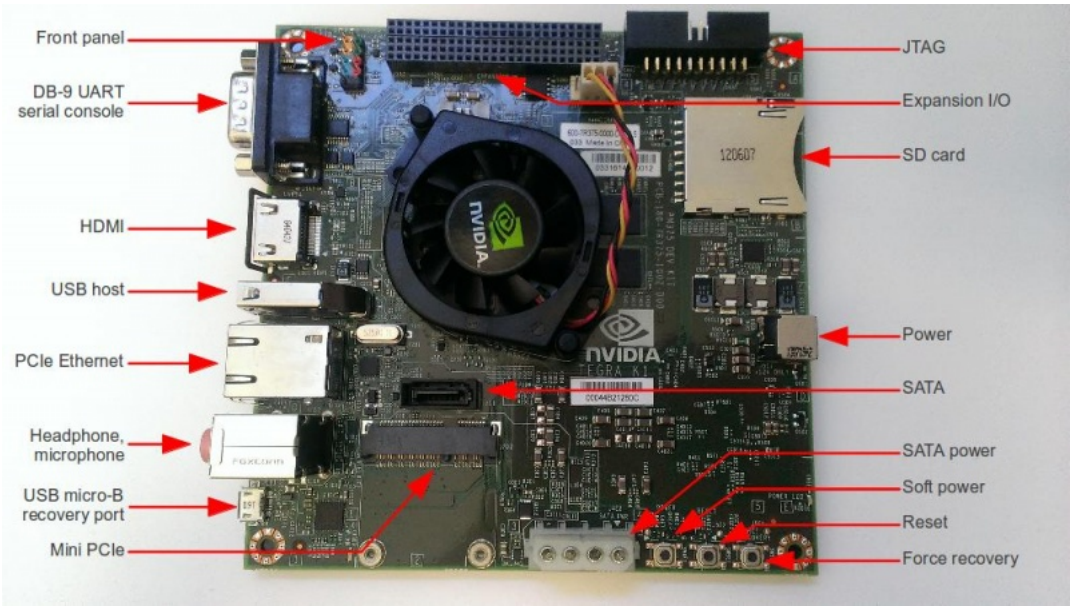


Figure 2.3: The NVIDIA Jetson TK1 control computer

## 2.2 ROS & The Software Stack

This project would have been exponentially more complex and difficult to implement without using the Robot Operating System (ROS) middleware. ROS is a collection of libraries that enable and simplify communications between processes and the physical robot [22]. The on-board Jetson natively supports Ubuntu 14.04 which is necessary for ROS Indigo Igloo, the ROS distribution that is necessary for us to proceed with this project. In addition to packages deployed by default while installing ROS, we have used the following additional packages:

1. ros-indigo-mavros [23]
2. ros-indigo-uvc_camera [24]
3. ros-indigo-ros-perception [25]

A brief introduction of how ROS works is necessary to completely understand the rationale for choosing this method. First, ROS, in a very simple way, acts as a communications handler that transports messages between processes and provides libraries and tools to help software developers create robot applications. It provides hardware abstraction, device drivers, libraries, visualizers, message-passing, package management, and more.

1. A node is a process that performs computation. Nodes are combined together into a graph and communicate with one another using streaming topics.
2. Topics are named buses over which nodes exchange messages. Topics have anonymous publish/subscribe semantics, which decouples the production of information from its consumption.
3. Nodes communicate with each other by publishing messages to topics. A message is a simple data structure, comprising typed fields including standard types (integer, floating point, boolean, etc.).
4. The ROS Master provides naming and registration services to the rest of the nodes in the ROS system. It tracks publishers and subscribers to topics as well as services.

Software in ROS is organized in packages. A package might contain ROS nodes, a ROS-independent library, a dataset, configuration files, a third-party piece of software, or anything else that logically constitutes a useful module. The goal of these packages it to provide this useful functionality in an easy-to-consume manner so that software can be easily reused. In general, ROS packages follow a "Goldilocks" principle:

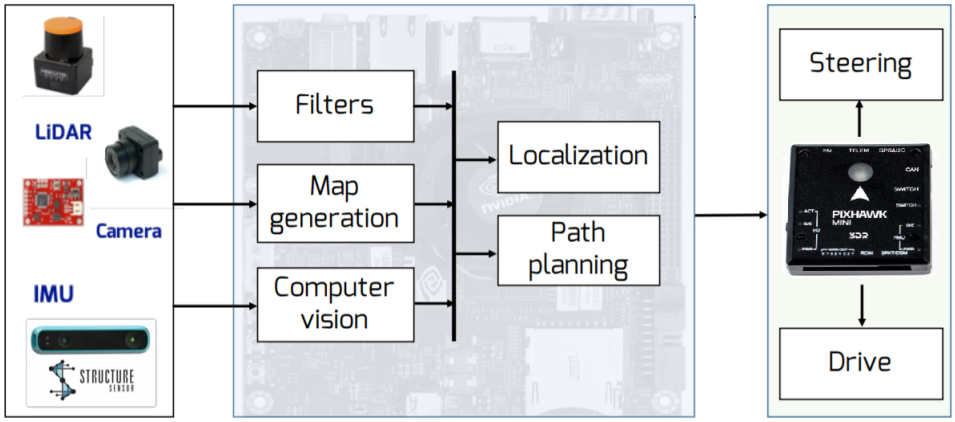


Figure 2.4: The general system architecture

enough functionality to be useful, but not too much that the package is heavyweight and difficult to use from other software.

Packages are easy to create by hand or with tools like catkin_create_pkg. A ROS package is simply a directory descended from ROS_PACKAGE_PATH that has a package.xml file in it. Packages are the most atomic unit of build and the unit of release. This means that a package is the smallest individual thing you can build in ROS and it is the way software is bundled for release (meaning, for example, there is one debian package for each ROS package), respectively.

ROS packages tend to follow a common structure. Here are some of the directories and files that are commonly used.

- include/package_name: C++ include headers (make sure to export in the CMakeLists.txt)
- msg/: Folder containing Message (msg) types
- src/package_name/: Source files, especially Python source that are exported to other packages.
- srv/: Folder containing Service (srv) types
- scripts/: executable scripts
- package.xml: Package catkin/package.xml
- CMakeLists.txt: CMake build file
- CHANGELOG.rst: Can be automatically injected into binary packaging

A brief explanation for the use of the additional packages is necessary here. The default ROS distribution is meant to support robot libraries that are most commonly used, but with the introduction of the Pixhawk controller and the monocular camera, we need to install additional libraries to support the new hardware.

Mavros is one of the most important additional library used in this thesis project. It is developed and maintained by the same community of developers that originally spearheaded research into the open-source autopilot system called Pixhawk. Mavros is a python and C based interface library that uses a fast transmission protocol call MAVlink to enable communications between the Jetson and the Pixhawk controller. Mavros publishes upto 22 different topics and 13 different nodes, but in the context of the thesis project, we are interested only in "actuator_control", "odom" and "imu/data".

"actuator_control" is a topic that uses the mavros message class ActuatorControl. ActuatorControl has three fields: (1) Header, which is a standard message (std_msg) type synchronization message, (2) control_group, which determines the type of vehicle the Pixhawk controller is attached to (in our case, the value is set to 0) and, (3) controls, which is a list of 8 elements representing individual control channels, each with floating values ranging from -1 to +1. Header is usually controlled by rosmaster, and control_group is set to a constant integer. Since the thesis project car needs only two channels, we chose to use channels 1 & 2 to control the car. For example, a value of -1 means full left steering or full reverse throttle and +1 means full right steering or full forward throttle.

"odom", or, odometry, is a geometry message type topic that uses inertial data to determine the current position of the car relative to its starting position and provides total distance travelled. In our thesis project, we are interested only in the latter. This topic is subscribed to only for battery resource management at a very high level. We have developed a process that uses odometry to determine range of travel, and this is especially important when the car is asked to undertake long distance missions. The process outputs the cars ability to undertake the current mission and stops the car if it determines that the current mission objectives can not be performed by the car.

