



# Indian Institute of Technology Kanpur

## IGVC Design Report

May 15, 2018



---

### FACULTY ADVISOR CERTIFICATION:

I hereby certify that the design and development of the vehicle "Daksh", described in this report is significant and equivalent to what might be awarded credit in a senior design course. This is prepared by the members of Team IGVC IITK under my guidance.

*mangal*

**Dr. Mangal Kothari**

Department of Aerospace Engineering

Indian Institute of Technology Kanpur

Email: [mangal@iitk.ac.in](mailto:mangal@iitk.ac.in)

# Table of Contents

1	Team Organization and Design Process.....	1
1.1	Introduction.....	1
1.2	Team Organization.....	1
1.3	Design Process.....	2
2	Innovations in Vehicle Design.....	3
2.1	Mechanical Innovations.....	3
2.2	Electrical Innovations.....	4
2.3	Software Innovations.....	4
3	Description of Mechanical Design.....	5
3.1	Chassis.....	6
3.2	Sensor Mounts.....	6
3.3	Suspension System.....	6
3.4	Weather Proofing.....	7
4	Description of Electrical System.....	7
4.1	Actuators and Control.....	7
4.2	Sensors.....	8
4.3	Computers.....	8
4.4	E-stop and Status LED Implementation.....	8
4.5	Power System.....	9
5	Description of Software Strategy.....	9
5.1	Computer Vision.....	10
5.2	Simultaneous Localization and Mapping.....	12
5.3	Motion Planning and Control.....	13
5.4	Interoperability Profiles.....	14
6	Failure Modes and Resolutions.....	14
7	Miscellaneous.....	15

# 1 Team Organization and Design Process

## 1.1 Introduction

The IGVC team from Indian Institute of Technology, Kanpur was conceived by the members (from the Y13, Y14, Y15 and Y16 batches) of the institute's earlier Robocon<sup>1</sup> team, in October 2016. In order to gain a good enough background in programming and algorithms for developing a fully autonomous vehicle, there was a month-long learning phase. During this phase, the team members enrolled in various online courses on topics such as computer vision, SLAM, motion planning, etc. To focus a majority of the workforce into these areas (rather than manufacturing), it was decided to use a NEX Robotics<sup>2</sup> OxDelta robot, purchased by the Intelligent Guidance & Control Laboratory at our institute.

The robot was initially a 4-wheeled, skid-steering vehicle with pre-installed motors, encoders, a low-level controller and supporting power electronics. Over the course of time, several modifications have been made to the mechanical and electrical systems and a complete software stack has been developed for autonomous navigation. The team will now be participating for the first time in IGVC.

## 1.2 Team Organization

Recruitment of new team members happens every year through a screening test, followed by an interview conducted by existing members. Knowledge is propagated from senior batches to junior batches through lectures and mini-projects. Work is mainly divided into five modules – computer vision, localization & mapping, motion planning & control, electrical systems, and mechanical design. Members choose their area of work based on previous experience, interests, coursework, and department. Because of the multidisciplinary nature of work, the team members are chosen from diverse academic backgrounds including both engineering and science majors from undergraduate as well as postgraduate curriculums. For IGVC 2018, the team organization is as follows:

Name (Department, Batch)	e-mail (@iitk.ac.in)	Role in Team
Prof. Mangal Kothari (AE)	<a href="mailto:mangal@iitk.ac.in">mangal</a>	Faculty Advisor
Aalap Shah (AE+EE, Y15)	<a href="mailto:aalap@iitk.ac.in">aalap</a>	Team Leader, SLAM, Motion Planning & Control, Mechanical Design
Hemanth Bollamreddi (EE, Y15)	<a href="mailto:blmhemu@iitk.ac.in">blmhemu</a>	Team Leader, SLAM, Interoperability Profiles, Electrical Systems
Abhishek Yadav (EE, Y16)	<a href="mailto:abhiyad@iitk.ac.in">abhiyad</a>	Localization, Computer Vision
Aditya Rohan (MSE, Y16)	<a href="mailto:raditya@iitk.ac.in">raditya</a>	Localization, Computer Vision
Ashish Kumar (AE, Y16)	<a href="mailto:ashikum@iitk.ac.in">ashikum</a>	Mechanical Design
Utkarsh Thukral (MTH, Y16)	<a href="mailto:utkarsh@iitk.ac.in">utkarsh</a>	Computer Vision
Vaibhav Agarwal (PHY, Y16)	<a href="mailto:vaibag@iitk.ac.in">vaibag</a>	Localization, Mechanical Design
Vardhan Gupta (ME, Y16)	<a href="mailto:vardhang@iitk.ac.in">vardhang</a>	Mapping, Mechanical Design
Deep Parikh (EE, Y16 PG)	<a href="mailto:deepj@iitk.ac.in">deepj</a>	Mentor - Control
Harsh Sinha (AE+EE, Y14)	<a href="mailto:harshsin@iitk.ac.in">harshsin</a>	Mentor - Computer Vision

It is worth noting that several members were involved in the earlier development of this vehicle during the year of 2016-17. Their contributions have been duly acknowledged at the end of the report.

