Perception Integration for an Autonomous Vehicle

Siriluck Ovenden - 2019

Perception Integration for an Autonomous Vehicle

Siriluck Ovenden

Final Year Project: 2018 Semester 2 – 2019 Semester 1

Department of Electrical and Computer Systems Engineering,
Monash University
Supervisor: Professor Tom Drummond

May 22, 2019

Significant Contributions

- Interfaced with LiDAR sensor data: extracted information from LiDAR packet, created ROS node to subscribe from LiDAR sensor and publish to SLAM.
- Integrated GPS measurements into SLAM.
- Implemented an approximation of the Mahalanobis distance into SLAM.
- Tested and tuned noise parameters in the EKF.
- Automated existing cone detection labeller using template matching to aid labelling efficiency and further tune the neural network.
- Miscellaneous:
  - Designed LiDAR Mount welding jigs in NX12 CAD program
  - Manufactured LiDAR mount for M19-D
  - Researched FastSLAM

MONASH University
Engineering

Department of Electrical and Computer Systems Engineering

ECE4095 Final Year Project 2019

Siriluck Ovenden

Perception Integration for an Autonomous Vehicle

Supervisor: Professor Tom Drummond

Project Aim

To integrate data from various sensors on an autonomous racing vehicle in real time and improve sensor data acquisition. Data from various sensors such as LiDAR, stereoscopic cameras, GPS and IMU are to be integrated into an EKF SLAM algorithm to provide car and obstacle location to feed to path planning and motion control.

Monash Motorsport

Monash Motorsport is a student run team that builds, develops and designs FSAE race cars. The Autonomous Section is responsible for much of the software and hardware that goes into making a Formula Student Driverless car.

GPS integration

With the already existing EKF SLAM (Simultaneous Localisation and Mapping) implementation using LiDAR, the next step was to integrate other sensor data to provide the vehicle with more certainty of its position and the cone locations which mark out the track. This was done on Ubuntu Linux operating system, using C++ and ROS framework for program communication.

Probabilistic Data Association

Improved SLAM by using the Mahalanobis distance which takes into account the uncertainty with which a cone was detected before adding it into the SLAM system.

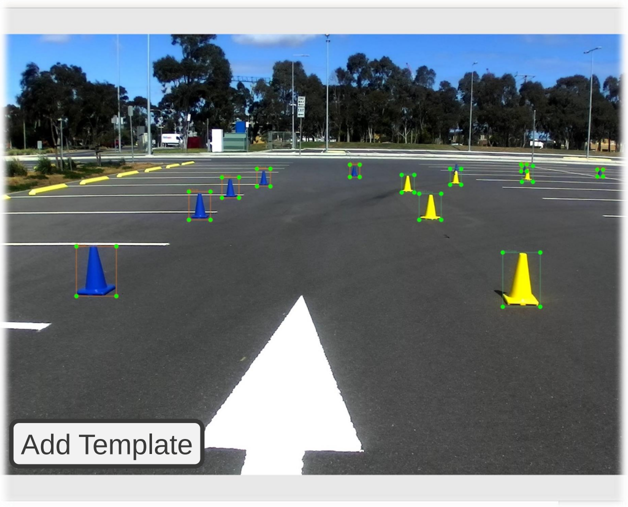


Image Labeling Tool

Prior to this project, a neural network was used to detect cones but had not been trained on enough data, so a tool was implemented to aid labeling of images. This involved adjusting and adding code to already existing open source software. Using template matching in OpenCV to detect and label cones decreasing tedious data labeling which is required to train a neural network.

Executive Summary

Formula Student is a competition for university students to build and design race cars. Monash Motorsport plans to compete in all three classes in 2019: electric, combustion and driverless. The driverless class involves an autonomous car fitted with sensors, computing units and actuators. As part of Monash Motorsport's Autonomous Section, this project contributed to the integration of perception and Simultaneous Localisation and Mapping aspects of the autonomous vehicle. This project also implemented the integration of GPS measurements into the EKF SLAM algorithm to increase certainty of the car's position. The accuracy of the vehicle's position contributes to the accuracy of the map created by SLAM based on detected cones which mark out a track. Probabilistic data association was also implemented using the Mahalanobis distance, which increased the accuracy of the track map produced by the SLAM algorithm. Integrating stereoscopic cameras into SLAM proved to be difficult, as the neural network used for detecting camera cones required more training data. This project also details the custom implementation of an auxiliary program to aid in image labelling to further train the neural network in the future.

Acknowledgements

The following list of people and many others contributed and guided me through making this project possible.

- Bryce Ferenczi - for being an excellent academic and critical thinker
- Michael Mattiske - for implementing a working EKF SLAM using LiDAR
- James Wyatt - for your love of solving problems
- Jack Coleman - for being all-knowing about cameras and neural networks
- Shreya Ramesh – for your work on GPS and IMU
- Dr. Titus Tang - for always providing enthusiasm and mentorship

Special thanks to my supervisor, Professor Tom Drummond, for providing excellent advice and wisdom to the Autonomous Section. I fondly owe my gratitude to everyone on the Monash Motorsport team who make M19-D testing possible, were supportive of the new Autonomous Section and who provided me with advice during the design and manufacturing period. I would like to mention my gratitude to the Autonomous Chief Engineer, Aryaman Pandav, for his dedicated hard work and for being a great mentor and friend. It has been an honour to be part of one of the highest performing teams in the world and have the privilege of developing the first driverless FSAE vehicle in Australia.

MONASH University
Engineering

Contents

List of Figures

List of Tables

Abbreviations

MMS	Monash Motorsport
FSAE	Formula Society of Automotive Engineers
FSD	Formula Student Driverless
FSG	Formula Student Germany
LiDAR	Light Detection and Ranging
GPS	Global Positioning System
INS	Inertial Navigation System
IMU	Inertial Measurement Unit
SLAM	Simultaneous Localisation and Mapping
EKF	Extended Kalman Filter
ROS	Robot Operating System
RViz	ROS Visualisation tool
YOLOv3	You Only Look Once version 3
MKL	Math Kernel Library
CAD	Computer Aided Design

Introduction and Motivation

Monash Motorsport (MMS) has been active since the year 2000. Traditionally, the purpose of the team was to design, build and race combustion cars in Australia at the Formula Society of Automotive Engineers (FSAE) competition. In 2017, MMS entered a second car into competition, an electric vehicle alongside the combustion car. In 2017, the competition expanded to include a new competition class called Formula Student Driverless (FSD) where Formula Student race cars navi-gate through known and unknown tracks. These tracks are simplistic in that they are primarily on flat ground, with coloured cones to form a path in an enclosed environment.

- In 2018, MMS started a new department dedicated in research and development for the driverless car.
- In 2019, the electric vehicle, named M18-E, was retrofitted with sensors, computing units and actuators to become the first ever Driverless car that MMS has produced, named M19-D. The research areas included low-voltage systems, stereoscopic cameras, LiDAR, GPS/INS, Computing, Path Planning and Ve-hicle Actuation.

In the first half of this project, the author was involved in LiDAR selection, LiDAR mount design and implementing Simultaneous Localisation and Mapping (SLAM). In the second half of this project, the author was part of the Perception section which includes SLAM, LiDAR, GPS/INS and stereoscopic cameras. The motivation for MMS to develop this car is to compete in Germany and promote interest in an Australasian driverless competition in the future. The motivation for this Final Year Project is to improve the perception and localization abilities of the race car.

The Team

The purpose of Monash Motorsport is for students to practice and extend their engineering skills, professionalism, camaraderie and improve performance. The team typically consists of about 80 students who belong to various sections such as Aerodynamics, Autonomous, Chassis (and suspension), Powertrain, Business and Upper Management. The two cars shown in Figure 2.1 are the combustion car (66, left) and electric car (E65, right). The electric car was transformed into M19-D, the driverless car of 2019, by

- removing the front wing to make room for the LiDAR VLP-16 mount,
- installing a mount for the Stereolabs ZED camera and a mount for the Piksi GPS,
- having the right radiator replaced with the computing box which contains interfacing components and computing units such as the NVIDIA Jetson TX2 and an Intel i7 Motherboard
- adding braking and steering actuators in the cockpit along with their pneumatic circuits.

Figure 2.1: MMS team of 2018 at the Australasian Competition, Winton

The Competition

Formula Student is the world's largest engineering student design competition, with hundreds of teams competing from over a hundred countries and held in several locations around the world. Traditionally, the vehicles that competed used combustion engines. After that, the electric vehicle class was introduced. Since 2017, the FSD class was introduced in Germany at Formula Student Germany (FSG), which was the first country to host a driverless event and the first to set the rule book for this new class. The following year, Formula SAE Italy, Formula Electric Italy, Formula Student UK and Formula Student East also held similar competitions that involved driverless technology. MMS plans to compete in FSG in the year 2020, with the aim to complete all events. That is, successfully start and finish all dynamics events. This may seem like a light goal, but it is indeed quite difficult to finish all competition events even in the traditional electric and combustion class. In its first year, 2017, with 15 teams entering FSD, only one team managed to finish the trackdrive event. In its second year with 17 teams, only three cars finished the trackdrive event with others failing on early laps. The team that has stood out from the rest is AMZ Driverless (Zürich ETH), who started off winning and have only improved in performance year by year. The competition consists of static and dynamic events. The tracks which are known prior (see Fig. 3.1. and Fig. 3.2) to the competition include the acceleration track, which is a straight of 75m, and skidpad which has two tangential circles designed to test the car's cornering ability. The car must recognise the stop area for both acceleration and skidpad events and come to a complete standstill to be successful. The autocross and trackdrive events are of interest because they require navigation through an unknown track autonomously. The track is marked out with yellow cones on the right and blue cones on the left. The trackdrive and autocross have identical closed-loop tracks (Fig. 3.3) that are unknown until the day at competition. The circuit can have a layout that has any combination of straights, turns and hairpins. In the autocross event the car must navigate through this unknown track for the first time, while in the trackdrive event the vehicle must navigate a racing line and complete 10 laps of the same circuit to be successful.