"imu" is a node that publishes inertial measurement data to ROS and has two topics: "imu/data" & "imu/data_raw". We chose to subscribe only to "imu/data" because this topic is the output of the Pixhawks onboard EKF, and is more reliable than "imu/data_raw". The "imu/data" topic is used to obtain critical vehicle states, like its linear velocities and pose. This information is sent to the Jetsons PID steering and velocity control nodes for smooth driving behaviour.

The other library that we depend upon for our thesis project is "vision_OpenCV" which is a bridge between ROS and CV libraries. ROS passes around images in its own sensor_msgs/Image message format, but we want to use images in conjunction with OpenCV. CVBridge is a ROS library that provides an interface between ROS and OpenCV. CvBridge can be found in the cv_bridge package in the vision_opencv stack.

## 2.3 Test Track & Other Parameters

As described in chapter 1, the thesis project car is designed to bridge the capability gap between full scale autonomous car and the Racecar Approach. In contemporary research, the most popular test scenario is a highway. Our thesis project was tested in a single lane road setup in the LinkLab at the University of Virginia. We created this setup under the following considerations.

- The lane width is scaled by the same factor as the RC car is scaled from a full scale car.
- The track produces just enough Gaussian noise as does a real road in a rural environment.
- Some sections of track turn at much tighter angles, while some others turn a lower angles.

In order to understand why these considerations are made, we request the reader to imagine driving a car on a highway through the country side. Depending on the traffic, this scenario is the least overwhelming for the human driver as compared to driving in a town or a metropolis. We have included a set of images that describe the track and, this will educate the reader about the track layout.

The track has a length of about 160 feet. It is bounded by 2 lanes of blue color and on a grey coloured surface. The filtering mechanism that extracts lane information is documented later in chapter 3.

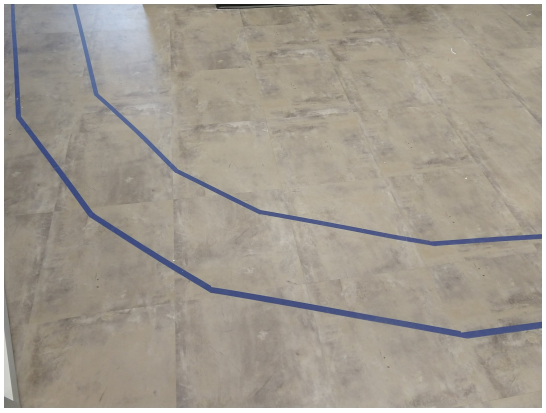

Figure 2.5: Test Track: Sample 1

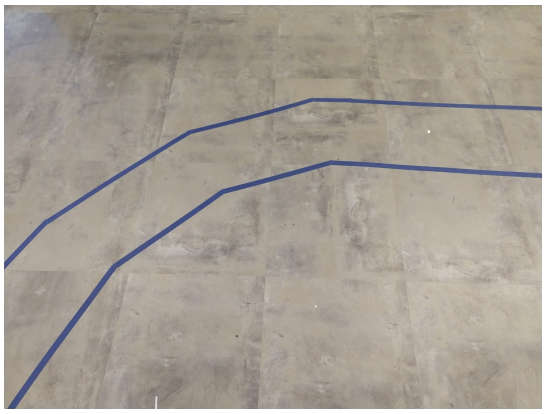

Figure 2.6: Test Track: Sample 2

Figure 2.7: Test Track: Sample 3

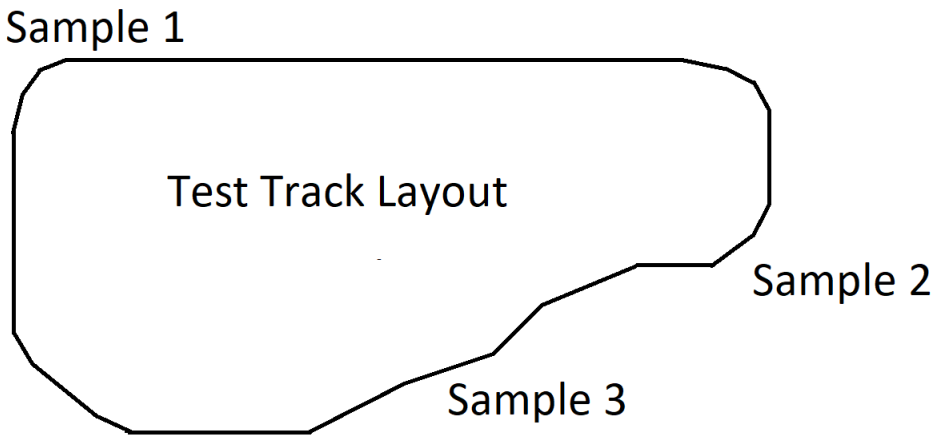


Figure 2.8: Test Track schematic layout

# Chapter 3

# Design, Implementation & Testing

In chapter 3, we describe the algorithms used by the thesis project car to perceive, plan and navigate in the test track. Broadly speaking, we have implemented the computationally intensive computer vision task on the Jetson and the control algorithms have been implemented on the Pixhawk controller. We have designed the system in this manner because of the relative strengths of the Jetson and the Pixhawk in different environments. We have described these strengths in detail in chapter 2.

## 3.1 System Architecture: Computer Vision

When we drive, we use our eyes to decide where to go. The lines on the road that delineate the lanes act as our constant reference for where to steer the vehicle. Naturally, one of the first things we would like to do in developing a self-driving car is to automatically detect lane lines using an algorithm. We want to start with an image like figure 3.1, process the image for lane detection like figure 3.2 and finally extrapolate and average those lines for a smooth lane detection feature which we can apply to video frames like figure 3.3. [26]

Figure 3.1: Raw image

Figure 3.2: Lines drawn from Hough space conversion

The first step to working with our images will be to convert them to grayscale. This is a critical step to using the Canny Edge Detector inside of OpenCV. we will talk more about what canny() does in in the next step, but right now its important to realize that we are collapsing 3 channels of pixel value (Red, Green, and

Figure 3.3: Final image

Blue) into a single channel with a pixel value range of [0,255]. Before we can detect our edges, we need to make it clear exactly what were looking for. Lane lines are always either yellow or white in the United States. Yellow can be a tricky color to isolate in RGB space, so lets convert instead to Hue Value Saturation or HSV color space. The target range for yellow values we used are below. Next, we will apply a mask to the original RGB image to return the pixels were interested in. [27]

```
lower_yellow = np.array([20, 100, 100], dtype = "uint8")
upper_yellow = np.array([30, 255, 255], dtype="uint8")
mask_yellow = cv2.inRange(img_hsv, lower_yellow, upper_yellow)
mask_white = cv2.inRange(gray_image, 200, 255)
mask_yw = cv2.bitwise_or(mask_white, mask_yellow)
mask_yw_image = cv2.bitwise_and(gray_image, mask_yw)
```

Now we apply a quick Gaussian blur. This filter will help to suppress noise in our Canny Edge Detection by averaging out the pixel values in a neighborhood.

```
kernel_size = 5
gauss_gray = gaussian_blur(mask_yw_image,kernel_size)
```

Now we compute our Canny Edge Detection. Basically, canny() parses the pixel values according to their directional derivative (i.e. gradient). Whats left over are the edges or where there is a steep derivative in at least one direction. We will need to supply thresholds for canny() as it computes the gradient. John Canny himself recommended a low to high threshold ratio of 1:2 or 1:3 [28]. In brief, the Process of Canny edge detection algorithm can be broken down to 5 different steps

Figure 3.4: BGR to Grayscale conversion

Figure 3.5: Grayscale image with all pixels that are not yellow or white set to black/zero

- Apply Gaussian filter to smooth the image in order to remove the noise
- Find the intensity gradients of the image
- Apply non-maximum suppression to get rid of spurious response to edge detection
- Apply double threshold to determine potential edges
- Track edge by hysteresis: Finalize the detection of edges by suppressing all the other edges that are weak and not connected to strong edges.

```
low_threshold = 50
high_threshold = 150
canny_edges = canny(gauss_gray,low_threshold,high_threshold)
```

We dont want our car to be paying attention to anything on the horizon, or even in the other lane. Our lane detection pipeline should focus on whats in front of the car. Do do that, we are going to create another mask called our region of interest (ROI) in our (human) perspective [29]. Everything outside of the ROI will be set to black/zero, so we are only working with the relevant edges.