<sup>1</sup> ABU Robocon is a popular Asia-level Robotics competition.

<sup>2</sup> NEX Robotics is a robot manufacturing company based in Mumbai, India.

### 1.3 Design Process

Some of the key concepts that motivated the design process followed by Team IGVC IITK have been highlighted below:

- **Modularity** – One of the major technologies enabling modularity within our platform is ROS (Robot Operating System<sup>3</sup>). ROS allows the entire software stack to be divided into separate packages, each of which consists of *nodes* that can communicate with each other either by *publishing* data on *topics* or by *subscribing* to them. This presents advantages such as:
  - natural work division within the team,
  - ability to replace subsystems during development, treating the rest as a black-box,
  - ability to use open-source packages for parts of the system as well as to share the other parts of the system with the open-source community.

Modularity is not just limited to software. All sensors are connected to the onboard computers via USB cables, and therefore, it is possible to replace/remove sensors at will. The localization algorithm automatically adapts to this change, but it might be necessary to modify drivers in some cases of replacements.

The team has gone so far as to provide even mechanical modularity – the robot can switch between 4-wheel and 3-wheel modes – the advantages of which have been explained in a later section.

- **Robustness** – For each subsystem, often various strategies were explored and implemented. They were first tested on simulations and recorded data. Once the performance achieved for one or more of these strategies was satisfactory, they were combined with other subsystems and tested on real-world environments similar to IGVC (made by the team). Any parameters involved in the software were then tuned for better performance. At the end of these tests, the best performing strategy/implementation was chosen. Besides accuracy, computational efficiency of code was also given importance since the vehicle’s reaction time directly depends on it. All test data has been recorded for future reference.
- **Open-Source and Documentation** – Since a lot of open-source software was used in the development of the vehicle, it was decided that it is best to make the software developed by the team open-source as well. Our complete source code has been made available through GitHub repositories<sup>4</sup>, which are continuously updated and well-documented. Making the code open-source also ensures that team members adhere to standard coding practices more stringently as well as focus on code readability since all of their work is publicly visible. Moreover, a lot of test data has also been made publicly available on SourceForge.<sup>5</sup> The team also actively maintains a YouTube channel<sup>6</sup> where video recordings and screencasts of all these tests can be seen. When combined, all of this amounts to excellent documentation and hence, reusability of our software. A GPL-3.0 license is applied to all software developed by the team in order to promote open-source development (the license requires all works using this code also to be made open-source).

---

<sup>3</sup> ROS is a popular open-source tool that provides a C++ library and several packages for robotics development.

<sup>4</sup> All of these repositories are available at <https://github.com/igvc-iitk>, our team’s GitHub organization.

<sup>5</sup> This test data is available at <http://sourceforge.net/projects/igvc-iitk-data>, our team’s SourceForge project.

<sup>6</sup> The YouTube channel, IGVC IITK can be found at the following link:

<https://www.youtube.com/channel/UCrQoKMEBySfXPEqXGk-CZ8A/videos>.

## 2 Innovations in Vehicle Design

### 2.1 Mechanical Innovations

- **Switchable Vehicle Dynamics** – Since the vehicle was initially a 4-wheel, skid-steer, differential drive model, it could not perform a finite-radius turn without slipping of the wheels (this is geometrically impossible). For indoor environments, the surface friction is often uniform, and therefore the distance traveled, as calculated from the odometry is accurate enough. However, for outdoor environments, the surface friction can no longer be assumed as uniform. Thus, the slipping of each wheel is different and not easily predictable (which notably complicates control too). Now, the distance traveled, and especially the heading, as calculated from the odometry is not accurate enough for localization. Two alternate solutions were implemented to overcome this problem:
  - Many visual odometry techniques were explored and tested before zeroing down to ORB-SLAM 2.<sup>7</sup> Fusing odometry from ORB-SLAM 2 with other sensor data ensures that localization is accurate despite slipping of the wheels.
  - The front wheels were removed, and a single, smaller, freely rotating caster wheel with a suspension was added at the front. This effectively changes the robot's vehicle dynamics to a unicycle model. Outdoor wheel odometry drastically improves after this change (as there is no slipping), eliminating the need for visual odometry.