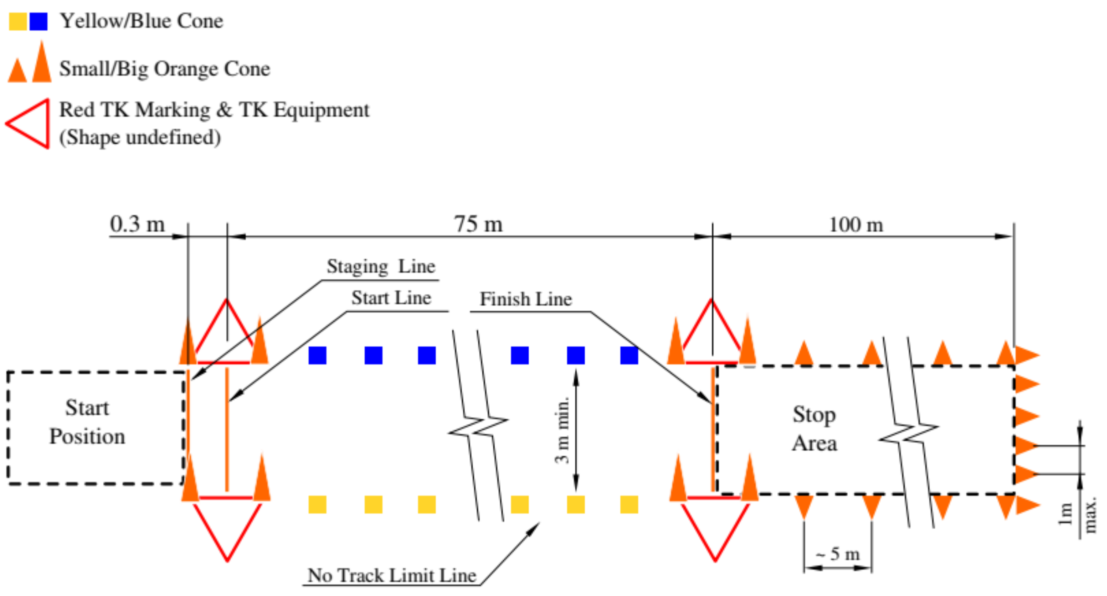


Figure 3.1: Acceleration Event

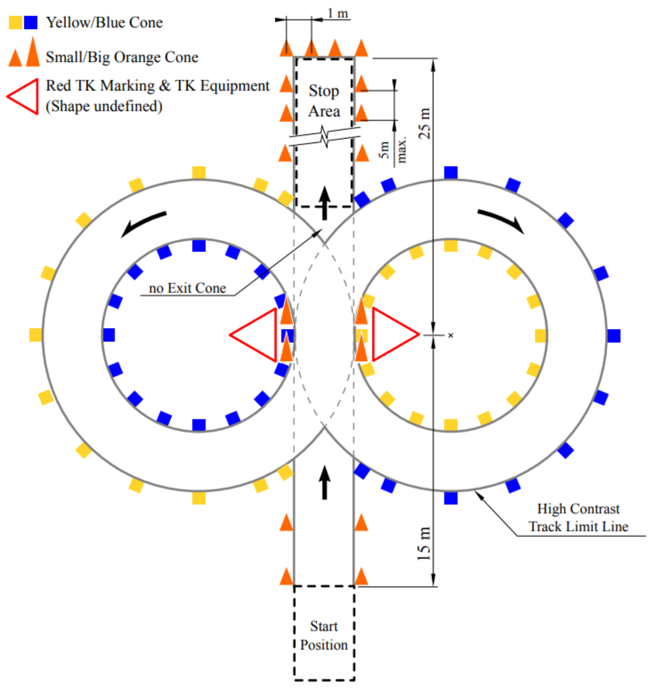


Figure 3.2: Skidpad Event

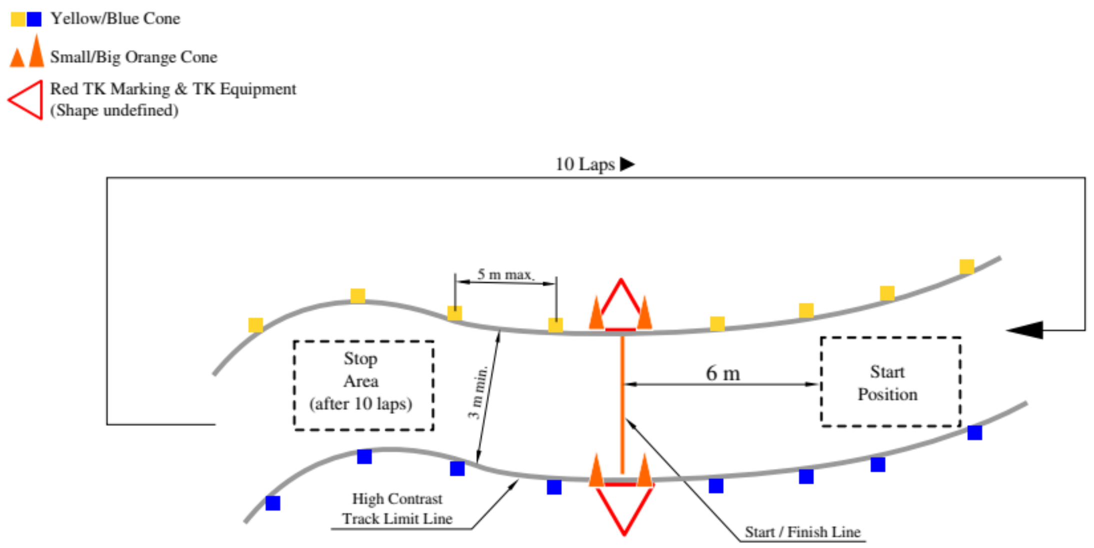


Figure 3.3: Trackdrive and Autocross Event

Literature Review

This project specifically is to aid in developing an autonomous race car for FSD, under the FSG rules. Therefore, most of the research performed is limited to what other driverless teams have done, and any papers they have released on this project. Observing what other teams that have come before Monash Motorsport have done with driverless technology gives an insightful overview for how the system should work. The analysis of some teams provides motivation for why the techniques discussed later in this report should be used, such as probabilistic data association (using the Mahalanobis distance) and a tool to aid in image labelling. Having a general overview of the hardware, cone detection and cone association algorithms used is useful for what MMS should aim towards.

A team from Norway, Revolve NTNU, used a LiDAR and a monoscopic camera [2]. The 3D points of the LiDAR are projected onto the image captured by the camera. Calibrating the two sensors requires knowledge of the relative position of the camera and LiDAR, which they determined using OpenCV functions to solve the projection problem. This is what MMS may consider in the future, as outlined in the Section 7.0.3: Future Work where camera cones and LiDAR cones get merged first before being processed by SLAM. This research paper suggested two methods which both require labelled training data for the machine learning algorithms. They calibrated LiDAR and Camera data using tools such as ROS Visualisation tool (RViz) for LiDAR and Geeqie for cameras, and then found a few distinct points in LiDAR data that are visually obvious in image data. They divided the task of labelling images across the whole team using the open source software, LabelImg, and suggested that a second person should check that all the frames and box positions are good enough to train the neural networks. Revolve also used YOLOv3, a neural network for camera cone detection. This team had a similar problem to MMS with the ZED stereo cameras when speed was increased, which led them to use a custom stereo camera setup with a global shutter. Revolve NTNU performed many tests with different neural networks and found YOLOv3 to be the best, which is what MMS is using.

The top team in this field of study is AMZ who have won all competitions that they have entered over 2 years and consistently release many helpful papers and other data which helps competing teams to gain traction in this difficult competition. AMZ use a higher resolution LiDAR than MMS, the Velodyne Puck Hi-Res while MMS use a VLP-16. The greater vertical resolution allows AMZ to detect cones using LiDAR at theoretically 4m further away than MMS (see Appendix A) given our cone detection technique. For camera cone detection, they developed a neural network that not only draws rectangular bounding boxes around a cone, but uses ‘edges’ at the 7 points that distinguish a cone, such as the top of the cone, or the colour transition from blue, to white, to blue again, or from yellow, to black, to yellow again [3]. This resulted in depth estimates from the PnP algorithm[1] of up to 10m. Feature matching and triangulation using stereoscopic cameras to batch right and left bounding boxes used SIFT descriptors and brute-force matching. AMZ use FastSLAM 2.0, because the particle filter structure inherently allows for computationally efficient independent data associations. LiDAR and camera pipelines are treated independently, providing observations at different uncertainty models and different delays. The SLAM map is updated each time a new landmark is observed. AMZ use the Mahalanobis distance as a measure of likelihood with which to associate cone measurements.

Another team, KTH Formula Student from Sweden, explored different methods of object recognition and different data association methods for SLAM. They explored the Mahalanobis distance to compute individual compatibility and concluded that it is adequate because spurious landmark measurements rarely occur and the sensors they used have high accuracy [1]. Another method is joint compatibility branch and bound, which considers all measurements in one update, but computation require-ments increase up to two orders of magnitude. They tested SLAM algorithms but seem to have only tested on small closed loop tracks using about 32 cones. For sensor fusion between LiDAR and Cameras, they viewed LiDAR and camera information on the same visualization by restricting the FOV of the LiDAR.

[1]The PnP (Perspective-n-Point) is the problem of finding the pose of a camera in the world frame given a set of n 3D points and their respective 2D image projections and known camera parameters.

[2]More detail is provided on the topic of different latency and timing issues between camera and LiDAR in Table 5.1 in Section 5.3

Software

5.1 The Framework

The Robot Operating System (ROS) framework was used to run and test sensor integration with the SLAM system. This required including packages and libraries such as Eigen (which was coupled with Intel Math Kernel Library (MKL)), and various standard sensor message types. The results and the track map from SLAM were displayed using RViz (ROS's 3D visualisation tool) which depended on message types provided by the mms master customized ROS package. Using ROS, data was transferred between subsystems through 'topics'. ROS nodes (C++ programs) subscribe to or publish to these topics. The ROS bag utility was used to store all recorded data from sensors or any program output. A bag may contain multiple topics and their message types which are unique to the topic. Each message in a ROS bag is coupled with a timestamp so the data can be played back in the exact order and timespan it was recorded. LiDAR cone detection was done in the mms lidar node, and camera cone detection was done in the mms camera node. The output of these nodes were then independently processed by the SLAM node. GPS and INS messages from the raw bag data were also transferred straight to SLAM running in the Ubuntu Terminal.

5.2 Integration Overview

In 2018, the research and development phase of developing the driverless car, M19-D, began. This included the components such as sensors and computing units, and the introduction of new activities such as gathering data and testing cone detection algorithms on the data gathered. The autonomous system uses a single LiDAR and camera, however the vehicle could theoretically run with only one of them. The team made the decision to purchase both for redundancy and research. The data gathered included the point cloud from the LiDAR, stereo images from the camera, GPS signals and data from the IMU. This data is incredibly useful when incorporated into SLAM, the sensor fusion algorithm, which gives a better estimate of the overall position of the car and the position of the cones on the track.

5.2.1 Gathering Data

MMS achieved the collection of raw data by setting up a track in an unused car park using cones to demarcate the path. A 'trolley' was used with the sensors and computing box mounted, pictured in Fig. 5.1, as no driverless car was available at the time for team members to test with. The computing units and sensors were mounted on this trolley, which was then pushed around the track (see Fig. 5.2). The motion and vehicle dynamics of the trolley is completely different from an actual car, making IMU sensors unreliable and very noisy. This is due to the surface of the bitumen and having no suspension on the trolley. It was not physically possible for team members to push the trolley around at high accelerations and turning speeds that the real car would produce, so there were limitations in testing this aspect of the sensors. Dealing with sensors such as IMU which are noisy due to a test vehicle that is dissimilar to the motions of the real car was not ideal. However, sensor data from LiDAR, GPS and cameras were gathered reliably because their perception was not interfered by how the trolley moved. Later in the timeline at the beginning of 2019, the electric FSAE vehicle built by MMS was able to be used for driverless purposes since it had finished its last competition in December, at the Australasian FSAE Competition 2018. This vehicle was fitted with the sensors, computing box and actuators in 2019 to become M19-D, Monash Motorsport's first driverless vehicle.

The cones used at the FSG competition (see Fig. 5.3) have black or white bands. Ideally, these would have been purchased from Germany and shipped to Australia, however this proved to be too costly. Therefore, the data gathered at testing is of cones that are pure yellow and pure blue. The implications of this is that the perception section is currently gathering data to train the neural network, YOLOv3, on generic blue and yellow cones without the black and white bands.

European project teams also exchange labelled data via a Github repository as the teams know that gathering and labelling camera data is extremely tedious and too time consuming given the yearly project timeline. MMS would like access to the database but the collaboration requires mandatory contribution of a minimum of 600 labelled mages of FSG cones which MMS does not have. Some teams have approached this issue by forming a partnership with a company that labels data professionally.

5.2.2 Testing

Gathering data using the trolley meant that the cone detection algorithms could be tested and GPS signals were recorded for later use in SLAM. At the time of this project, camera cone detection and stereo matching[1] were not yet robust enough to be integrated into SLAM, so only LiDAR cone detection was used. Once the FSAE vehicle was ready in March 2019, testing was performed with one of the team's drivers who drove around the track at high speeds, producing more accurate data from the cameras and IMU. This made it more realistic for testing software, such as cone detection algorithms and the SLAM algorithm. A setback became evident when the driver turned around corners at higher speeds (about 40km/h) and caused the rolling shutter from the camera to be evident, which affects how the neural network detects cones. This cannot be easily solved through software, as it is a phenomenon that occurs when the camera vibrates, such as when it is fixed to a car. A solution through software would require the motion of the car to be taken into account as the camera writes the pixel values. However, it is simply easier to acquire a camera that has a global shutter.

When the car is turning, the cones in the camera image are skewed as seen in

[1]Stereo matching is the process of forming a 3D construction from 2 sets of images. Another team member is working on improving robustness and computation time, as this requires a GPU which the Author does not have.

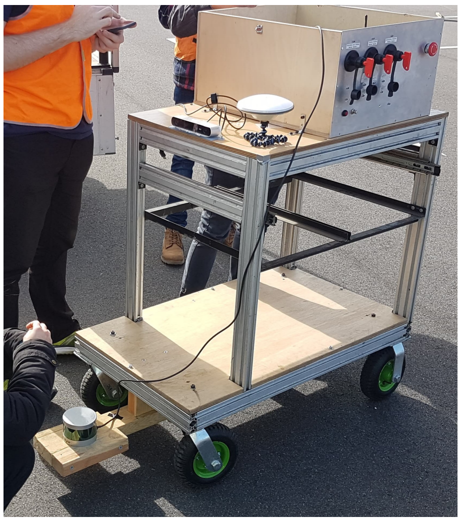


Figure 5.1: Testing trolley with mounted sensors and computing box

Figure 5.2: Data was gathered by pushing a trolley around a track

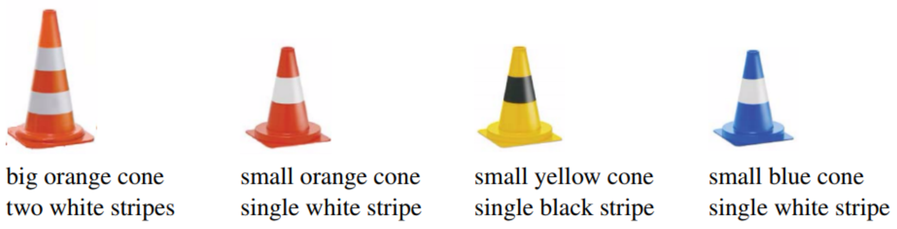


Figure 5.3: Cones types at the FSG competition

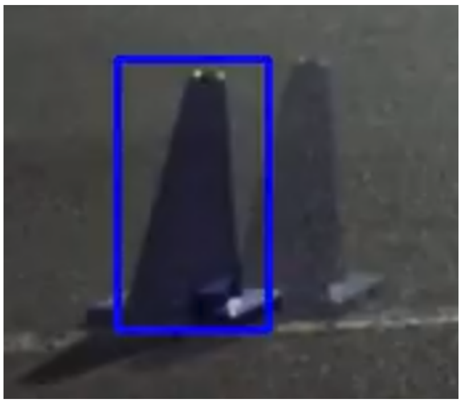


Figure 5.4: Cone skew due to rolling shutter

Fig. 5.4. This figure shows a blue cone in shadow that is captured but appears asymmetric about the vertical axis. Note that the ghost shown in this image is not seen by the neural network, it is only included for visual analysis of the stereoscopic camera.