Figure 3.6: Canny edge detection

```
roi_image = region_of_interest(canny_edges, vertices)
```

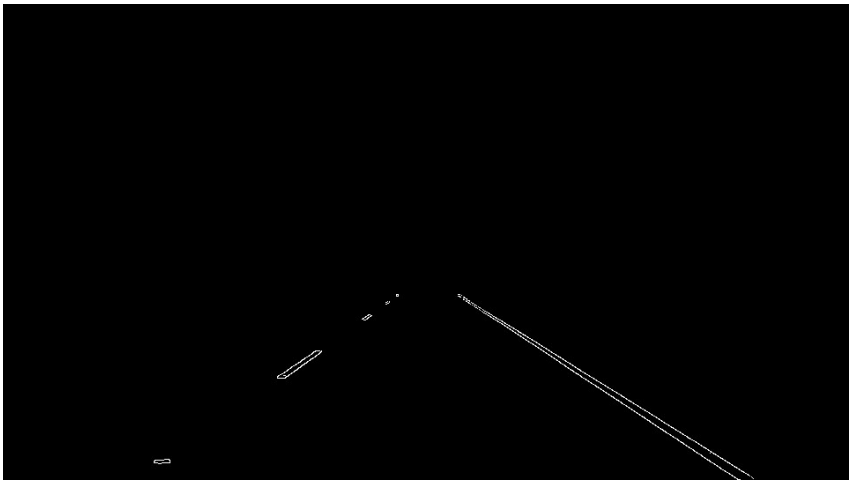

Figure 3.7: The ROI for lane detection

When images are to be used in different areas of image analysis such as object recognition, it is important to reduce the amount of data in the image while preserving the important, characteristic, structural information. Edge detection makes it possible to reduce the amount of data in an image considerably. However the output from an edge detector is still an image described by its pixels. If lines, ellipses and so forth could be defined by their characteristic equations, the amount of data would be reduced even more. The Hough transform

was originally developed to recognize lines, and has later been generalized to cover arbitrary shapes. We use Hough Transform [30] in the next stage under the following assumptions:

- Pixels are considered points in XY space
- hough_lines() transforms these points into lines inside of Hough space
- Wherever these lines intersect, there is a point of intersection in Hough space
- The point of intersection corresponds to a line in XY space

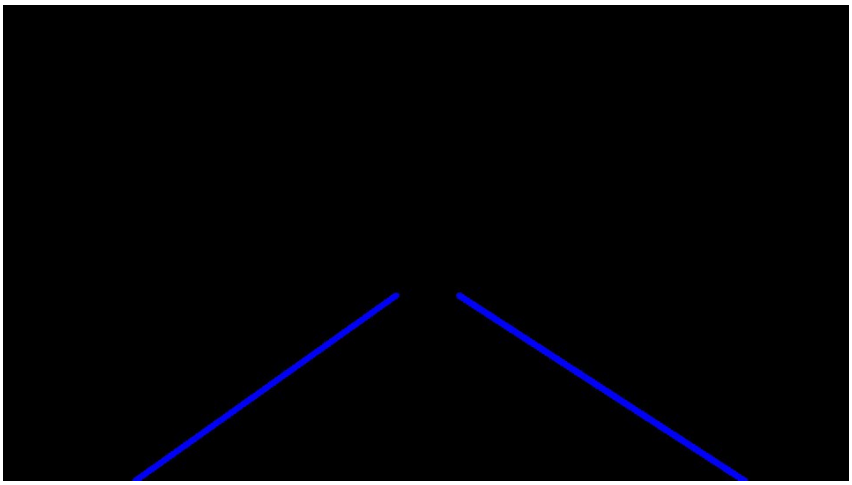

Figure 3.8: Lines from Hough transform

The key observation about the image above is that it contains zero pixel data from any of the photos we processed to create it. It is strictly black/zeros and the drawn lines. Also, what looks like simply two lines can actually be a multitude. In Hough space, there could have been many, many points of intersection that represnted lines in XY. We will want to combine all of these lines into two master averages. Once we have our two master lines, we can average our line image with the original, unaltered image of the road to have a nice, smooth overlay.

```
complete = cv2.addWeighted(initial_img, alpha, line_image, beta, lambda)
```

In short, here is the working model pipeline for lane tracking employed in our thesis project.

```
def image_pipeline(self, data):
        try:
            cv_image = self.bridge.imgmsg_to_cv2(data, "bgr8")
        except CvBridgeError as e:
```

Figure 3.9: Result of First Stage

```
    rospy.logerr(e)
# Inverse Perspective Mapping
self.ipm.initializeTransformationMatrix(cv_image)
warped = self.ipm.warp(cv_image)
# crop
cropped = self.img_prep.crop(warped, self.above_value, self.below_value, self.side_value)
# grayscale
gray = self.img_prep.grayscale(cropped)
# blur
blurred = self.img_prep.blur(gray, (self.deviation, self.deviation), self.border)
# canny
canny = self.img_prep.edge_detection(blurred, self.threshold_low, self.threshold_high, self.ape
heigth, width = canny.shape
if width == 133:  # NOTE: This parameter is valid only for UVA LinkLab HW Lab
    cv2.line(canny, (0, 4/2), (18/2, heigth), (0, 0, 0), 2)
    cv2.line(canny, (width, 4/2), (width - 18/2, heigth), (0, 0, 0), 2)
else:
    cv2.line(canny, (0, 4), (18, heigth), (0, 0, 0), 2)
    cv2.line(canny, (width, 4), (width - 18, heigth), (0, 0, 0), 2)
# Lane Detection
canny = cv2.cvtColor(canny, cv2.COLOR_GRAY2BGR)
```

```
self.lane_model.update_segments(canny.copy())
self.lane_model.draw_segments(canny)
state_point_x = self.lane_model.state_point_x()
```

Once the computer vision node has completed processing the current video frame, it sends the following tracking values to the Pixhawk controller through mavros node:

- left lane track distance from lower left corner
- right lane track distance from lower left corner
- geometric centre of lane calculated from the previous two values
- vehicle's current center

These values are used by the mavros node to calculate the deviation of the vehicle in the current frame and produce error correction values so that the vehicle returns to the center.

## 3.2 System Architecture: Command & Control

The thesis project car is modelled around a servo mechanism [32] for both steering and throttle. This is a common actuator in a control system. It directly provides rotary motion and, coupled with wheels or drums and cables, can provide translational motion. The electric equivalent circuit of the armature and the free-body diagram of the rotor are shown in the following figure.