Thus, the robot's vehicle dynamics can be switched at will. This gives a nice tradeoff. In 4-wheel mode, the robot can navigate more harsh terrains, climb steeper inclines, and have very low vibrations (since the wheels have air-filled tubes) at the cost of higher computational load due to ORB-SLAM 2. In 3-wheel mode, the vehicle gives accurate odometry without additional computational resources and is easier to control but can only work in less harsh terrains (such as the IGVC course).

- **Adjustable but Rigid Camera Mount** – The field of view of a camera depends on its height and orientation. So, having a camera mount that can be adjusted for both is desirable. However, simple ball joints such as those usually found in webcams can change orientation on experiencing jerks. So, a rigid-yet-adjustable mount was designed and fabricated from nylon sheets using precision water jet manufacturing. As seen in figure 1, the height and orientation can be adjusted simply by removing and re-inserting two bolts. The orientation can be set to fixed pitch angles from 0° to 60° in increments of 10°.



Figure 1 - Adjustable but Rigid Camera Mount

<sup>7</sup> See 'ORB-SLAM2: an Open-Source SLAM System for Monocular, Stereo and RGB-D Cameras' by Raul Mur-Artal.



## 2.2 Electrical Innovations

- Seamless Manual (RC) to Autonomous Switching** – It is desirable to drive the vehicle manually to a location before testing (as opposed to carrying it there) or to control the vehicle manually in the middle of a run. Thus, an RC transmitter is made to bind to a receiver attached to an Arduino Mega, which passes on the commands via serial communication to a ROS node running on the onboard computer. A particular knob on the remote can be turned on/off to switch the mode from manual to autonomous at will. When in autonomous mode, the robot takes commands from an automatic controller which follows the path given by the path planner. When in manual mode, these commands are ignored, and the RC controller takes precedence.

Thus, it is possible to take over control in the middle of an autonomous run, then drive the robot for a while and then have it switch back – all without any hardware changes. Notably, localization and mapping are not interrupted by the switch.

## 2.3 Software Innovations

- Dynamic Inverse Perspective Transform** – For mapping lanes, it is necessary to first transform the camera image to a top view (or birds-eye-view). Almost all IGVC teams use a pre-calibrated intrinsic and extrinsic transformation matrix for this purpose.<sup>8</sup> The transformation equations used for this purpose are:

$$\begin{bmatrix} x' \\ y' \\ w \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

$$x = \frac{x'}{w}, y = \frac{y'}{w}$$

Here,  $X, Y, Z$  are the real-world coordinates, and  $x, y$  are the image coordinates. The first matrix (intrinsic) is calibrated in the usual manner using a chessboard with known dimensions.

However, for our vehicle, the second matrix (extrinsic) is dynamically calculated. An IMU is attached on the top of the camera for this purpose. The pitch and roll readings from the IMU and known geometric information (the height of the camera) are used to calculate the extrinsic matrix. This allows for a dynamic inverse perspective transform. The advantage of this method is that the top view image is correct even when the vehicle is on an incline or when the camera is shaking due to excessive vibrations. This is illustrated in figure 2.

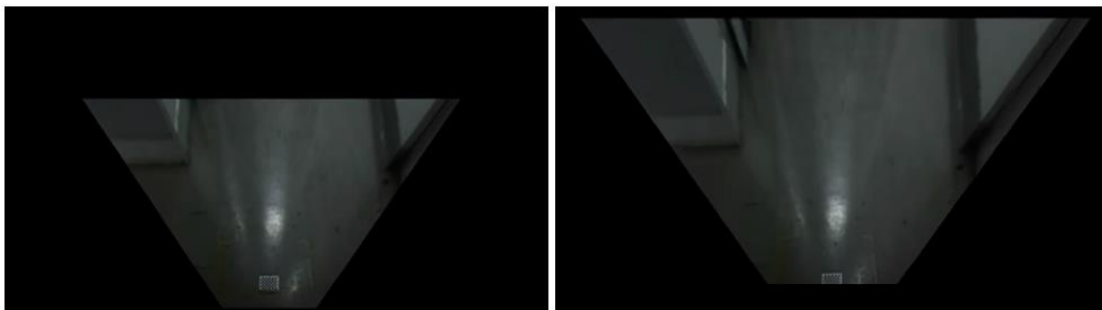


Figure 2 - Dynamic Inverse Perspective Transform (In the image on the left, the camera is pointing lower.)

<sup>8</sup> This is as per the earlier years' design reports.