After raw LiDAR data and camera data was gathered, these are then put through their respective cone detection algorithms. The output of these algorithms provide cone locations and cone types to the SLAM algorithm in real time. So far, only LiDAR cone detection works when integrated into SLAM, implemented by another

FYP student. Testing of the SLAM algorithm and the integration of GPS and IMU can be done on any computer that has Linux, all the required ROS packages and math libraries installed. Testing of the camera cone detection neural network re-quired a computer which has a GPU to perform the parallel processes that real-time image processing requires. The cone outputs would then be stored in a bag which can be run on a regular computer to test integration with the SLAM algorithm. In general, 10 minutes of recorded stereoscopic camera data uses about 210GB while other sensors such as LiDAR and GPS/INS inclusive use only 3GB over the same interval. These two requirements of a computer with a GPU and a lot of storage space encumbered camera integration, as the team only has access to 2 devices that met these requirements: the computer mounted on the car and a team member's personal desktop.

5.3 Sensor Fusion

The cone detection outputs are passed on to a SLAM algorithm, which is a form of sensor fusion. SLAM itself is a generic term for a class of algorithms that may differ depending on the type of mapping and the scope required for a particular use case. In this case, the SLAM method was based on the Extended Kalman Filter (EKF). The EKF is simply a Kalman Filter that has been extended to non-linear systems using a Jacobian. The general overview of how a SLAM algorithm works is by using a combination of dead reckoning and perception sensors to improve the accuracy of the position and heading of a robot or vehicle. A simple SLAM system may take odometry readings and sensor data from a laser scanner, which detects the distance of the surrounding objects. Each time the vehicle moves, the algorithm will predict where the objects will be in the current time step and compare this to the actual measured distances. This comparison means taking the difference between predicted cone position and measured cone position, otherwise known as the innovation. The algorithm keeps track of every object seen and the vehicle's position (and derivatives) in a state vector, and their relations in a covariance matrix. The state vector contains the pose of the vehicle and the cone locations, while the covariance matrix holds the uncertainties of the state variables in relation to each other. Generally, the school of thought is that adding more sensors will provide more information about the environment, thus making the SLAM system more accurate. One aspect of the mapping component of SLAM is the data association method, which is used to determine if a newly measured object is the same as what is already in the state, or if it should be appended to the state. A key term in SLAM is loop closure, which is when the vehicle recognises an environment that it has seen before by the position of the cones, the type of cones, and the vehicle's current position. Loop closure is useful for gaining certainty of position and understanding the start and stop line in SLAM for the trackdrive event. The sensor fusion aspects discussed here is the addition of GPS measurements into the SLAM system.

Sensor fusion itself is a broad topic. The term could refer to fusion on various levels including fusion of features and decisions [4]. The type of sensor fusion relevant to this report is

- cross sensor fusion, where the sensor measurements involve the same physical

  objects, such as LiDAR and stereoscopic cameras measuring the position of cones
- cross attributes fusion, where sensors measure different qualities across the same situation

These types of fusion will contribute to receiving an input, detecting features, and making a decision to feed to the next block in the pipeline: path planning and motion control. A full system diagram is shown in Fig 5.5.

The LiDAR and stereoscopic camera in this case are complementary, because the sensors do not directly depend on each other, but can be combined to provide more accurate cone locations. They each offer different attributes outlined in Table 5.1. Together, they compensate for each other's weaknesses to create a perception system that builds a more complete and robust environment. The LiDAR and stereoscopic camera sensors have a cooperative configuration with GPS/INS and wheel speed sensors which are fused in SLAM to provide more accurate data about vehicle location and cone location.

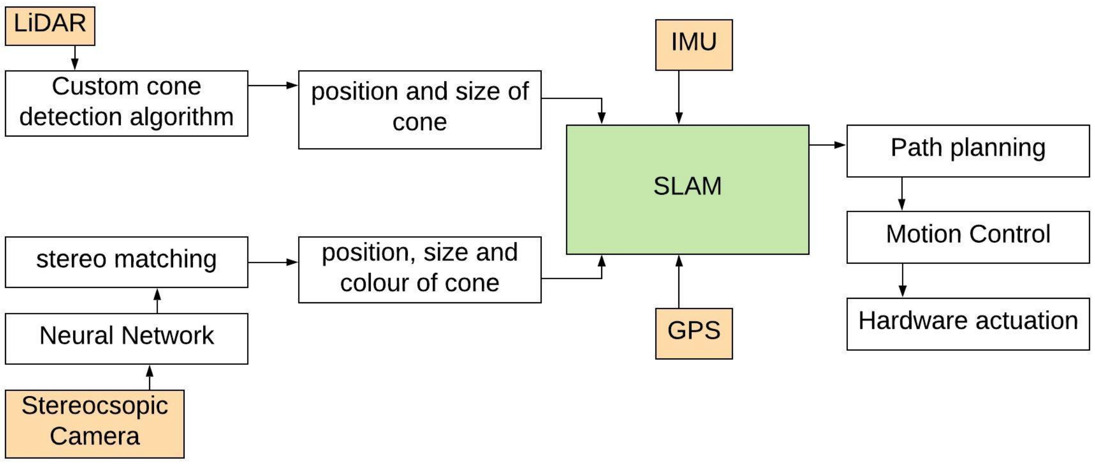


Figure 5.5: Full perception system flow diagram

Table 5.1: LiDAR v Camera

	LiDAR	Stereoscopic Camera
Colour	Can see colour of cones but to a limited extent, and would need a higher vertical resolution to draw useful conclusions	Can see colour of cones
FOV	Larger horizontal FOV, with 360deg FOV but once mounted on the car is limited to 240deg, which is better for hairpin corners.	Cameras are only forward facing, with 70deg FOV
Weather conditions	Not susceptible to weather conditions, lighting conditions, (rain, overcast, sunny)	Detection algorithm must be trained on data that has a wide range of weather and lighting conditions (rain, overcast, sunny)
Latency	A LiDAR sweep is every 100ms, cone in-formation message is published 5ms af-ter this	Frame rate is 30 FPS, cone information message is published 20ms after this.
Number of objects	LiDAR produces 1 cone at a time as detection is done as the LiDAR sweeps.	Cameras may detect any number of cones from an image.
Distance	Detect a cone at 5-10m, limited by ver-tical resolution	Distance: Neural network can detect a cone and feed to stereo matching for cones 15-20m away

5.3.1 Adding GPS into SLAM

After EKF SLAM was working in C++ purely using LiDAR cone detection, sensor inputs from GPS needed to be included. The motivation for using GPS is to reduce the uncertainty of the vehicle's position, which in turn reduces the uncertainty of cone positions in the world coordinate frame. The GPS system produces centimetre accurate results due to a base station and RTK (real time kinematics). Running EFK SLAM solely on LiDAR cone detection exhibited some drift of about 2-3 meters by the time the car 'closed the loop' after it had driven through the circuit and returned to the starting position. While existing MATLAB code from a previous FYP student provided calibration of the GPS and integration into its own EKF, it lacked any methods to combine the coordinate frames which were different for GPS and the global frame.

There are two cartesian coordinate frames that are relevant when fusing the GPS sensor with SLAM.

1. the 'world' coordinate frame, which is the ground-truth frame that doesn't change. When SLAM is initialised, this and the vehicle frame are initialized such that the positive x-direction of the frame is the vehicle's initial forward heading.
2. the 'GPS' coordinate frame, which is also initialised to the same position as the 'world' and 'vehicle' coordinate frames, however the relative rotation between them is unknown. The X and Y axis represent longitude and latitude respectively. The positive x-direction of the GPS frame is always in the easterly direction.

Before GPS can be integrated, the car location in the GPS frame must be rotated by an angle α_{gps} to match the car location in the SLAM frame. If the car is heading east, given the GPS positive x-direction is east, then the measurements agree. However, the rotation of points is required if the car is heading in any other direction, such as north-west, while the positive x-direction of GPS remains pointing towards east. To solve this problem, the GPS frame is initially assumed to coincide with the SLAM frame. Now the SLAM frame will have the car position at one point, while the GPS will likely measure the car to be at another point. The angle between two vectors is calculated, where one vector is of the x and y coordinates of the car's position in the SLAM frame, and the second vector is the x and y coordinates of the car's position in the GPS frame. The angle between the two vectors is calculated from the x and y coordinates of the car from the SLAM frame (x_{state}, y_{state}) and the GPS frame (x_{gps}, y_{gps}).

$$\alpha = \cos^{-1} \frac{x_{state}x_{gps} + y_{state}y_{gps}}{|x_{state}, y_{state}| \cdot |x_{gps}, y_{gps}|}$$

Before calculating α, SLAM must wait for the vehicle to move a certain distance away from the origin before attempting to integrate GPS. This is so that the angle of the vector of the car's position in the SLAM and GPS frame can be calculated with respect to the origin and any noise in the GPS measurement or SLAM estimation doesn't cause wildly varying angles at small distances.

To ensure that the correct sign of the angle is applied to the rotation, the sign of the angle is reversed when the cross product is greater than zero. This is to ensure that this method works on different tracks that may start a westerly heading direction, which would be opposite to the GPS's heading direction.

$$cross\ product = x_{state}y_{gps} - y_{state}x_{gps}$$

The angle $\alpha_{slam,i}$ with which to rotate the car from the GPS frame to the SLAM frame is calculated using a weighting. This weighting is between previous angles $\alpha_{slam,i-1}$ calculated and the current angle α calculated and n is the number of times a GPS measurement has been received. This behaves in the same way as storing all previous estimations of the relative frame angle and averaging them. This is reasonable as the angle is a fixed constant once SLAM is initialized.

$$\alpha_{gps,i} = \alpha_{gps,i-1}(\frac{n}{n+1}) + \alpha(\frac{1}{n+1})$$

Finally, the GPS x and y coordinates are calculated from rotating the original x_{gps}, y_{gps} by the new GPS frame angle.

$$\begin{bmatrix} x_{rotated} \\ y_{rotated} \end{bmatrix} = \begin{bmatrix} cos(\alpha_{gps,i}) & -sin(\alpha_{gps,i}) \\ sin(\alpha_{gps,i}) & cos(\alpha_{gps,i}) \end{bmatrix} \begin{bmatrix} x_{gps} \\ y_{gps} \end{bmatrix}$$

The SLAM algorithm may exhibit some accumulated error when using only LiDAR for the update step, shown in Fig. 5.6. The integration of GPS results in the large orange start cones being seen in almost exactly the same position as they were first seen, shown in Fig. 5.7, which means the end result is more accurate. This can potentially aid in loop closure when the car is travelling at a faster pace and the INS can not accurately provide measurements to help predict the vehicle position.