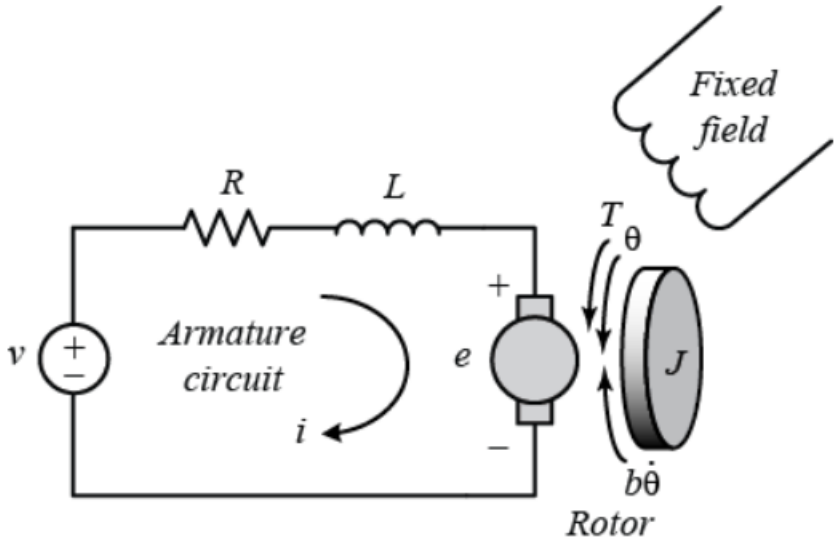


Figure 3.10: Free-body diagram of Servo Motor

In general, the torque generated by a DC motor is proportional to the armature current and the strength of the magnetic field, which in our project is controlled using a PWM pulse with duty cycle varying between 1000 microseconds and 2000 microseconds. The dynamic equations of the servo mechanism in state-space form are given below.

$$\frac{d}{dt}\begin{bmatrix} \theta \\ \dot{\theta} \\ i \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & -\frac{b}{J} & \frac{K}{J} \\ 0 & -\frac{K}{L} & -\frac{R}{L} \end{bmatrix} \begin{bmatrix} \theta \\ \dot{\theta} \\ i \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ \frac{1}{L} \end{bmatrix} V$$

$$y = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} \theta \\ \dot{\theta} \\ i \end{bmatrix}$$

In control engineering, a state-space representation is a mathematical model of a physical system as a set of input, output and state variables related by first-order differential equations or difference equations. State variables are variables whose values evolve through time in a way that depends on the values they have at any given time and also depends on the externally imposed values of input variables. Output variables values depend on the values of the state variables. The "state space" is the Euclidean space in which the variables on the axes are the state variables [31]. The state of the system can be represented as a vector within that space. The above equations are of the form of standard state-space equations as described below.

$$\dot{\mathbf{x}} = A\mathbf{x} + Bu$$

$$y = C\mathbf{x}$$

With a 1-radian step reference, the design criteria are the following.

- Settling time less than 0.05 seconds
- Overshoot less than 20%
- No steady-state error, even in the presence of a step disturbance input

In control theory the settling time of a dynamical system such as an amplifier or other output device is the time elapsed from the application of an ideal instantaneous step input to the time at which the amplifier output has entered and remained within a specified error band. Overshoot refers to an output exceeding its final, steady-state value. Steady-state error is defined as the difference between the input (command) and the output of a system in the limit as time goes to infinity (i.e. when the response has reached steady state).

Since all of the state variables in our problem are very easy to measure (simply add an ammeter for current, a tachometer for speed, and a potentiometer for position), we can design a full-state feedback controller for the system without worrying about having to add an observer. The control law for a full-state feedback system has the form $u = r - K_c$. The associated block diagram is given in Fig 3.11.

Recall that the characteristic polynomial for this closed-loop system is the determinant of $sI - (A - BK_c)$ where $s$ is the Laplace variable. Since the matrices $A$ and $BK_c$ are both 3x3 matrices, there should be 3

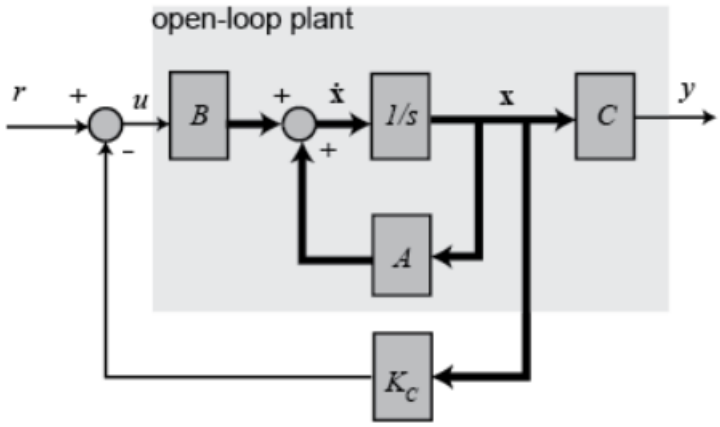


Figure 3.11: Control Block Diagram of Servo Mechanism

poles for the system. This fact can be verified with the MATLAB command order. If the given system is controllable, then by designing a full state-feedback controller we can move these three poles anywhere we'd like. Whether the given system is controllable or not can be determined by checking the rank of the controllability matrix $[B\ AB\ A^2B\ \ldots]$. The MATLAB command ctrb constructs the controllability matrix given $A$ and $B$. Additionally, the command rank determines the rank of a given matrix, though it can be numerically unreliable. Therefore, we will use the command det to calculate the determinant of the controllability matrix where a full rank matrix has a non-zero determinant.

From the above, we know that our system is controllable since the determinant of the controllability matrix is not zero and hence we can place the system's closed-loop poles anywhere in the s-plane. We will first place the poles at -200, -100+100i and -100-100i. By ignoring the effect of the first pole (since it is faster than the other two poles), the dominant poles correspond to a second-order system with $\zeta = 0.5$ corresponding to 20% overshoot and $\sigma = 100$ which corresponds to a settling time of 0.05 seconds. Once we have determined the pole locations we desire, we can use the MATLAB commands place or acker to determine the controller gain matrix, $K_c$, to achieve these poles. We will use the command place since it is numerically better conditioned than 'acker'. However, if we wished to place a pole with multiplicity greater than the rank of the matrix $B$, then we would have to use the command 'acker'.

Referring back to the equations and schematic at the top of the page, we see that employing a state-feedback law $u = r - K_c\mathbf{x}$, the state-space equations become the following.

$$\dot{\mathbf{x}} = (A - BK_c)\mathbf{x} + Br$$

$$y = C\mathbf{x}$$

After incorporating this, we test for the system's step response, shown in figure 3.12.

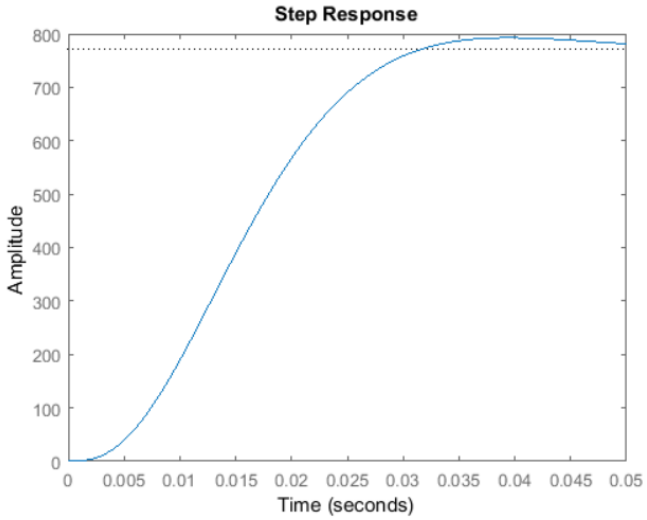


Figure 3.12: Step response of the controller

Note that our given requirements are not met, specifically, the steady-state error is much too large. Before we address this, let's first look at the system's disturbance response. In order to observe the system's disturbance response, we must provide the proper input to the system. In this case, a disturbance is physically a load torque that acts on the inertia of the motor. This load torque acts as an additive term in the second state equation (which gets divided by $J$, as do all the other terms in this equation). We can simulate this simply by modifying our closed-loop input matrix, $B$, to have a $1/J$ in the second row assuming that our current input is only the disturbance. The step response of the system to disturbance is given in Fig 3.13.