- Visual Calibration of Laser Scanners** – A ROS node was developed for obtaining the calibration between the robot’s front and rear laser scanners. The ROS node publishes a transform between the two frames that can be updated in real-time using multiple sliders. Then, RViz<sup>9</sup> was used to visualize the data points from both laser scanners in one window. The sliders were then adjusted until the two scans matched. This method can be used for rapid calibration between any two sensors with an overlapping field of view (even stereo cameras).

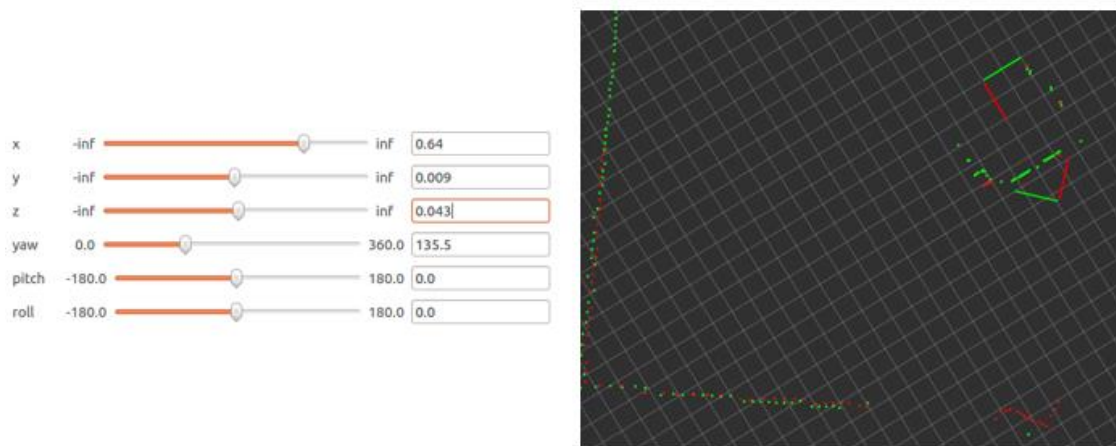


Figure 3 - Green points are from the front scanner, red points are from the rear scanner.

- LiDAR-based Safety braking** – The robot’s driver has been modified in such a way so that it subscribes to topics on which the laser scans are being published. Then, when the robot is being issued control commands, it continuously checks whether the command may lead to a possible collision and the decision to actually pass on that command to the low-level controller is only made if it is safe to do so. In autonomous mode, this prevents collisions that occur when an obstacle suddenly appears in the field of view (and there hasn’t been enough time for a map update followed by a newly planned path). In manual mode, this serves as a great safety feature as it prevents the human controller from accidentally crashing the robot into an obstacle/wall.
- Mission Control Dashboard** – An rqt-based GUI<sup>10</sup> has been developed in order to summarize all the information about the robot’s current state. Since this is very helpful during analyzing failure modes, it has been covered in greater detail in the section on failure modes.

### 3 Description of Mechanical Design

Despite the robot being manufactured by a company, there were several add-ons and modifications made by the team. One of the major changes was in the vehicle dynamics itself, as mentioned earlier. The other significant change was the inclusion of a spring-based suspension system, which became necessary after the addition of a caster wheel at the front. Besides, mounts were prepared for each sensor.

<sup>9</sup> RViz is a 3D environment visualization tool compatible with ROS. We have also used RViz within the mission control dashboard for visualizing all localization and mapping data at once.

<sup>10</sup> rqt is an open-source framework for GUI development with ROS.

### 3.1 Chassis

The vehicle's chassis is a rigid iron frame covered with steel plates that enclose the entire electrical system except for the sensors, antennas, etc. Two plates are rigidly attached at the front and the rear, using hexagonal aluminum supports – one for the caster wheel and one for the rear laser scanner. Similarly, a stand is attached in the middle, which is used for the camera mount. A rendered image of the CAD model of the entire chassis, along with the sensor mounts, sensors, and status LED can be seen in figure 4.

### 3.2 Sensor Mounts

The camera mount's design has been addressed earlier. Both laser scanners are directly attached using bolts. Additionally, a mount for the IMU and GPS has been manufactured, keeping in mind that:

- magnetic interference from the motors at the location of the IMU is low, and
- the GPS module is not covered from above by any object.