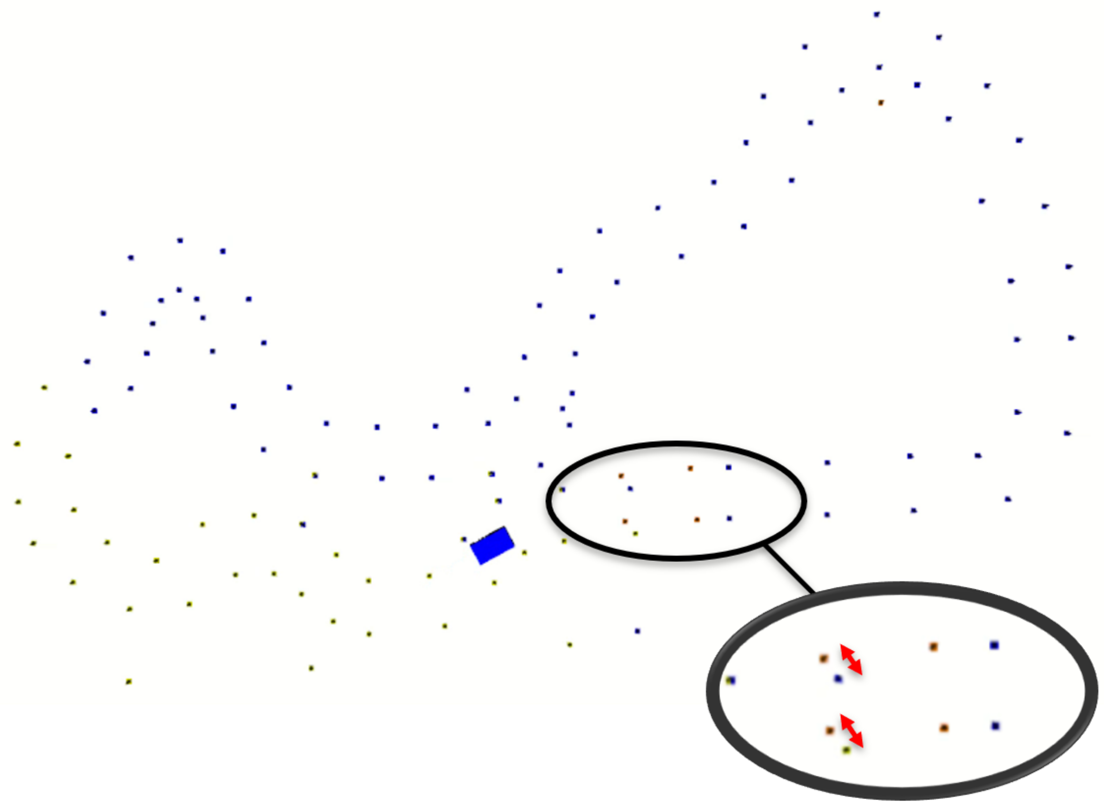


Figure 5.6: SLAM without GPS

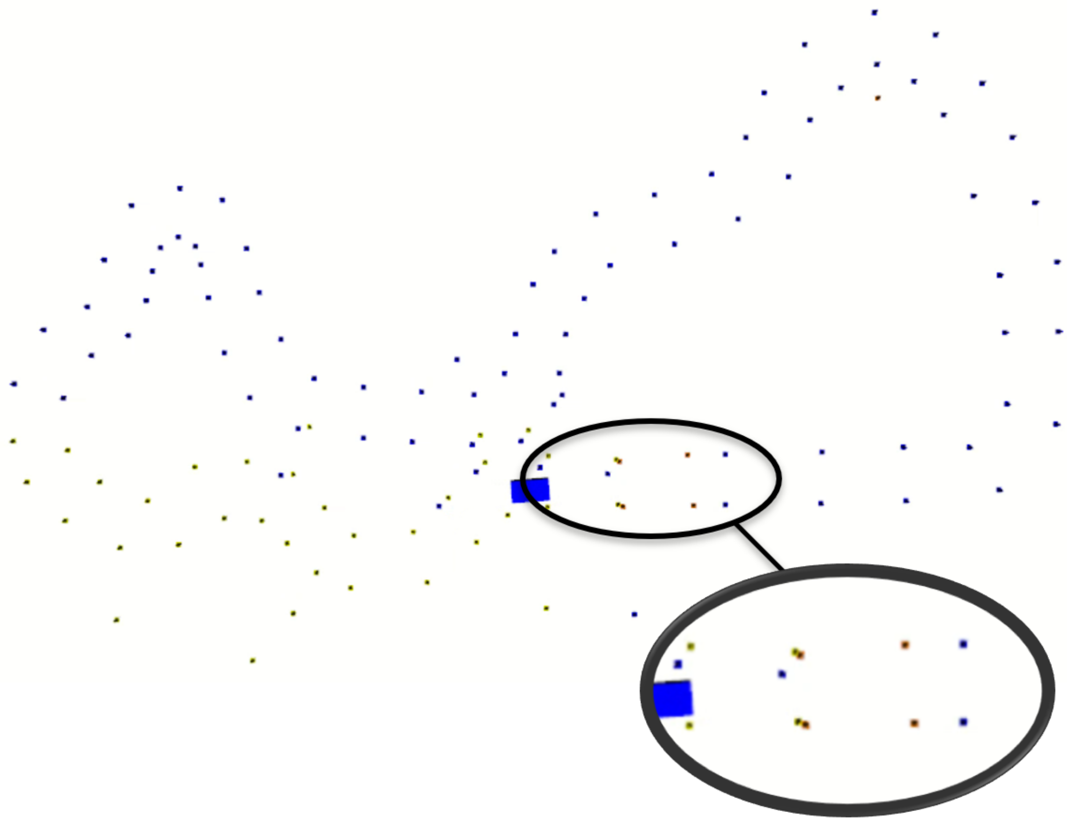


Figure 5.7: SLAM with GPS

5.3.2 Adding IMU into SLAM

The Inertial Measurement Unit (IMU) consists of accelerometer, magnetometer and gyroscope. These sensors provide information about the motions of the vehicle such as the rate of turning around a corner and acceleration. In order to integrate IMU, the EKF SLAM code had to be changed to accommodate eight state variables along with other system variables. The state matrix is an n × 1 matrix that contains the pose of the vehicle and the positions of all landmarks. The pose of the vehicle was expanded to include yaw rate $\dot{\theta}$, acceleration $\ddot{x}$, and acceleration $\ddot{y}$. The 8 states variables are

$$x, y, \theta$$

$$\dot{x}, \dot{y}, \dot{\theta}$$
$$\ddot{x}, \ddot{y}$$

IMU integration is a 'double edged sword' in that it comes with the benefits of frequent sensor inputs, but this also consumes processing time and power, which slows down the overall computation of SLAM. IMU is the fastest sensor integrated with SLAM, capable of providing measurements at 200Hz, while other sensors are typically 30Hz. Including the IMU is computationally expensive, requiring large calculations of the covariance matrix at each update. With the computing power available, the only way to integrate IMU without SLAM failing was to subsample by an empirically determined factor of 8, beyond this point SLAM could no longer complete all the required computations to ensure no data was lost. At the time of this project, the IMU mounted to the testing trolley was noisy due to the trolley vibrations on the rough surface of the bitumen in the carpark, and any cornering did not produce clean enough[2] acceleration values to be useful in SLAM. The accel-eration magnitudes are much greater in a vehicle being driven (Fig. 5.8) compared to the trolley (Fig 5.9). This figure shows noisy IMU on the trolley while the trol-ley was being pushed slowly straight ahead from a starting position. This meant that the data was more useful to analyse and develop algorithms for. However, the IMU sensor itself has still proven to be noisy so the measurements aren't as heavily weighted by the SLAM algorithm as other sensor inputs or the control input. The SLAM algorithm 'trusts' (weights more) the information provided by a sensor mea-surement based on the noise covariance that the programmer provides. For instance, if there is a sensor that is noisier than others, then its noise covariance values are increased by the programmer, so SLAM won't allow the sensor to heavily influence the state. The programmer would obtain these numbers through empirical testing the accuracy and repeatability of a sensor. If a sensor is not 'trusted' it means the sensor's input does not weigh heavily in SLAM's update step.

[2]After March 2019, when the electric vehicle could be used, the sensors were mounted and another team member successfully integrated IMU and gyroscope into SLAM.

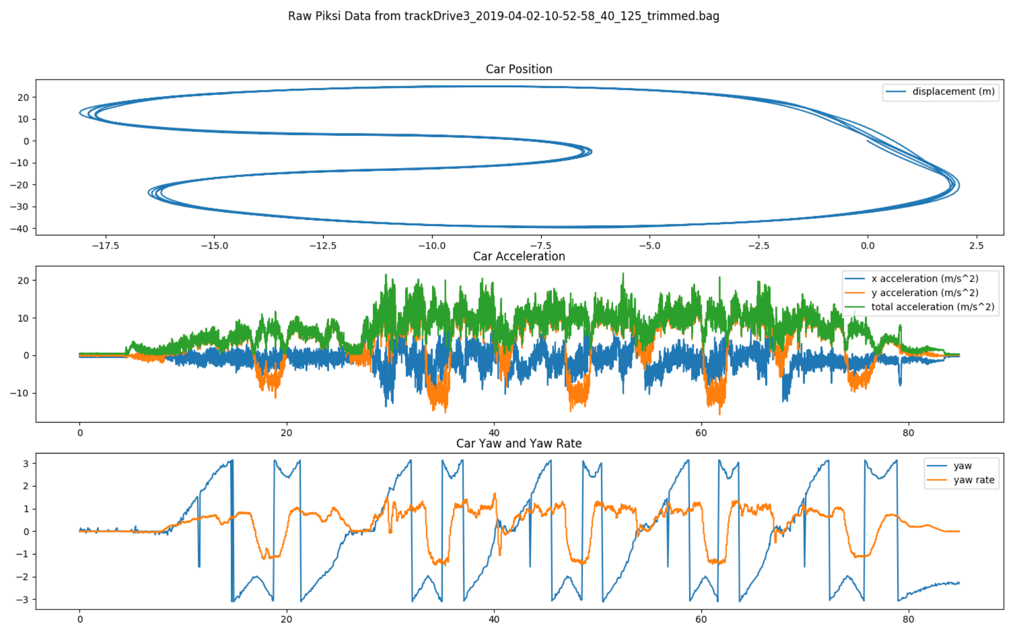


Figure 5.8: IMU data from vehicle has higher accelerations and is less noisy than trolley.

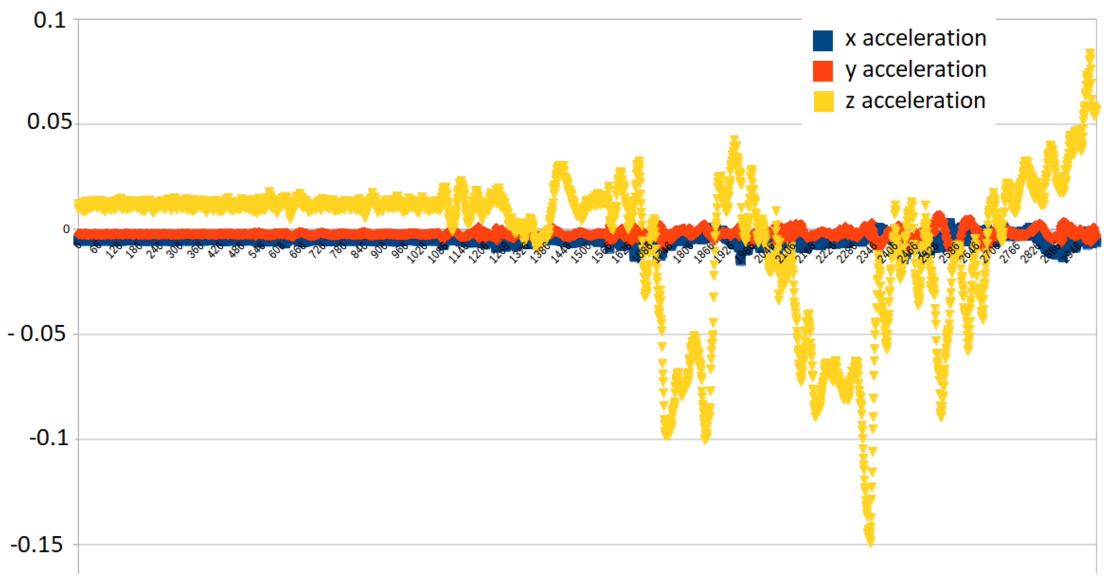


Figure 5.9: IMU data from trolley

5.3.3 Probabilistic data association

Data association in SLAM involves cone detection and the addition of new cones into the SLAM state. Data association is the process of making the decision to either update an existing cone based on a new measurement or add a new cone to the state from the new measurement. Data association is the process that adds a new cone to the track. Before this FYP, data association was done based on the Euclidean distance: if the measured cone position is too far away from any other cone in the state, then append the measured cone to the state. If the measured cone position is close enough to a cone that is already in the state, then they are deemed to be associated. The existing cone in the state is updated using the new measurement. Implementing probabilistic data association allows SLAM to take into account the existing cone's certainty[3] from the covariance matrix and compare it with the measured cone's certainty which is based on the vehicle and measurement's certainty. That is, if the vehicle's position is uncertain, then the measured cone's position must be even more uncertain. The motivation for replacing Euclidean distance with the Mahalanobis distance is to prevent false positives, to more accurately combine cones that are seen twice after loop closure, and to distinguish cones that are placed side by side. The scenario of when cones are placed side-by-side occurs at the starting position of the track drive and autocross event. A covariance of each cone location is kept in the EKF SLAM covariance matrix, which provides information about the certainty of a cone position in the SLAM frame.