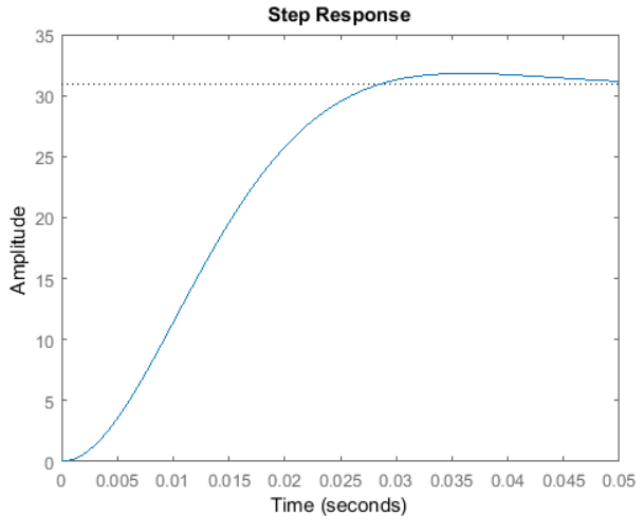


Figure 3.13: Step response of the controller with disturbance

Notice that the error due to the step disturbance is non-zero. Therefore, this will also need to be

compensated for. We know that if we put an extra integrator in series with the plant it can remove the steady-state error due to a step reference. If the integrator comes before the injection of the disturbance, it will also cancel a step disturbance input in steady state. This changes our control structure so that it now resembles the block diagram shown in the following figure 3.14.

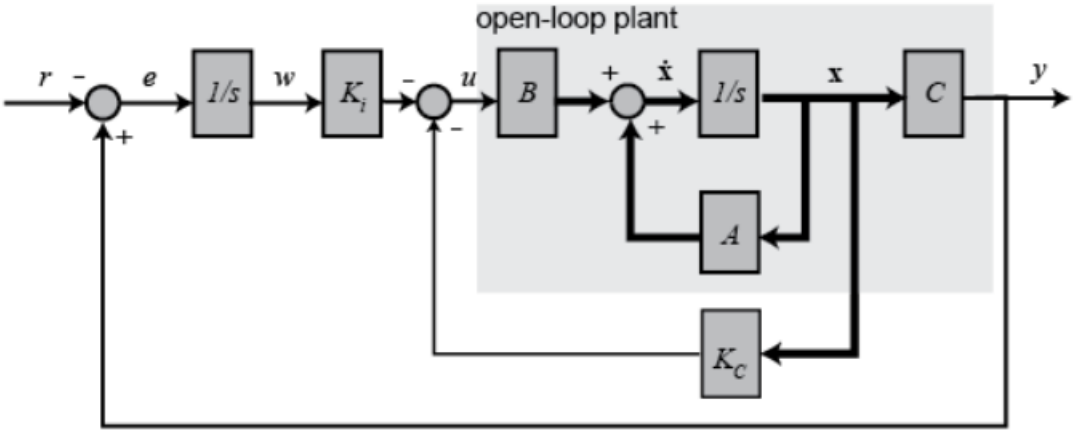


Figure 3.14: Control Block Diagram of Servo Mechanism with Integrator

We can model the addition of this integrator by augmenting our state equations with an extra state for the integral of the error which we will identify with the variable $w$. This adds an extra state equation, where the derivative of this state is then just the error, $e = y - r$ where $y = \theta$. This equation will be placed at the bottom of our matrices. The reference $r$, therefore, now appears as an additional input to our system. The output of the system remains the same.

$$\frac{d}{dt}\begin{bmatrix} \theta \\ \dot{\theta} \\ i \\ w \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & -\frac{b}{J} & \frac{K}{J} & 0 \\ 0 & -\frac{K}{L} & -\frac{R}{L} & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \theta \\ \dot{\theta} \\ i \\ w \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ \frac{1}{L} \\ 0 \end{bmatrix} V + \begin{bmatrix} 0 \\ 0 \\ 0 \\ -1 \end{bmatrix} r$$

$$y = \begin{bmatrix} 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \theta \\ \dot{\theta} \\ i \\ w \end{bmatrix}$$

These equations represent the dynamics of the system before the loop is closed. We will refer to the system matrices in this equation that are augmented with the additional integrator state as $A_a$, $B_a$, $C_a$, and $D_a$. The vector multiplying the reference input $r$ will be referred to as $B_r$. We will refer to the state vector of the augmented system as $\mathbf{x_a}$. Note that the reference, $r$, does not affect the states (except the integrator state) or the output of the plant. This is expected since there is no path from the reference to the plant input, $u$, without implementing the state-feedback gain matrix $K_c$.

In order to find the closed-loop equations, we have to look at how the input, $u$, affects the plant. In this case, it affects the system in exactly the same manner as in the unaugmented equations except now $u = -K_c\mathbf{x} - K_i w$. We can also rewrite this in terms of our augmented state as $u = -K_a\mathbf{x_a}$ where $K_a = [K_c \ K_i]$. Substituting this $u$ into the equations above provides the following closed-loop equations.

$$\dot{\mathbf{x}}_a = (A_a - B_a K_a)\mathbf{x}_a + B_r r$$

$$y = C_a \mathbf{x}_a$$

In the above, the integral of the error will be fed back, and will result in the steady-state error being reduced to zero. Now we must redesign our controller to account for the augmented state vector. Since we need to place each pole of the system, we will place the pole associated with the additional integrator state at -300, which will be faster than the other poles. We can see that all of the design specifications are close to being met by this controller. The settle time may be a little large, but by placing the closed-loop poles a little farther to the left in the complex s-plane, this requirement can also be met. The step response of the system is shown in Fig 3.15.

We have implemented this controller in the Pixhawk, which accepts reference inputs as either steering angle or throttle position, and drives the system from the current state to the requested state.

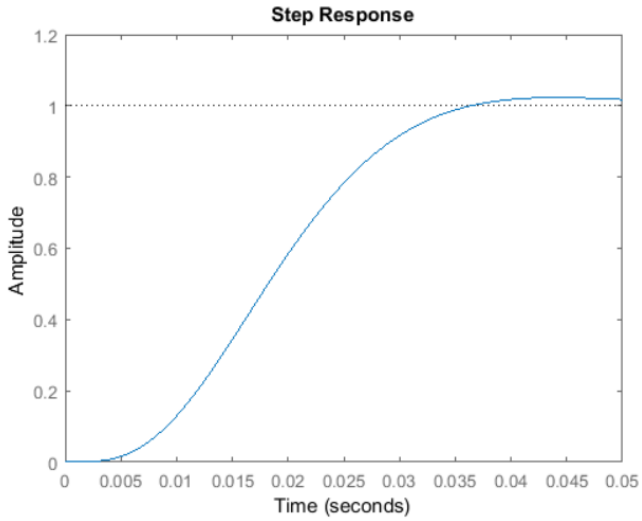


Figure 3.15: Step response of the controller with Integrator

# Chapter 4

# Results & Conclusion

## 4.1 Results

ROS has some powerful data visualizations tools that capture the system dynamics and present it to the user for analysis. In this thesis, we used two such tools: rosbag and rviz.

'rosbag' is used to collect some or all data from the thesis car and store it on the Jetson and this data is transferred in a file format known as 'bag' to another computer for analyzing the data. We will explain why the data transfer process is necessary later in this section. 'rosbag' is a set of tools for recording from and playing back to ROS topics. It is intended to be high performance and avoids deserialization and reserialization of the messages. In Fig 4.1, we have provided a screen-shot of how a bag file is visualized. While topics that carry image data are processed using the ROS_openCV bridge, topics relating to the Pixhawk controller are processed as standard data by ROS environment. We are interested in two main topics here: ActuatorControl and IMU; and, in continuation of chapter 2, we have included the description of the messages here.