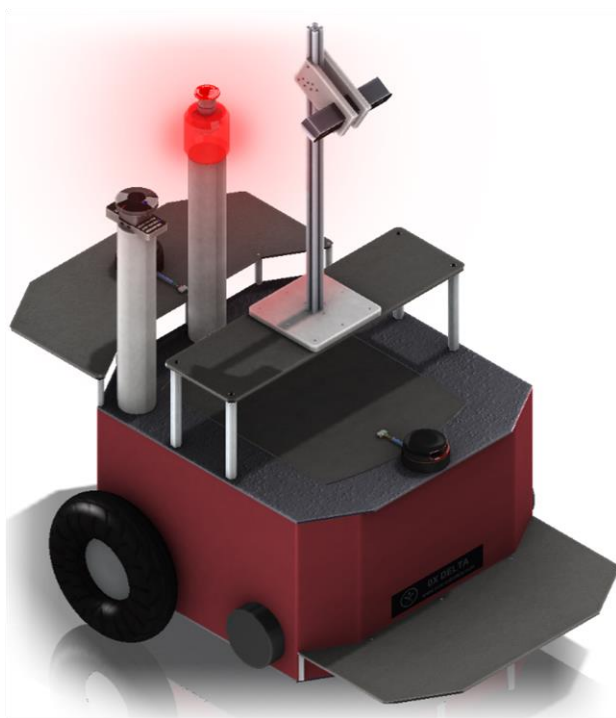


Figure 5 – CAD Model of “Daksh” with All Sensors



Figure 4 – Suspension System (compressed state at bottom)

### 3.3 Suspension System

When in 3-wheel mode, the vehicle requires a suspension system for the caster wheel. This is because it has a hard plastic surface – unlike the other two wheels, which have air-filled tubes (in other words, they are pneumatic wheels). Without the suspension system, the camera encounters large vibrations, which adversely affect mapping.

As seen in figure 5, the suspension system consists of four half-threaded M8 Allen bolts that are rigidly attached to the caster wheel mount and allow for only vertical motion at the points where they are attached to the chassis. Springs are positioned around these bolts so that the vertical motion of the caster wheel is controlled by them. For calculating the required spring constants, the robot was weighed, and the relative distances between the springs, front point-of-contact, center-of-mass, and rear point-of-contact were measured. Then, the resulting force-balance and moment-balance equations were solved to find the spring constants.



### 3.4 Weather Proofing

The vehicle's chassis encloses as much of the electrical system as possible. Only the sensors (laser scanners, stereo camera, IMU, and GPS) are placed outside the chassis, which is unavoidable. The top of the chassis is covered by a lid, which has holes to allow for the passage of the cables connected to the sensors. Thus, most of the electronics are protected from dust and light rain. For the protection of the sensors, a lightweight frame made out of GI wire and wrapped in polythene can be optionally attached at the top of the camera mount. Since this is the highest point on the vehicle, it covers all the sensors. The chassis itself is corrosion resistant. Finally, the plastic surface of the caster wheel has been covered in a bicycle-tire-grip to improve wear resistance.