A simple comparison of the Euclidean distance to the Mahalanobis distance given using Fig. 5.10 and Fig. 5.11 and Table 5.2. The figures show the blue cones with their respective covariance ellipses circled around them.

Table 5.2: Are A and B the same cone?

	Fig. 5.10	Fig. 5.11
Euclidean	Yes	Yes
Mahalanobis	Yes	No

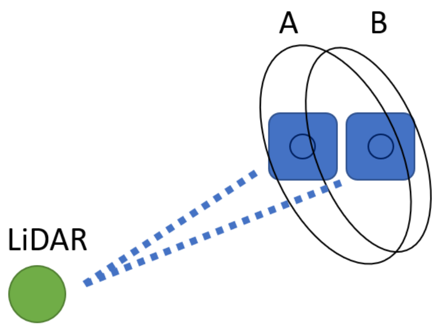


Figure 5.10: Cones with overlapping covariance

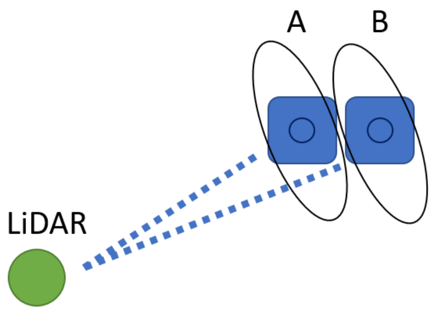


Figure 5.11: Cones with separate covariance

[3]What is meant by 'certainty' here is the 2x2 covariance matrix of the cone's x and y position.

The implementation of the Mahalanobis distance calculation is a statistical approximation, as more specific calculations require too much computation and time for it to be a feasible real-time solution. Σ_{pooled} is the combination of the covariance of the cone from the state and the covariance of the measured cone.

$$\begin{bmatrix} d_x \\ d_y \end{bmatrix} = \begin{bmatrix} x_{state} \\ y_{state} \end{bmatrix} - \begin{bmatrix} x_{cone} \\ y_{cone} \end{bmatrix}$$

$$\Sigma_{pooled} = \frac{1}{2}(\Sigma_{state} + \Sigma_{meas})$$

Mahalanobis distance:

$$D^2 = \begin{bmatrix} d_x \\ d_y \end{bmatrix}^T \Sigma_{pooled} \begin{bmatrix} d_x \\ d_y \end{bmatrix}$$

A cone is then combined into the state if its Mahalanobis distance is within a threshold to an existing cone. Otherwise, it is added to the state as a new cone.

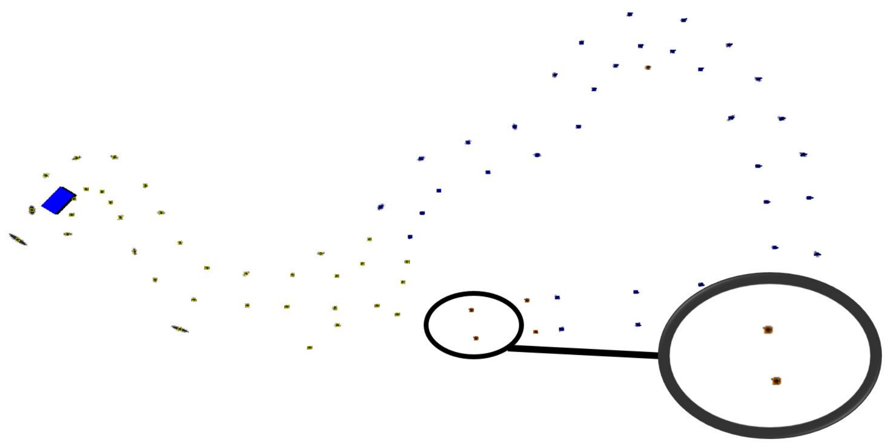


Figure 5.12: SLAM using Euclidean Distance.
This shows the map created by SLAM before using the Mahalanobis distance.

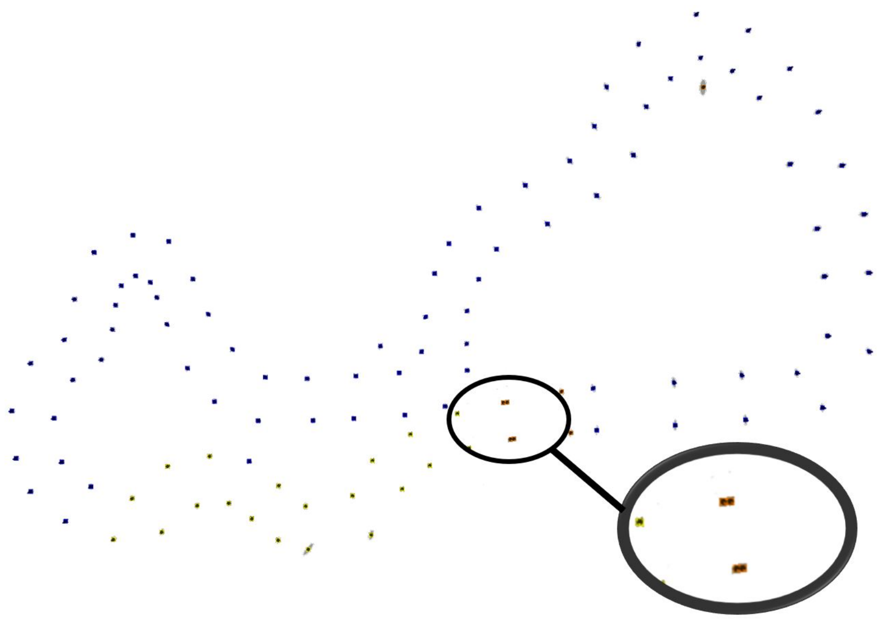


Figure 5.13: SLAM using Mahalanobis Distance.
This shows the map created by SLAM after using Mahalanobis distance. It can be seen that the two start cones (see Fig. 5.14) are no longer associated as one cone.

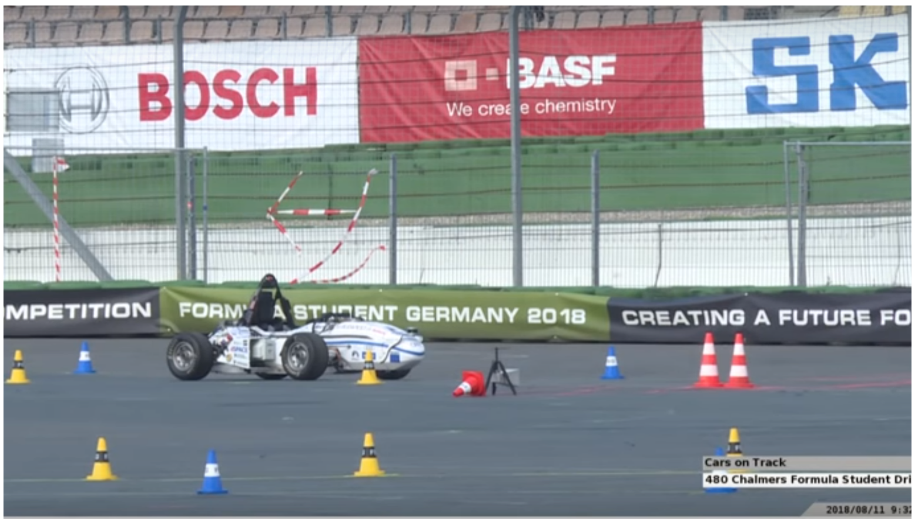


Figure 5.14: FSG screenshot from live stream. The big orange cones are placed next to each other after the starting position

5.4 Camera Cones Annotation

Camera-cone detection in MMS's system is very jittery and inaccurate, which affects stereo-matching and therefore the placement and distance of the cones when input into the SLAM algorithm, as shown in Fig. 5.15. The mms camera node performs isolated stereo matching on regions of the image detected as a cone, and constructs a set of cone positions on a 2D map. This map viewed from the vehicle's point of view is shown in Fig. 5.16.

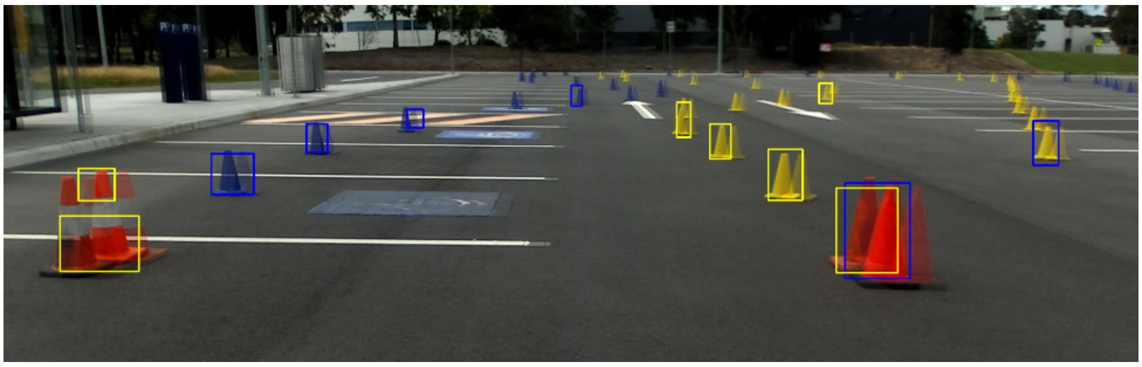


Figure 5.15: Camera cone detection output

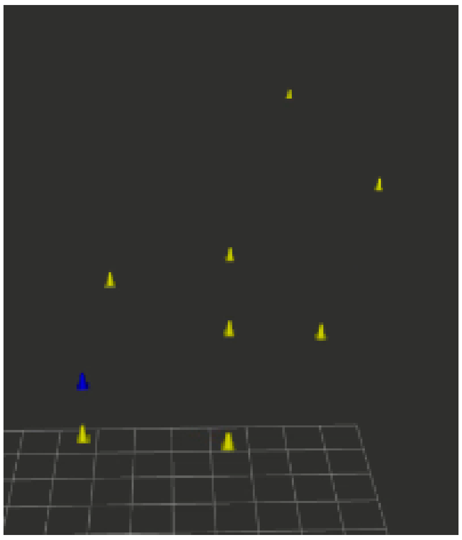


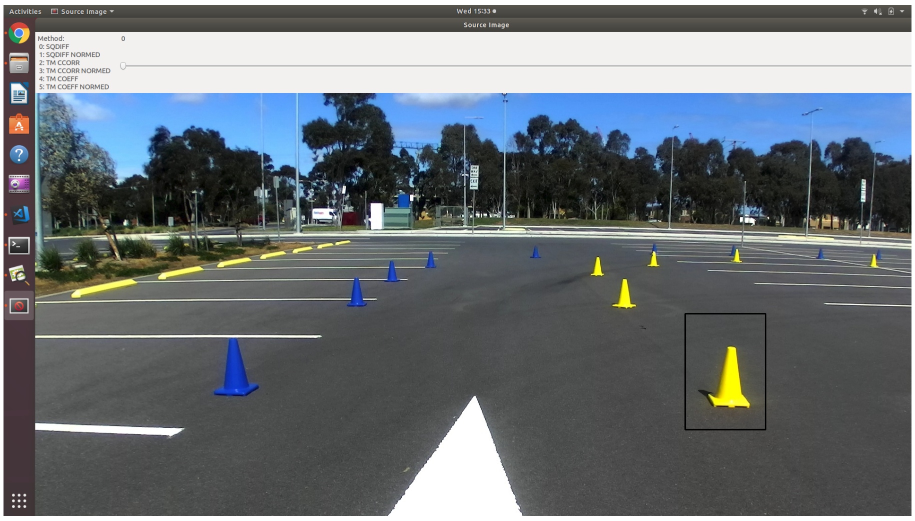
Figure 5.16: Cone position as a result of camera cone detection

It can be seen in Fig. 5.15 that the placement of the bounding boxes around the cones does not match the cone's in the image very well. What may have contributed to this inaccurate matching is the neural network (NN) has not been trained on enough data. It has only been trained on 300 images, while other teams have trained their neural networks on tens of thousands of images. Some teams claim to have 45,000 images labelled[4], typically by sponsors or otherwise outsourcing this service. When this cone detection output is provided to SLAM, the positions of the cones keep changing, and some of this can be attributed to non-robust stereo matching. Another disadvantage is that the cones MMS have access to, are not the same as

[4] Revolve NTNU Facebook page, 'Reveal 2019' video

those used at FSG. This difference in the types of cones detected can affect how the neural network performs when faced with FSG cones. While we are training our data with artificially augmented cones to develop a robust neural network, we cannot know the actual performance when our system sees the actual competition cones.