Figures 4.4 to 4.6 describe the system behaviour on the test-track. The viewing window presents us with the Racecar camera's view, and above it, we can see the lane tracking window. In the tracking window, the white lines represent the lane (and any other body with straight dimensions in the viewing window), and the two green circles represent how the algorithm tracks the left and the right lane. The centre red line is the geometric centre of the lane produced after an inverse perspective transform and the red circle shows the Raecar's current steering angle. The red circle tracks the red line using a PID control loop, whose gains are tuned using a reinforcement learning function that has been pre-implemented in the Pixhawk controller.

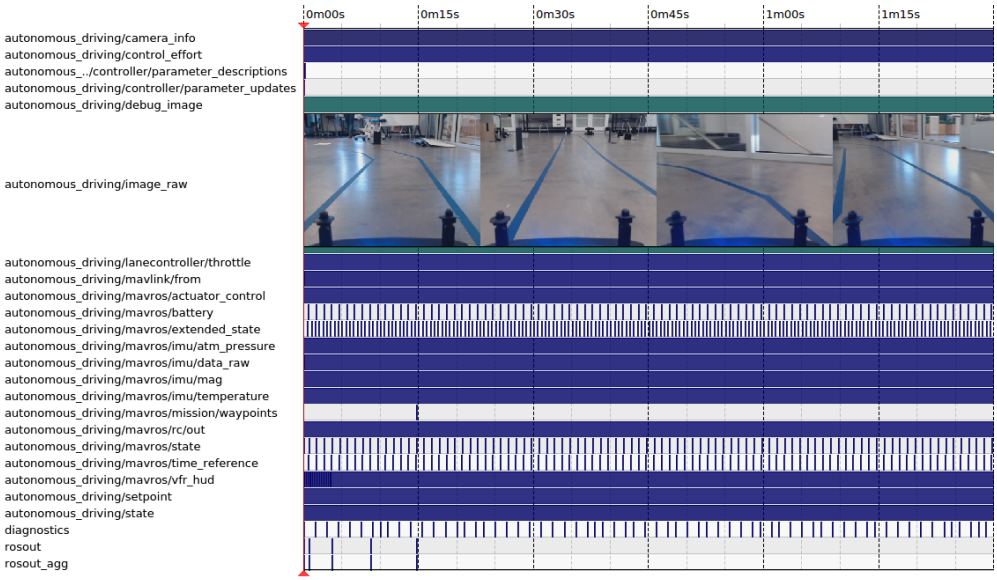


Figure 4.1: Bag File data from ROS

**File: mavros_msgs/ActuatorControl.msg**

**Raw Message Definition**

```
# raw servo values for direct actuator controls
#
# about groups, mixing and channels:
# https://pixhawk.org/dev/mixing

# constant for mixer group
uint8 PX4_MIX_FLIGHT_CONTROL = 0
uint8 PX4_MIX_FLIGHT_CONTROL_VTOL_ALT = 1
uint8 PX4_MIX_PAYLOAD = 2
uint8 PX4_MIX_MANUAL_PASSTHROUGH = 3
#uint8 PX4_MIX_FC_MC_VIRT = 4
#uint8 PX4_MIX_FC_FW_VIRT = 5

std_msgs/Header header
uint8 group_mix
float32[8] controls
```

**Compact Message Definition**

```
uint8 PX4_MIX_FLIGHT_CONTROL=0
uint8 PX4_MIX_FLIGHT_CONTROL_VTOL_ALT=1
uint8 PX4_MIX_PAYLOAD=2
uint8 PX4_MIX_MANUAL_PASSTHROUGH=3
std_msgs/Header header
uint8 group_mix
float32[8] controls
```

Figure 4.2: ActuatorControl message description

We use another ROS visualization tool called 'rviz' to visualize the viewing window data. 'rviz' cannot be used to record data and, it has serious network bandwidth limitations which makes it difficult to get data in real time. Hence, we use 'rosbag' instead.

**File: sensor_msgs/Imu.msg**

## Raw Message Definition

```
# This is a message to hold data from an IMU (Inertial Measurement Unit)
#
# Accelerations should be in m/s^2 (not in g's), and rotational velocity should be in rad/sec
#
# If the covariance of the measurement is known, it should be filled in (if all you know is the
# variance of each measurement, e.g. from the datasheet, just put those along the diagonal)
# A covariance matrix of all zeros will be interpreted as "covariance unknown", and to use the
# data a covariance will have to be assumed or gotten from some other source
#
# If you have no estimate for one of the data elements (e.g. your IMU doesn't produce an orientation
# estimate), please set element 0 of the associated covariance matrix to -1
# If you are interpreting this message, please check for a value of -1 in the first element of each
# covariance matrix, and disregard the associated estimate.

Header header

geometry_msgs/Quaternion orientation
float64[9] orientation_covariance # Row major about x, y, z axes

geometry_msgs/Vector3 angular_velocity
float64[9] angular_velocity_covariance # Row major about x, y, z axes

geometry_msgs/Vector3 linear_acceleration
float64[9] linear_acceleration_covariance # Row major x, y z
```

## Compact Message Definition

```
std_msgs/Header header
geometry_msgs/Quaternion orientation
float64[9] orientation_covariance
geometry_msgs/Vector3 angular_velocity
float64[9] angular_velocity_covariance
geometry_msgs/Vector3 linear_acceleration
float64[9] linear_acceleration_covariance
```