## 4 Description of Electrical System

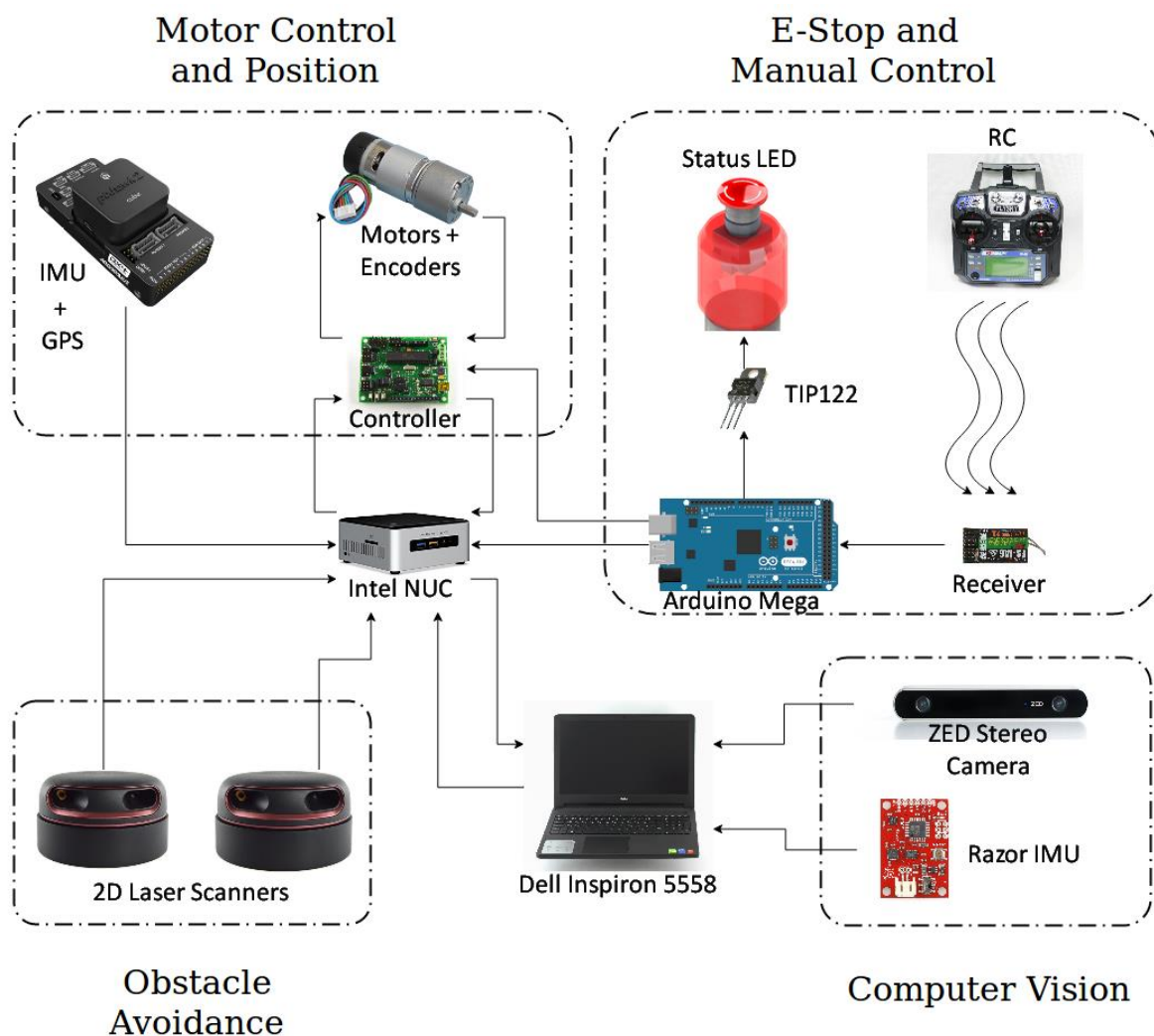


Figure 6 - Electrical Subsystem Overview

### 4.1 Actuators and Control

The actuators along with their integrated encoders are provided by NEX Robotics. The input to the motors is given by a low-level controller which in-turn takes velocity command from the computer (Intel NUC) via a USB cable. The encoder ticks are used by the controller to control each motor separately and are also provided to the computer for odometry.

## 4.2 Sensors

The team has attempted to keep the sensor budget as low as possible, while not sacrificing much in aspects such as perception range or localization accuracy. Sensor specifications are as follows:

- **Laser Scanners** – Two RPLIDAR<sup>11</sup> A2 2D Laser Scanners are used. The range of each scanner is 8 m (26 ft), while the angular resolution is 1° when operating at 4000 samples/sec.
- **Stereo Camera** – A ZED<sup>12</sup> Camera is used with a custom mount instead of the default one. It can output two 720p video streams at 60 Hz if a USB 3.0 port is used. The depth information obtained using the disparity map is accurate up to 20 m (66 ft).
- **Encoders** – All four integrated encoders have an accuracy of 2048 counts/revolution.
- **IMUs** – A Pixhawk<sup>13</sup> 2 Cube is used which has two IMUs. These are internally fused on the Pixhawk itself using an EKF algorithm. A Razor IMU manufactured by SparkFun is mounted on top of the camera and used for dynamic inverse perspective transform (discussed earlier).
- **GPS** – A ‘Here GNSS’<sup>14</sup> is connected to the Pixhawk for obtaining globally-referenced position data required for waypoint navigation. It supports GPS, Galileo, GLONASS, and BeiDou.

## 4.3 Computers

Ultimately, almost all sensor data goes into the computers for processing. For good performance, it is generally recommended to isolate the computationally intensive, high-throughput vision pipeline from low-latency systems such as SLAM, motion planning, and control. A Dell Inspiron laptop is used for running the vision pipeline, while an Intel NUC is used to run the remaining processes. The two computers are connected via Ethernet. The ROS master on the NUC manages this communication.

Name	Purpose	Processor	RAM	Dedicated GPU
Intel NUC	SLAM, Planning, Control	i5 6th Gen	8 GB DDR3	None
Dell Inspiron 5558	Vision Pipeline	i5 5th Gen	4 GB DDR3	1GB Nvidia 920M

## 4.4 E-stop and Status LED Implementation

Two knobs on a 2.4 GHz RC transmitter can be used as switches - one for the wireless e-stop and one for seamless manual to autonomous mode switching (discussed earlier). The data received by the receiver is present as a PWM output at its pins. An Arduino Mega decodes this PWM signal and uses serial communication to send the data to a ROS node, which publishes it for manual control. In case of e-stop, the Arduino puts a HIGH (5 V) on the kill pin of the low-level controller which then powers down the motors. The mechanical e-stop directly cuts off the connection between the power source and the motors.