To make the process of labelling images easier, a tool was implemented to be used alongside the existing LabelImg software. LabelImg is an open source image labelling tool that can output labelled image data in the YOLO format and PASCAL VOC format. Labelling images is a tedious and time-consuming activity for the Autonomous members of the team. A tool was developed which utilized OpenCV's template matching function to extract multiple cone positions from an image based on a template of a cone. The original code only returned one instance of a region in the target image which best matched with the template. This program was com-piled and included in a fork of the LabelImg repository. This open source software code was transformed to include a button for the user that executes the template matching C++ code. The progression of the code development can be seen from Fig. 5.17 to Fig. 5.18 to Fig. 5.19 and eventually to Fig. 5.20.



```
Method:          0
 0: SQDIFF
 1: SQDIFF NORMED
 2: TM CCORR
 3: TM CCORR NORMED
 4: TM COEFF
 5: TM COEFF NORMED
```

Figure 5.17: Off the shelf Template Matching

Originally, the intention of the program was to draw bounding boxes around cones using fixed templates to minimise user input and maximise automation. However, it only works when the template is very similar to those regions present in the image. This resulted in the merge of the C++ program for cone detection with the LabelImg program to have an easy interface for the user to add a template. At the click of the button "Add Template", all other similar cones are labelled. To detect more distant cones in the image, the template size had to be changed from large to small so that the template matching worked properly. To minimise false positives, the hue value of the middle pixel of the template must be close enough to the hue of the middle of the matched regions.

If one general template is used for all images, template matching fails because the lighting conditions are different. Although they may be the same hue, the resulting image is completely different and contains incorrect boxes and many false positives, as seen in Fig. 5.21 when using the template from the previous image, 5.22. The

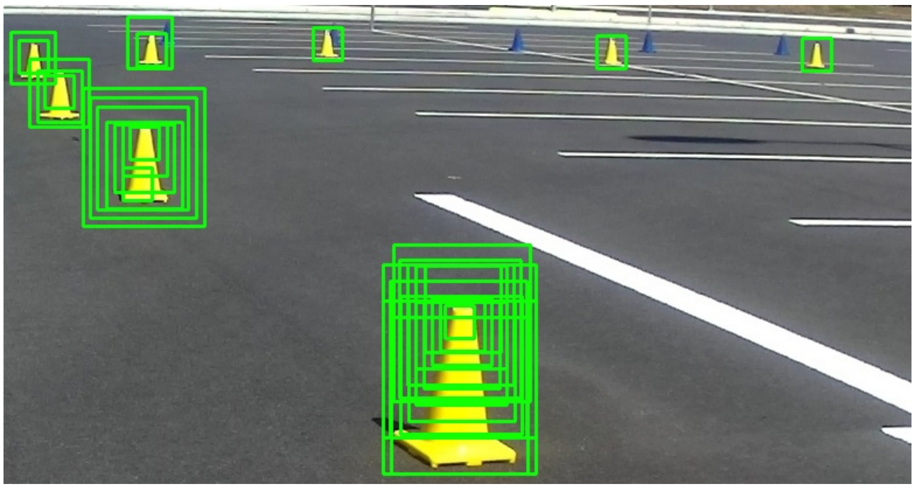


Figure 5.18: Off the shelf Template Matching

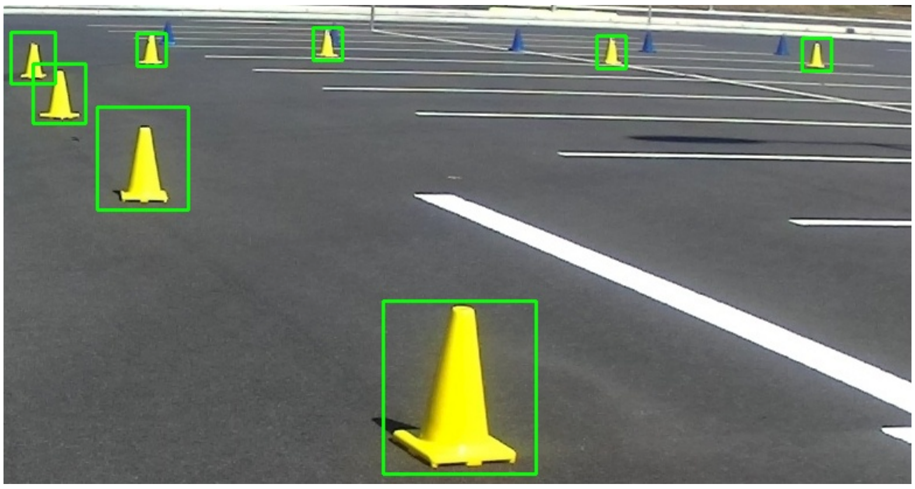
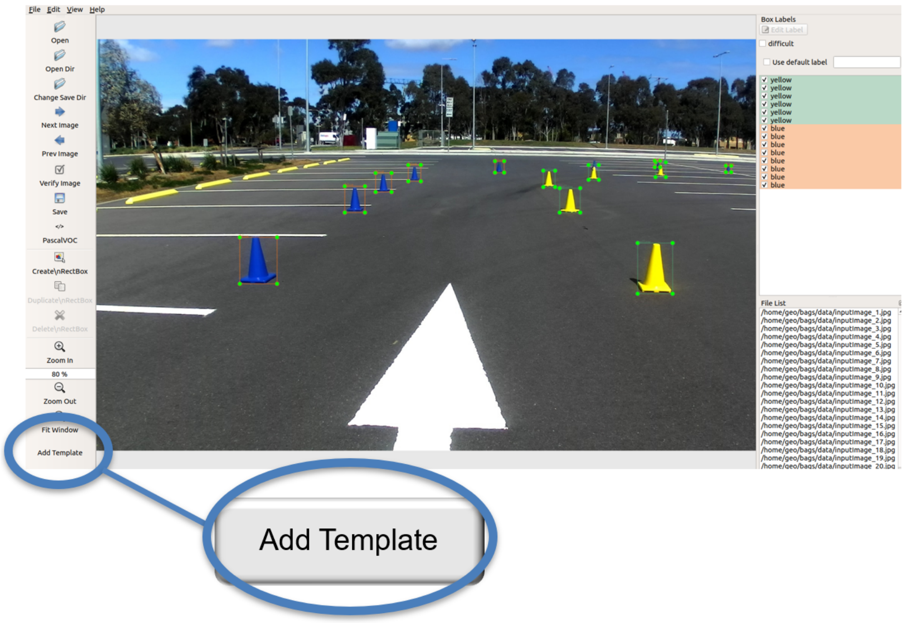


Figure 5.19: Template Matching of different sized objects

template does not resemble the cones in the target image well enough, as the template is in full sunlight while the cones in the target image are in shadow. This is the reason for including the ability for the user to add a template. The template drawn in the current image matches the lighting conditions in which the cones were captured.

The limitations of this is solution is it does not work well for very distant cones, which look too blurry as shown in Fig. 5.23 to be matched well using a template. Hence, the distant cones would have to be drawn in manually using the LabelImg tool. Even if the user added another template by drawing one around the small blue cones, the template matching algorithm will likely pick up many false positives because the blue colour is so similar to the grey bitumen. In addition, the cone in the template might have white background lines from the car park while other cones do not, which would further interfere with the template matching algorithm.



```
/home/geo/bags/data/inputImage_1.jpg
/home/geo/bags/data/inputImage_2.jpg
/home/geo/bags/data/inputImage_3.jpg
/home/geo/bags/data/inputImage_4.jpg
/home/geo/bags/data/inputImage_5.jpg
/home/geo/bags/data/inputImage_6.jpg
/home/geo/bags/data/inputImage_7.jpg
/home/geo/bags/data/inputImage_8.jpg
/home/geo/bags/data/inputImage_9.jpg
/home/geo/bags/data/inputImage_10.jpg
/home/geo/bags/data/inputImage_11.jpg
/home/geo/bags/data/inputImage_12.jpg
/home/geo/bags/data/inputImage_13.jpg
/home/geo/bags/data/inputImage_14.jpg
/home/geo/bags/data/inputImage_15.jpg
/home/geo/bags/data/inputImage_16.jpg
/home/geo/bags/data/inputImage_17.jpg
/home/geo/bags/data/inputImage_18.jpg
/home/geo/bags/data/inputImage_19.jpg
/home/geo/bags/data/inputImage_20.jpg
```

Figure 5.20: Modified LabelImg program

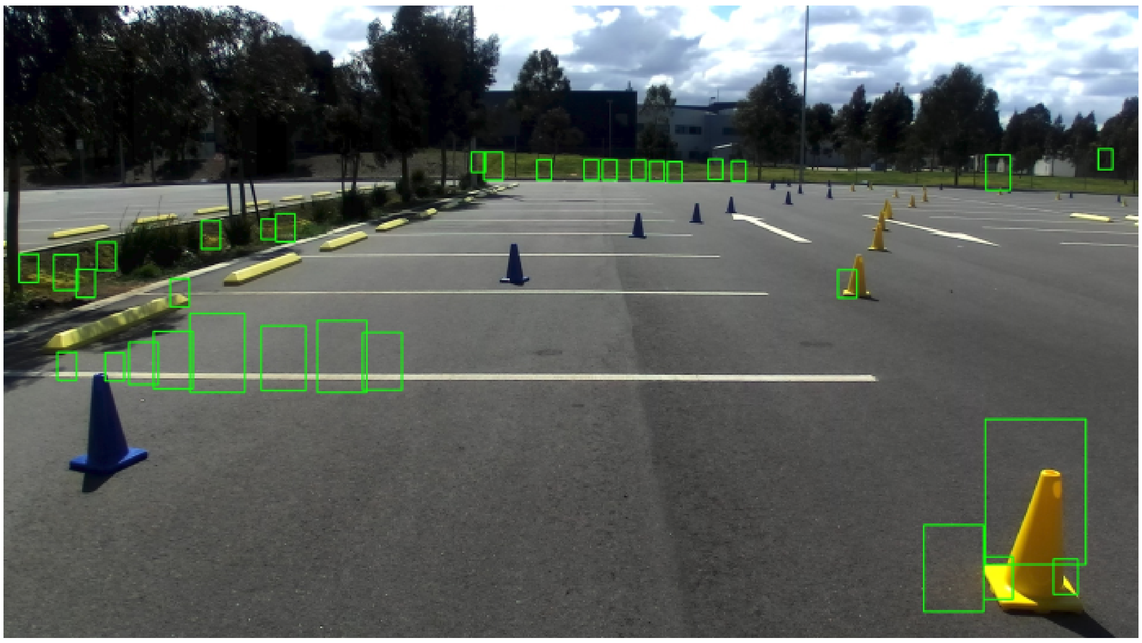


Figure 5.21: Template Matching, with false positives

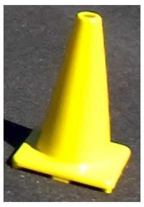


Figure 5.22: Bright yellow cone template

Other methods apart from template matching were explored, such as using flood-fill methods and HSV methods. However, these methods proved to be too sensitive to different shading and lighting conditions, because it would be difficult to set a good threshold value for each new image from different days of testing. HSV and flood-fill can also be sensitive when the blue colour blends in too much in the bitumen on a cloudy day, or when a yellow cone blends in with a white background line of the car park. These experiments are shown in Fig. 5.24, and Fig. 5.25 where orange and yellow cones cannot be easily distinguished from each other. Flood-fill also failed on a dark day where blue cones blended in with the bitumen and on sunny days where the yellow cones blended with the painted white lines. The threshold values for both methods were not robust under different lighting conditions. HSV was implemented in such a way to guard against false positives for the main template matching algorithm.