Figure 4.3: IMU message description

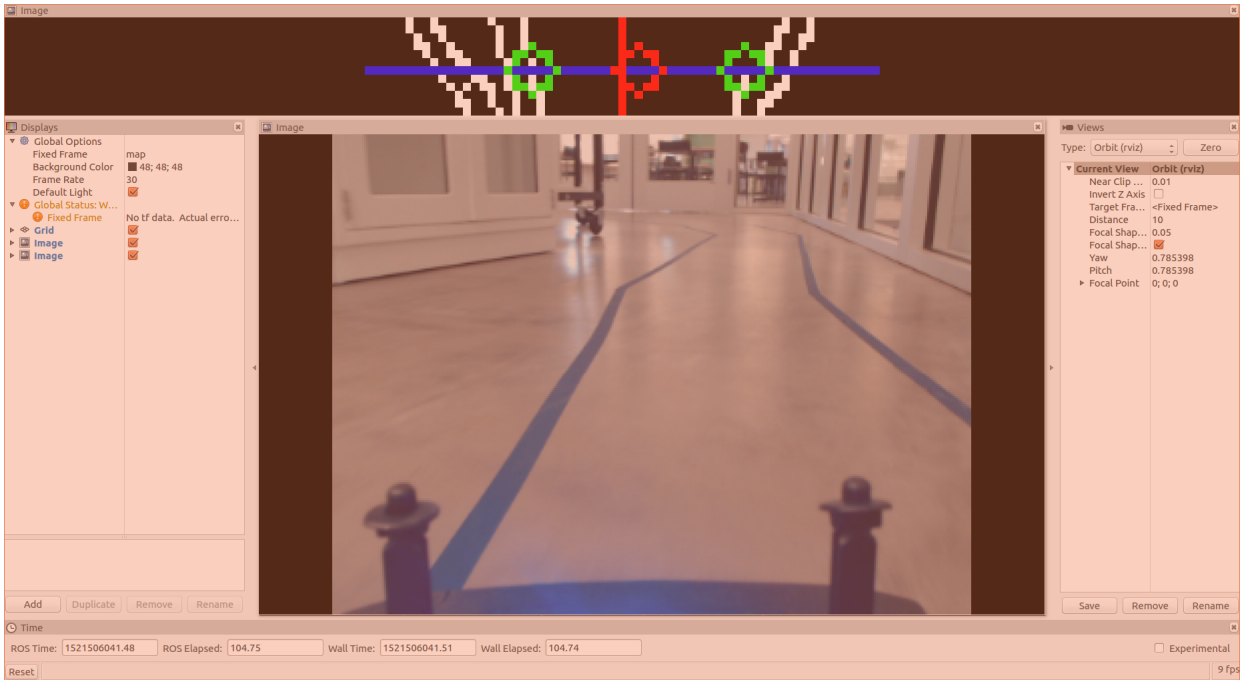


Figure 4.4: Viewing window on straight line: Sample 1

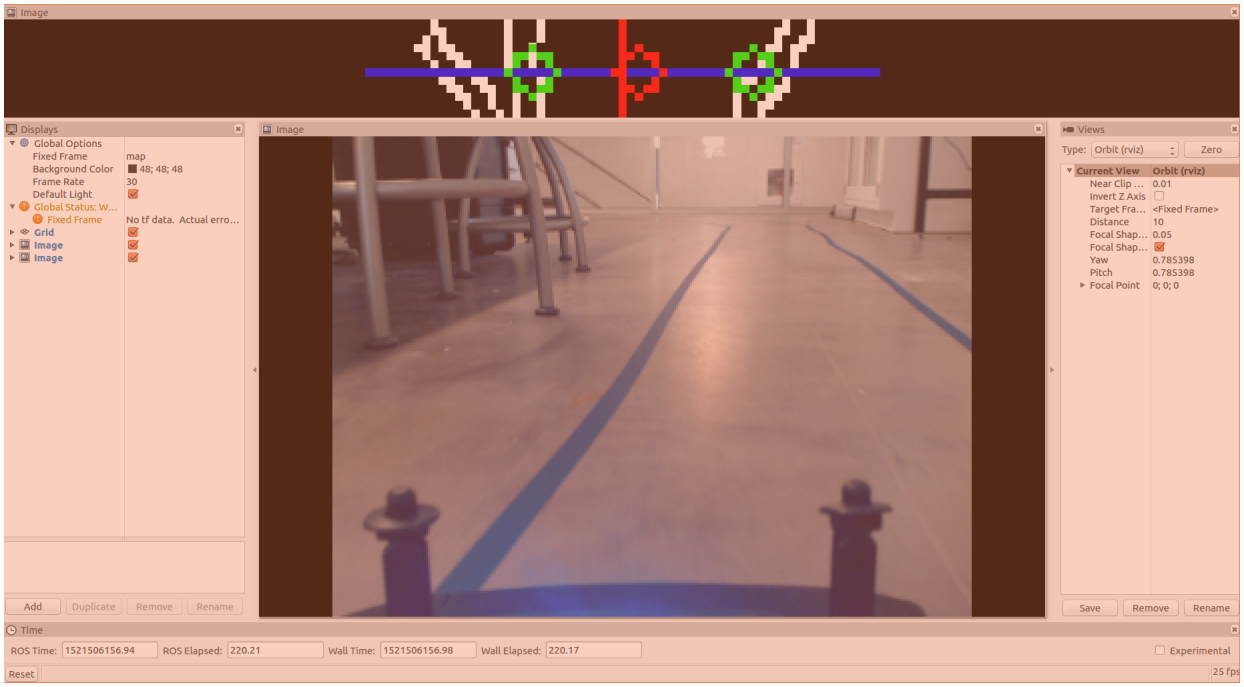


Figure 4.5: Viewing window on straight line: Sample 2

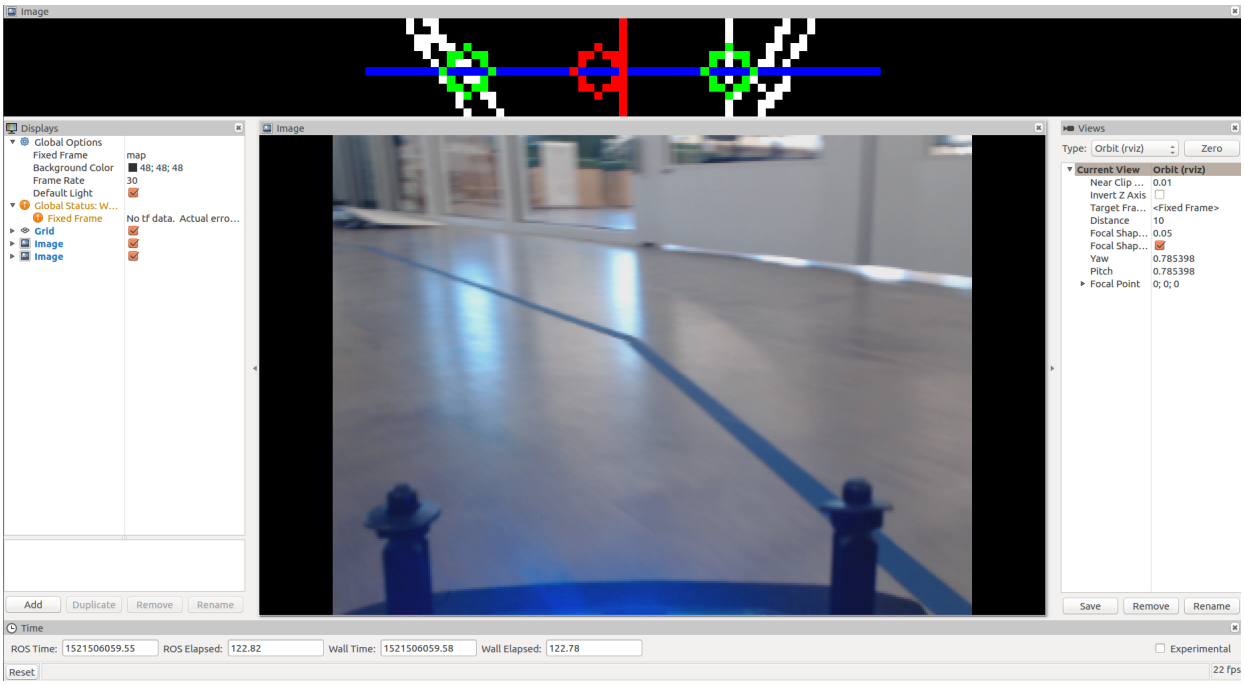


Figure 4.6: Viewing window at a curve

The final results for the success metrics are subjective, because the Racecar's erratic behaviour did not let us extract enough data to determine system performance. However, the following table represents comments from people (some with technical background, and some with non-technical background) invited to judge the performance of the vehicle:

| Metric/Percentage | Smooth PID | Corner Handling | Disturbance Rejection |
|---|---|---|---|
| Poor Behaviour | 0 | 0 | 0 |
| Average Behaviour | 0 | 50 | 30 |
| Good Behaviour | 60 | 50 | 70 |
| Excllent Behaviour | 40 | 0 | 0 |

Similarly, the vehicle's navigation performance as a function of distance from the remote computer 'Vehicle 1' (in meters), compared to that of a vehicle performing all the computation on board 'Vehicle 2':

| Distance | 30mts | 50mts | 100mts |
|---|---|---|---|
| Behaviour | stable | stable | marginally unstable |

Where 'stable' indicates that Vehicle 1 performance is comparable to that of Vehicle 2, and 'marginally unstable' indicates that Vehicle 1 network is congested with either high latency, full network buffer or radio range decay. In near future, we plan on testing the vehicle at much larger distances so that we can obtain a characteristic curve of performance vs range.

With the data from the two tables above, we have come to the conclusion:

- The thesis project car with the on-board computer outperforms the car with a networked computer:
  - Subjectively; with better handling in both corners and straight lanes,
  - Objectively; with faster response times to changes in track layout and, using at most 60 percent of computational resource compared to the networked car
- The networked car requires frequent PID tuning for changes in the car's state; including battery reserves, weight, distance from the remote computer etc., but the car with the on-board computer is able to run the entire course of the battery with minimal changes (and, sometimes, no changes) to the PID gains. This indicates that the car with the on-board computer is more robust to environmental changes as compared to the networked car.