LED strips placed below the e-stop indicate the mode of operation. In autonomous mode, the LEDs start flashing, while in manual mode they display a solid red color. When the robot is powered off or when the e-stop is used, it becomes dark/off. TIP122 is a Darlington transistor by ON semiconductors. The LED Strips work at a voltage of nearly 22.4V while all the digital systems use 5V logic. Hence TIP122 is used as an electronic switch to change the state of the LED Strips.

<sup>11</sup> RPLIDAR A2 is a low-cost 2-dimensional laser scanner model manufactured by SlamTec Co.

<sup>12</sup> The ZED camera is a stereo camera manufactured by Stereolabs Inc., intended specifically for SLAM usage.

<sup>13</sup> Pixhawk is a well-known, low-cost flight controller hardware project.

<sup>14</sup> ‘Here GNSS’ is a low-cost GPS sensor compatible with the Pixhawk, manufactured by ProfiCNC.

## 4.5 Power System

The Robot is powered up using two batteries. A 6-cell, 22.2 V LiPo battery of 10,000 mAh capacity is used to power up the actuators and LED lights. A 4-cell, 14.8 V LiPo battery of 10,000 mAh capacity powers up the NUC and all the attached electronic peripherals (2 laser scanners, 1 Pixhawk, 1 GPS, 1 Arduino and 1 RC receiver). The laptop computer and the sensors attached to it (1 stereo camera, 1 IMU) are powered by the internal battery of the laptop. After a complete recharge, the 6-cell battery can power the robot for 2.4 km (1.5 miles), when running at 1 m/s (3.3 ft/s) without reaching critical voltage. As the components powered by the 4-cell battery do not consume much power, it discharges at a much lower rate than the 6-cell battery.

## 5 Description of Software Strategy

Software development was the main focus of the team. The overall architecture was incrementally developed and continuously updated throughout our design process. The top-level software architecture is described in figure 7 and explained piece-by-piece in the rest of this section. The three major modules are Computer Vision, SLAM, and Motion Planning & Control.