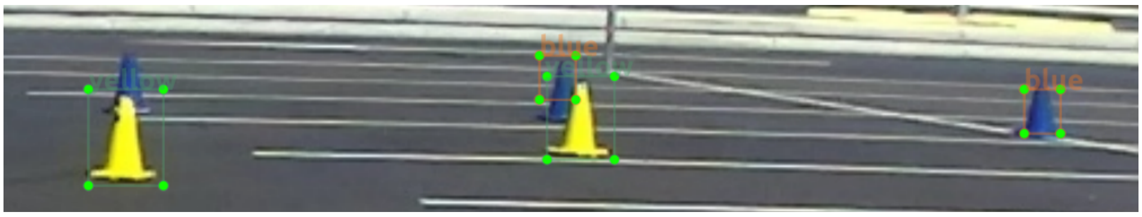


Figure 5.23: Distant blue cones do not get matched very well

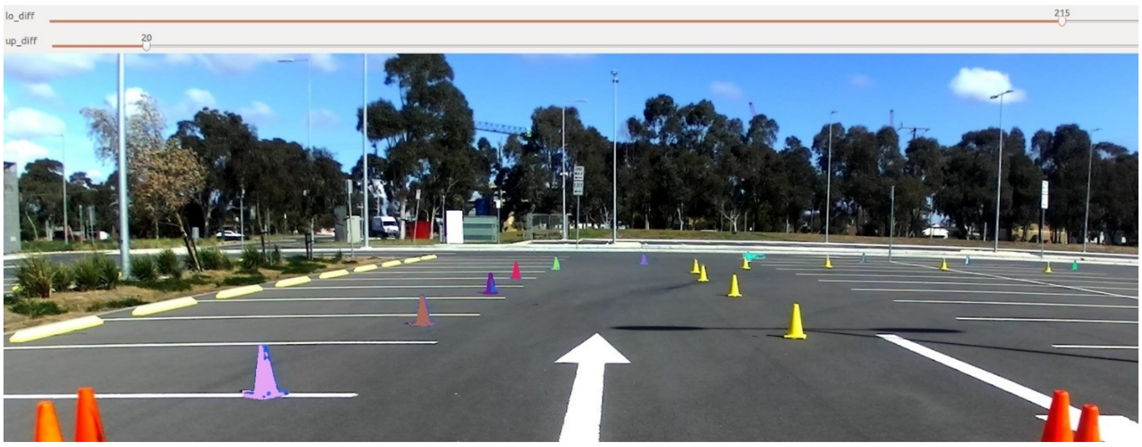
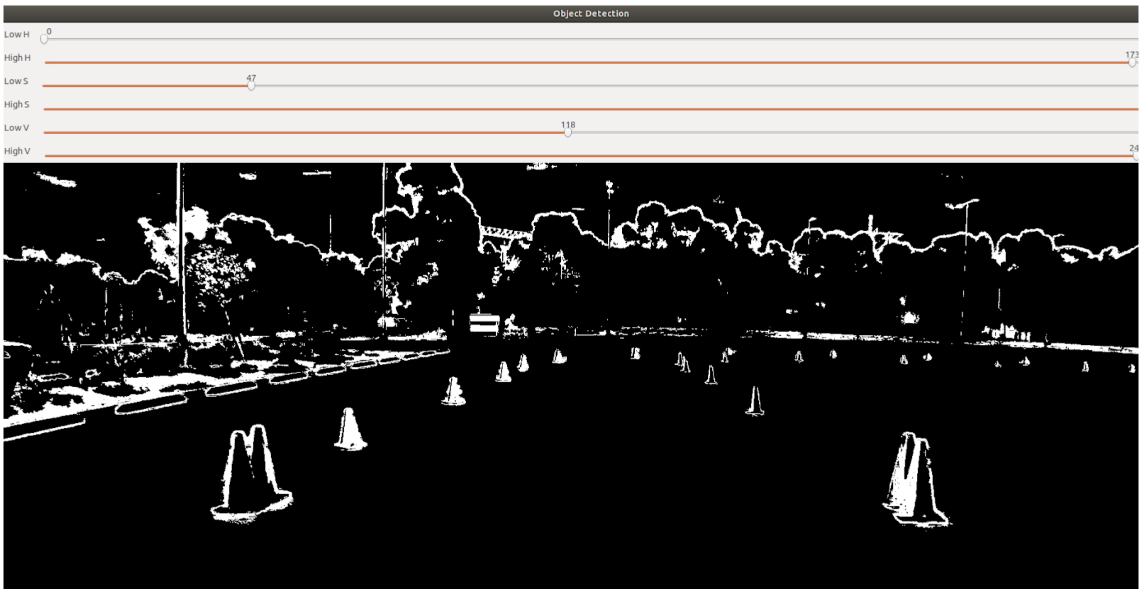


Figure 5.24: Exploring Flood-Fill



Low H 0
High H 173
Low S 47
High S
Low V 118
High V 24

Figure 5.25: Exploring HSV

Discussion

As part of the Autonomous section, this FYP contributed to the perception systems for M19-D; the driverless FSAE vehicle. The GPS has been successfully integrated into LiDAR-based SLAM. The drift in the SLAM system due to the summation of error over each measurement and distance travelled has been corrected by the integration of GPS. The SLAM system also has more accurately determined cone positions by using the Mahalanobis distance as the data association method without sacrificing noticeable computational power. Team members can use camera anno-tation software to quickly annotate images containing cones from data gathered at testing. The tool has been integrated with an already existing and popular tool, LabelImg. The template matching tool is therefore easy to use and improves effi-ciency. The customised camera annotation software has limitations in that it can have false positives which need to be deleted and it requires manual editing if the algorithm could not detect all cones. False positives may occur due to the distance of a cone or different shadowing present in the testing area, in which case the user can easily delete them. As Monash Motorsport's driverless package continues to be developed, this FYP provides a foundation for current and prospective Perception Engineers working on future vehicles.

Miscellaneous

This is other work that was contributed to the Monash Motorsport team. The LiDAR mount is designed to be robust and protect the LiDAR from cone hits while the car travels at high speeds. The manufacturing of this was the Author's responsibility. Research was also done into FastSLAM which may provide some help for future implementation.

7.0.1 LiDAR Mount

Fig. 7.1 shows the LiDAR mount highlighted in orange, which sits at the front of M19-D, and the grey object underneath are the jigs which were designed for the purpose of manufacturing. The full document of how the mount was made is documented on the MMS internal Wikipedia. The resulting product can be seen in Fig. 7.2.

Thank you to all those who welded these parts for me and provided advice: Filip Surla, James Murray, Reece Day, Lachlan Hore, Aryaman Pandav, George Lloyd, Will Harding, Paul Hendy, Leon Shi, Justin Green and others.

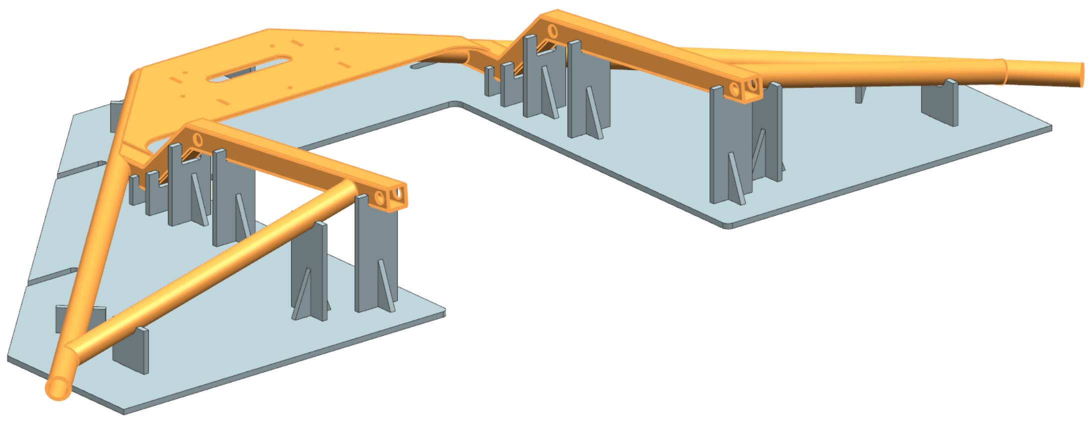


Figure 7.1: LiDAR Mount
LiDAR Mount (orange) and manufacturing jigs (grey) in Computer Aided Design (CAD)

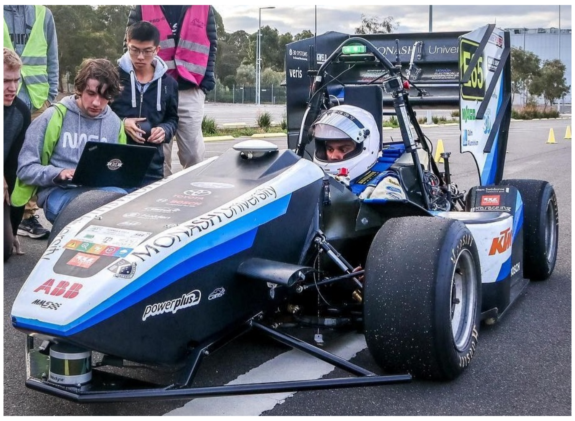
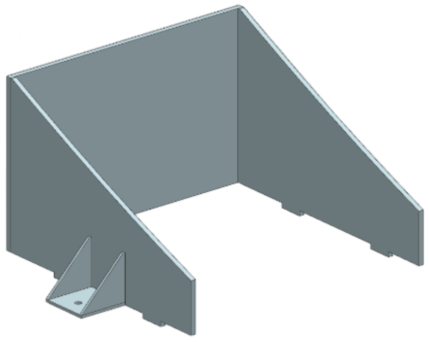


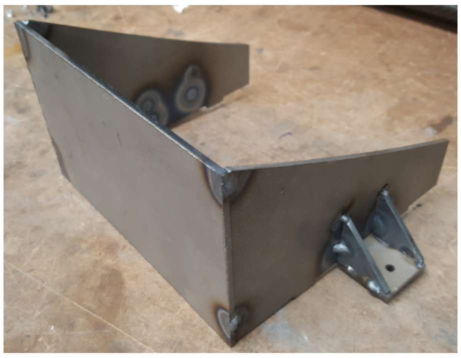
Figure 7.2: LiDAR Mount final product
Finished LiDAR mount on M19-D, when M19-D was driven manually before autonomous actuators were implemented.

7.0.2 LiDAR Cage

The LiDAR Cage protects the LiDAR while it is mounted on the car but not in use. The nature of the workshop where M19-D is built, tested and maintained involves people moving around. Since the LiDAR is mounted low to the ground, there was a risk of people hitting it with their boots or dropping tools on the expensive sensor.



(a) LiDAR Cage



(b) LiDAR Cage welded, before sand blasting and painting

7.0.3 FastSLAM

The initial plan was to implement FastSLAM, but because EKF was working successfully with LiDAR, the team pushed forward to integrate everything else to get the driverless system up and running as soon as possible. This significantly re-duced the amount of time dedicated to working on FastSLAM and thus it never reached the development stage. The following summary may prove useful for future development:

The initial goal is to get a small system to work first, with a small number of particles on your preferred programming language that includes an IDE that is viable for easy visualisation. Without seeing the program run on a 2D map with vehicle pose and cone location, there is no evidence that your algorithm works. An overview of the

FastSLAM algorithm is described by the flow diagram in Fig. 7.4.

How the FastSLAM algorithm works: Begin by defining a FastSLAM class which uses the Particle class. Each particle contains the pose of the vehicle, landmark positions and their covariances (2x2 EKF). Each particle represents one path and one map. Each particle is ”certain” they know exactly where they are. The Particle class is responsible for these operations:

- Calculates vehicle pose given the landmarks seen and given the control input. Hypotheses are made on the motion model, which can be non-linear and non-gaussian.
- Calculates the EKF of each landmark, new information is used to update the PDF (probability distribution function) of the car pose and to decrease uncertainty.
- Computes the likelihood of each landmark being correlated to a landmark that has already been seen.
- The correlation could be done using a minimum mean square algorithm.
- If not enough correlation, a new instance of a landmark is created.
- If a landmark has been seen before, then its correspondence gets updated to a new value.

The FastSLAM class is responsible for these operations:

- Predicts where each particle will be, given the control inputs (movement, odometry)
- Each particle is updated with a new weight given how much ”what is seen” matches what was ”predicted to be seen”.
- The particle is given a new ’weight’ or gets decommissioned if it’s unlikely that that particle represents the true state of the world. The weight of the particles is determined by two sources of uncertainty: the sensor and the estimate (motion model)
- Each particle belongs in a pool of samples that all get given a ’weight’ depending on how likely it is to be true given the control input and the observation.
- The weights are normalized, i.e. distributed to sum to one.
- Particles get resampled in the next time step according to their weights.
- Resampling should only be done if the car has moved, or new measurements have come into play to avoid particle deprivation.
- A map of each resampled particle is updated using a standard EKF. But sensor noise must be white, zero-mean and gaussian, and the observation mode is linearized.

Loop Closure can be aided by the fact that big orange cones will be placed before and after the start, finish and timekeeping lines. FastSLAM will be difficult to implement because of all the parameters that must be tuned, such as the number of particles used, noise variables and FastSLAM will require more computation. It is worth for future Autonomous Systems members to look further into this.