## 4.2 Conclusion & Future Work

The lane detection techniques play a significant role in intelligent transport systems. In this thesis project, an OpenCV based lane detection method has been studied. It has resulted in a higher than average rate of lane enforcement than traditional lane enforcement methods that use dual camera based PID lane control. Therefore, further improvements can be done to enhance the results. In the near future, one can modify the existing image pipeline so that it can measure both the curved and straight roads more accurately. Various steps should be taken to improve the results in different environmental conditions like sunny day, foggy day, rainy day etc.

With respect to the thesis car itself, we plan on further modifying the software base to implement machine learning for coordinated vehicular navigation with two or more other similar cars. We also plan on modifying the hardware to include the later NVIDIA Jetson TX2 computer along with DC power boosters that will allow us power the entire car (traction motor and computer) from a single Lithium battery: this will allow to simplify the hardware design and make the car lighter. We aim to do so to study and model the car's behaviours at higher velocities to deepen our understanding of controls of vehicles at extreme scenarios (including driving during rain and, driving in a city).

# Bibliography


[1] 'Phantom Auto' will tour city: Journal article embedded in The Milwaukee Sentinel. 8 December 1926. Retrieved by Google 22 April 2018.
URL: https://news.google.com/newspapers?nid=1368&dat=19261208&id=unBQAAAAIBAJ&sjid

[2] Massachusetts Institute of Technology: A Powerful Platform for Robotics Research and Teaching. RACECAR (Rapid Autonomous Complex-Environment Competing Ackermann-steering Robot) Michael T. Boulet, Spring 2016
URL: http://fast.scripts.mit.edu/racecar/

[3] University of Pennsylvania F1/10 Autonomous Racing (f1tenth.org): Madhur Behl Et al. Spring 2017
URL: http://f1tenth.org/

[4] Self-driving cars and the future of the auto sector. Hans-Werner Kaas, Detlev Mohr: McKinsey and Company Publishing, Stamford Sping 2018
URL: https://www.mckinsey.com/industries/automotive-and-assembly/our-insights/self-driving-cars-and-the-future-of-the-auto-sector

[5] GPS.gov: GPS Accuracy, differentiating civilian and military use of NavStar: U.S. Department of Energy in cooperation with the U.S. Air Force July 1998
URL: https://www.gps.gov/systems/gps/performance/accuracy/

[6] PANS: A Portable Navigation Platform, Carnegie Mellon University: Todd Jochem Et al., Robotics Institute, ACM Digital Spring 1996
URL: https://www.ri.cmu.edu/publications/pans-a-portable-navigation-platform/

[7] Wilfried Elmenreich, An Introduction to Sensor Fusion: Vienna University of Technology Fall 2002
URL: http://www.vmars.tuwien.ac.at/php/pserver/extern/docdetail.php?DID=805&viewmode=paper

[8] FLIR Driver Vision Enhancement System PathFindIR II, FLIR Systems Inc. (2016)
URL: https://www.flir.com/products/pathfindir-ii/

[9] Velodyne HDL-64E with 360 field of view and very high data rate, Velodyne LiDAR Systems; Released 2014
URL: http://velodynelidar.com/hdl-64e.html

[10] Piksi Multi multi-band, multi-constellation RTK GNSS receiver, Swift Navigation; Released 2016
URL: https://www.swiftnav.com/piksi-multi

[11] Texas Instruments sensor fusion architecture for automotive application: Heinz-Peter Beckemeyer (2013)
URL: http://www.ti.com/lit/wp/szzy006/szzy006.pdf

[12] Travis J.Crayton Et al., Autonomous vehicles: Developing the future of transportation policy, University of North Carolina; Spring 2017
URL: https://publicpolicy.unc.edu/files/2017/09/Meier_Crayton-article-9.2017.pdf

[13] Morrison, Margaret (2009): Models, measurement and computer simulation: Springer Philosophical Studies Vol 143
URL: https://link.springer.com/article/10.1007

[14] Extended Kalman Filter, Keisuke Fujii, ACFA-Sim-J Group, Physics and Detector Study for International Linear Collider in Japan
URL: http://www-jlc.kek.jp/subg/offl/kaltest/doc/ReferenceManual.pdf

[15] High Latency vs Low Bandwidth  Impact on Performance (2015): Goran andrli CDN Archives
URL: https://www.globaldots.com/high-latency-vs-low-bandwidth-impact-web-performance/

[16] Ashish Gupta Et al.: High-Availability at Massive Scale: (BIRTE), Springer (2015)
URL: https://static.googleusercontent.com/media/research.google.com/en//pubs/archive/44686.pdf

[17] NVIDIA Announces World's First AI Computer to Make Robotaxis a Reality: Drive PX (Fall 2017)
URL: https://nvidianews.nvidia.com/news/nvidia-announces-world-s-first-ai-computer-to-make-robotaxis-a-reality

[18] Design of an Ackermann-type steering mechanism: Jing-Shan Zhao Et al. Sagee publications (2013)
URL: http://journals.sagepub.com/doi/abs/10.1177/0954406213475980

[19] Pulse Width Modulation: John Caldwell, Texas Instruments SLAU508, Summer 2013
URL: http://www.ti.com/lit/ug/slau508/slau508.pdf

[20] Design of a UART: Eduardo Sanchez, EPFL Teaching material 2009
URL: http://lslwww.epfl.ch/pages/teaching/cours_lsl/ca_es/UART.pdf

[21] NVIDIA Jetson TK1 CUDA performance (2014): Donald Kinghorn, Puget Systems
URL: https://www.pugetsystems.com/labs/hpc/NVIDIA-Jetson-TK1-CUDA-performance-569/

[22] ROS: an open-source Robot Operating System (wiki.ros.org)
URL: http://www.willowgarage.com/pages/software/ros-platform

[23] MAVROS – MAVLink extendable communication node for ROS with proxy for Ground Control Station (wiki.ros.org/mavros)
URL: https://github.com/mavlink/mavros

[24] USB Video Class camera driver for ROS (wiki.ros.org/uvc_camera)
URL: https://github.com/ros-drivers/camera_umd

[25] Packages for interfacing ROS with OpenCV, a library of functions for real time computer vision
URL: https://github.com/ros-perception/vision_opencv

[26] Building a lane detection system using Python 3 and OpenCV: Gallen Ballew
URL: https://medium.com/@galen.ballew/opencv-lanedetection-419361364fc0

[27] Udacity Self Driving Car Nanodegree Sping 2016
URL: https://www.udacity.com/course/self-driving-car-engineer-nanodegree–nd013

[28] A Computational Approach to Edge Detection: John Canny, IEEE Transactions on Pattern Analysis (1986) Vol 8
URL: https://pdfs.semanticscholar.org/55e6/6333402df1a75664260501522800cf3d26b9.pdf

[29] Perspective Transform and Vision System for Robotic Applications: Vincenzo Niola, WSEAS Transactions on Systems, April 2006
URL: https://www.researchgate.net/publication/241152750_Perspective_Transform_and_Vision_System_for_Robotic_Applic

[30] Use of the Hough Transformation To Detect Lines and Curves: Richard O. Duda Et al., Communications ACM 1972 Vol 15
URL: https://dl.acm.org/citation.cfm?id=361242

[31] A general algorithm for determining state-space representations: W.A.Wolovich, Automatica Vol 13 (1977), Published 1978
URL: https://www.sciencedirect.com/science/article/pii/0005109877900565

[32] An approach to the linear multivariable servomechanism problem: P. C. Young, IJC Issue 15 (1972)
URL: https://www.tandfonline.com/doi/abs/10.1080/00207177208932211