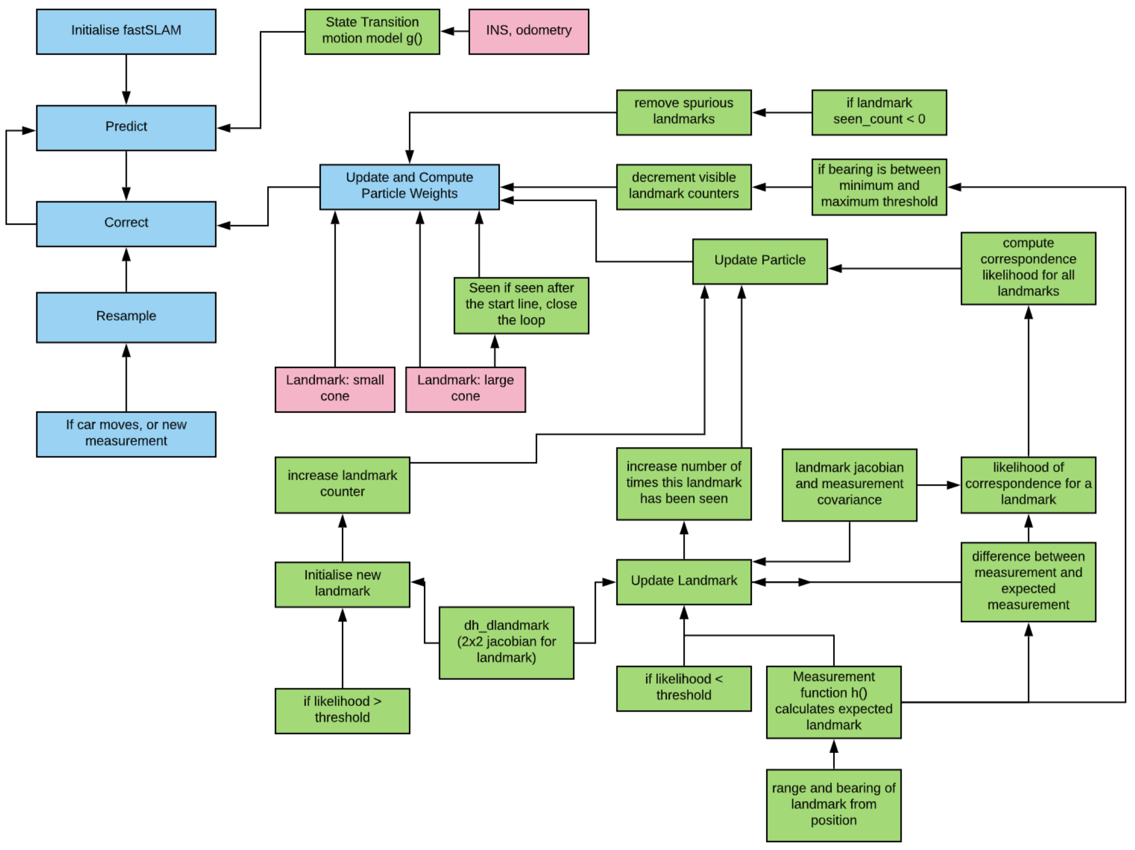

Initialise fastSLAM
State Transition motion model g()
INS, odometry
Predict
remove spurious landmarks
if landmark seen_count < 0
Update and Compute Particle Weights
decrement visible landmark counters
if bearing is between minimum and maximum threshold
Correct
Update Particle
compute correspondence likelihood for all landmarks
Seen if seen after the start line, close the loop
Resample
Landmark: small cone
Landmark: large cone
If car moves, or new measurement
increase landmark counter
increase number of times this landmark has been seen
landmark jacobian and measurement covariance
likelihood of correspondence for a landmark
difference between measurement and expected measurement
Initialise new landmark
Update Landmark
dh_dlandmark (2x2 jacobian for landmark)
if likelihood > threshold
if likelihood < threshold
Measurement function h() calculates expected landmark
range and bearing of landmark from position

Figure 7.4: FastSLAM algorithm
The green blocks represent functions that are called for each particle. The blue blocks represent the FastSLAM algorithm itself with the predict, correct and resample steps. The pink blocks represent incoming information while the algorithm is active.

Future Work

These suggestions for future work for Monash Motorsport are from the perspective of other people who are involved in the Autonomous Perception section. The sug-gestions include looking into different SLAM algorithms to make sensor fusion faster in real time and looking into simulation methods so that perception algorithms can be more easily tested. The suggestions are as follows:

- Other SLAM algorithms
  - Particle-based filters such as FastSLAM because it may be more robust than the Extended Kalman Filter SLAM. Particle filters are appropri-ate for when the vehicle faces hard data association problems, where the vehicle can be globally uncertain and might choose from a number of hypotheses about where it could be in the environment. FastSLAM adds computational complexity and it is more difficult to develop. How-ever, with EKF SLAM implemented, this can be used to develop Fast-SLAM which consists of many small EKF calculations. The downsides of fastSLAM are potential lack of diversity in particles if there is not enough noise in the system to diversify them, or this can be done artifi-cially. Particle deprivation can be avoided as well as decreasing required computations by only resampling when the car is moving, or when a new observation has been made. Low-variance sampling can also help with particle deprivation, where if all samples have the same importance weight, no samples are lost. FastSLAM still has a higher computational requirement than EKF.
  - Modular EKFs which involve stitching of smaller EKFs, so that EKF does not have to do a full update when it is reaching the end of the track in which case it might have stored 300 landmarks. The full update can be slow and ROS messages occasionally get missed upon loop closure, especially when there are high frequency sensors such as IMU which pro-vide data at 200Hz. A modular EKF design that only updates the most recent section of the map until it is filled with a specific number of cones before 'starting new' at the next set of cones. When the big orange start cones are seen again, EKF may then perform loop closure.
  - Unscented Kalman Filter which has more complexity and harder imple-mentation but is better for non-linear functions like the one the driverless car has.
  - Sparse Information EKF using sparse matrices, which can be exploited for better computation time at the expense of landmark accuracy.

  In any case, the next implementation should try to use matrix decompositions and avoid doing matrix inverses because these are slow. MMS could implement one type of SLAM for the discovery lap, and another type of SLAM after loop closure that is purely for localisation instead of the current EKF, however, this poses no issues at the moment.

- Improve cone annotation algorithm for faster computations. One method to implement is to exploit the fact that smaller cones will be more distant, so they will only appear in the pixels in the upper half of the image. The template matching could therefore be optimized by only applying matching of smaller templates in the pixels above where the last, largest template was matched.

- Integrate wheel speed sensors along with a motion model into SLAM. That is, if a wheel speed sensor reports zero, it does not mean that the car is stationary but rather the wheels a locked up due to the breaking actuator and the car still moves a significant distance which should be accounted for in SLAM. The model that includes wheel speed sensors therefore needs to take into account the history of the motion of the car and take into account wheel slip.
- Integration of cameras and LiDAR such that they don't both feed cone information to the EKF, but decide on cone position together on a separate node before crowding the EKF with updates. A Kalman Filter or other sensor fusion technique should be used to combine cone information from camera and LiDAR and produce a certain result. However, this may not produce real time results because the camera has a lot of latency while the LiDAR does not.
- Implement and design more visual debugging methods such as having a separate window that displays the EKF covariance matrix where you can see how it updates with each sensor input.
- The noise parameters in EKF SLAM have been tuned to work, so that SLAM does not break. So far, it has been difficult for the team to use its available resources (team member time and testing venue availability) to do tests that will properly quantify GPS noise and drift, and IMU noise and drift. This would be especially useful as the testing vehicle can now actually accelerate unlike the testing trolley that was used in the early onset of this project.
- AMZ verified their car motion model by calculating velocities and comparing them to the IMU's measurement values. Verifications like these would be good to have.
- Experiment to quantify noise on positioning sensors by running SLAM based on dead reckoning alone, hence, no perception updates.
- Getting better cameras that reduce or eliminate rolling shutter and can handle the large vibrations that occur during a fast drive of the vehicle.
- Better and more stable stereo matching algorithm, which depends on team members having access to the necessary and expensive computing units.
- Implement easier platform for testing and simulation: Install a few dedicated office computers for the Autonomous section that have all the necessary hard-ware to run modular simulations of a detection system, and full simulations as if the car was on track with all included vehicle dynamics, all equipped with GPUs to handle stereo camera outputs.
- It is suggested that MMS moves towards a heavily simulated framework to test software, and reserve physical vehicle testing for actuation control. One useful aspect may be to create random track generation with all the parameters de-fined in the rules such as no longer than 80m straights, constant turns less than 50m diameter, hairpin turns greater than 9m outside diameter, chicanes, de-

creasing radius turns, etc. A full final-year-project could be dedicated to this. The methods previously suggested in meetings with the supervisor include

- Start with a circular track and then randomly deform it, eg. apply Gaussian filters to push corners in and other corners out.
- Use Fourier descriptors by regarding each x, y point of the track as x+iy, add harmonics to deform the circle into an ellipse and add other shapes. By limiting the number of harmonics, you limit sharp corners.
- Using GANs (generative adversarial network) to create fake tracks.

7.1 Supporting Resources

Camera Annotating, C++ code https://bitbucket.org/geosov/camera_annotating/SLAM, C++ Code https://bitbucket.org/mms-driverless/mms_slam/master/

Editions to LabelImg, Python code https://bitbucket.org/geosov/labelimg_ auto/

Video Link: www.youtube.com

References

[1] J. Cressell and I. Törnberg, “A combined approach for object recognition and localisation for an autonomous racecar,” 2018.

[2] T. D. Frøysa, “Perception for an autonomous racecar,” *NTNU Master’s Thesis*, 2018.

[3] N. B. Gosala, A. Bühler, M. Prajapat, C. Ehmke, M. Gupta, R. Sivanesan, A. Gawel, M. Pfeiffer, M. Bürki, I. Sa, R. Dubé, and R. Siegwart, “Redundant perception and state estimation for reliable autonomous racing,” *CoRR*, vol. abs/1809.10099, 2018. [Online]. Available: http://arxiv.org/abs/1809.10099

[4] H. B. Mitchell, *Multi-Sensor Data Fusion: An Introduction*, 1st ed. Springer Publishing Company, Incorporated, 2007.

Appendix

The vertical resolution of the LiDAR matters because a better resolution means we can detect cones at a greater distance, which will ultimately allow the vehicle to make decisions faster about a racing line.

Realistically the LiDAR might detect cones from a position above the ground (see Fig. 8.1) because it is mounted as such on the car. The LiDAR mount was specif-ically designed to hold the LiDAR such that its centre is at half the height of the cones. The figure below is the ideal case where the LiDAR will detect a cone at the maximum distance that it can achieve.

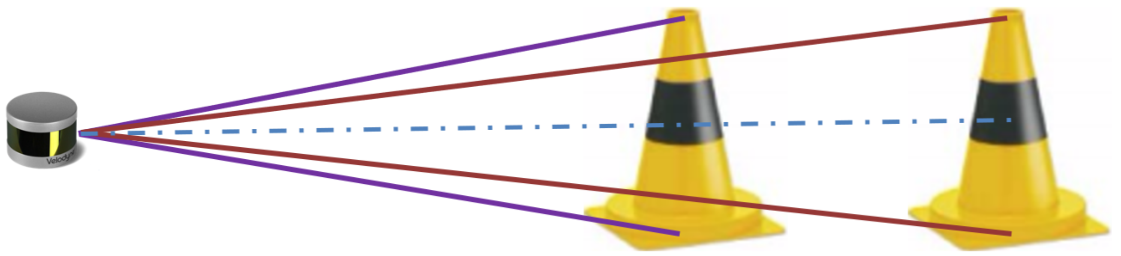


Figure 8.1: Actual height of LiDAR as it detects cones

Calculations using trigonometry will achieve the same result if the calculations are done by pretending that the LiDAR is actually at ground level, as shown in Fig. 8.2.

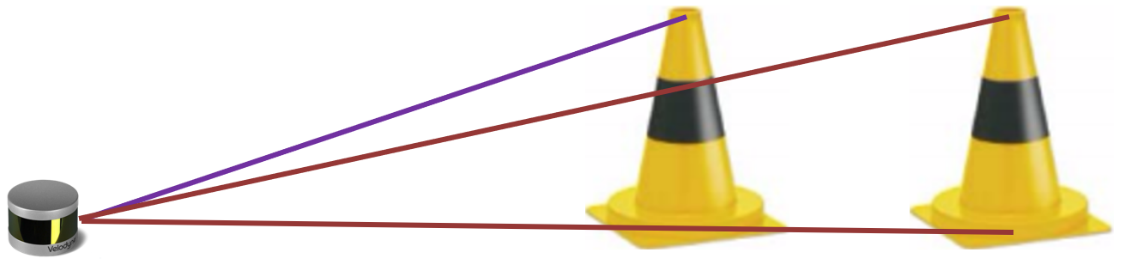


Figure 8.2: Image to aid calculation of distance given vertical resolution

	Velodyne Puck Hi-Res2	Velodyne VLP-16
Vertical resolution:	1.33°	2°
Maximum Cone detection distance achieved:	14m	9.3m

Velodyne Puck Hi-Res can theoretically detect cones at a distance of 4.7m further than the VLP-16.