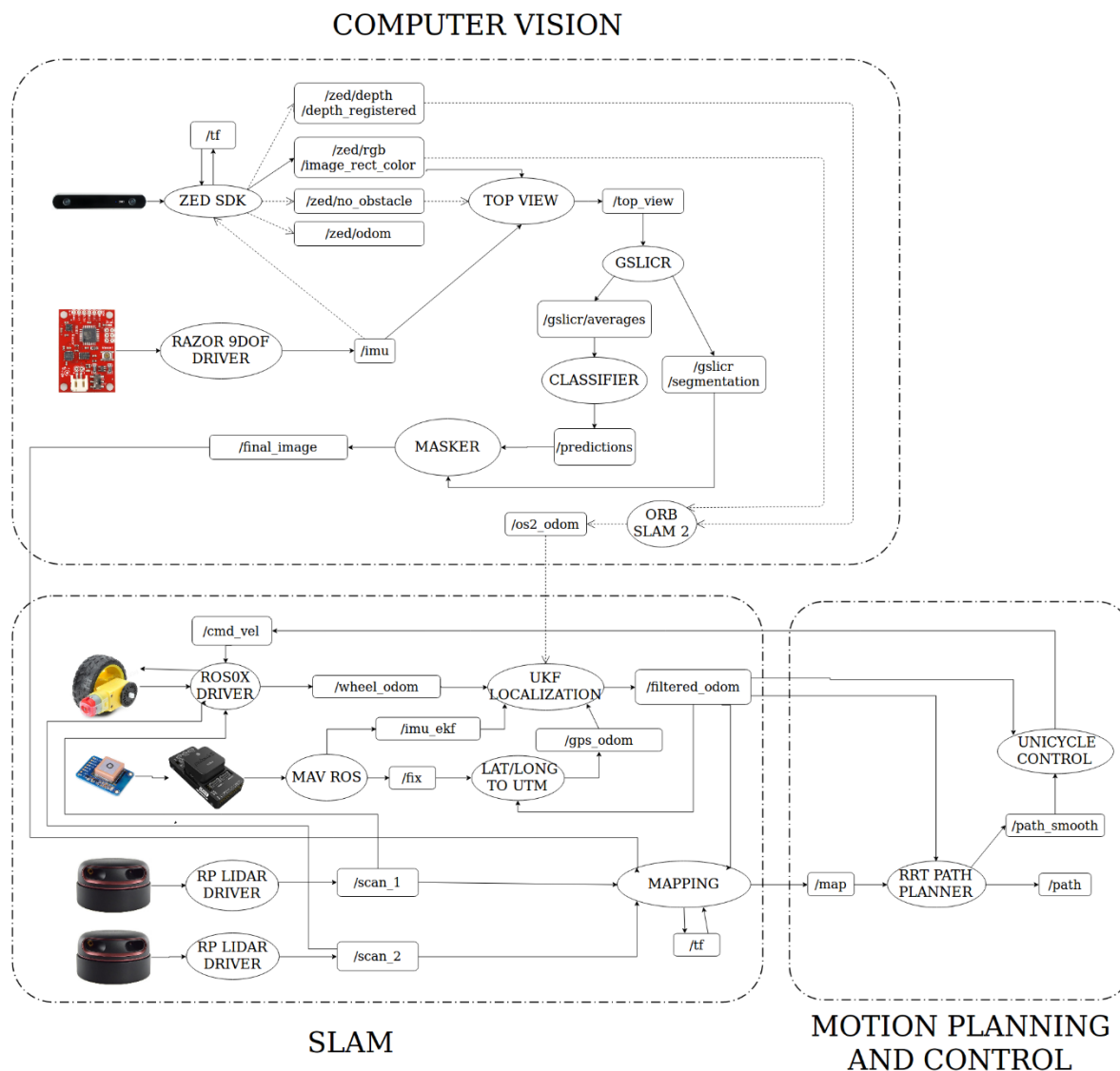


Figure 7 - The overall software architecture is split into 3 major modules. Ellipses denote ROS nodes (programs) and rectangles denote ROS topics (data). The dotted lines in the computer vision section represent optional parts.

## 5.1 Computer Vision

This aspect of our software architecture involved the most experimentation. The inputs are the two video streams from the stereo camera and orientation data from an IMU. For IGVC, the end goal of the vision pipeline is to detect the lanes (and optionally output visual odometry). The IGVC environment can be treated as a combination of lanes, grass, and obstacles (everything above ground). The core reason as to why this task is so challenging is that there are environmental variations in lighting conditions, shadows, and grass textures. The subtasks involved can be classified into the following categories:

- **Visual Odometry** – It is required only when there is significant wheel-slipping. This part is separate from the rest of the vision pipeline. Initially, we tried to estimate poses from optical flow, but the drift was very high for our implementation. So we performed a comparative study of open-source implementations of stereo visual odometry algorithms such as PL-SLAM, DSO, ORB-SLAM 2, etc. ORB-SLAM 2 was selected because of its speed and superior performance, specifically in IGVC-like environments. Note that despite being a SLAM algorithm, it does not contribute to mapping in our use case, since ORB features are of no interest in IGVC.
- **Transformations** – Two types of image transformations are performed. The first is the spherical distortion correction, done using the camera’s SDK itself. The calibration for this was performed using a chessboard with known dimensions. The rectified image is then optionally checked for obstacles before performing the dynamic inverse perspective transform (discussed earlier). The relative ordering of this step within the vision pipeline is crucial. It must be performed after any obstacle detection algorithms because they get distorted after the inverse perspective transform. But it must also be performed before the actual lane detection because, in a perspective view, lanes become thinner as they go farther (which is not optimal for most lane detection algorithms).
- **Obstacle Detection** – Identifying obstacles using the camera is not mandatory in itself (the laser scanners are more accurate). However, it still serves as an important pre-processing step for better lane detection as it leaves only lanes and grass in the image. One approach used for obstacle detection was to project the laser scanner points onto the image and then use a watershed transformation in OpenCV to eliminate the entire obstacle. However, this method did not work well for striped obstacles.

Therefore, a more robust method was used. The stereo camera’s driver computes a disparity map by matching the pixels of the images from the left and right cameras. This is used to obtain depth information, which essentially gives a 3D point cloud. We have modified the driver so that it can use the position of the camera relative to the ground to classify all points above the ground as obstacles. The results can be seen in figure 8.



Figure 8 - Robust Obstacle Detection Using Depth from Stereo

- Superspixel Segmentation** – Since we are using a neural network based classifier for lane detection, we first downscale the image for real-time performance. The image is downscaled from 1280x720 to 80x45 (each new pixel, referred to as a superspixel from now on, is almost equal to the thickness of one lane). For such a large amount of downscaling, an efficient yet information-preserving method is required. We have used an open-source GPU accelerated implementation of SLIC<sup>15</sup>, known as *gSLICr* and developed a ROS wrapper for it. Downscaling using segmented superspixels rather than the usual averaging over each 16x16 cell preserves more color information, as seen in figure 9.

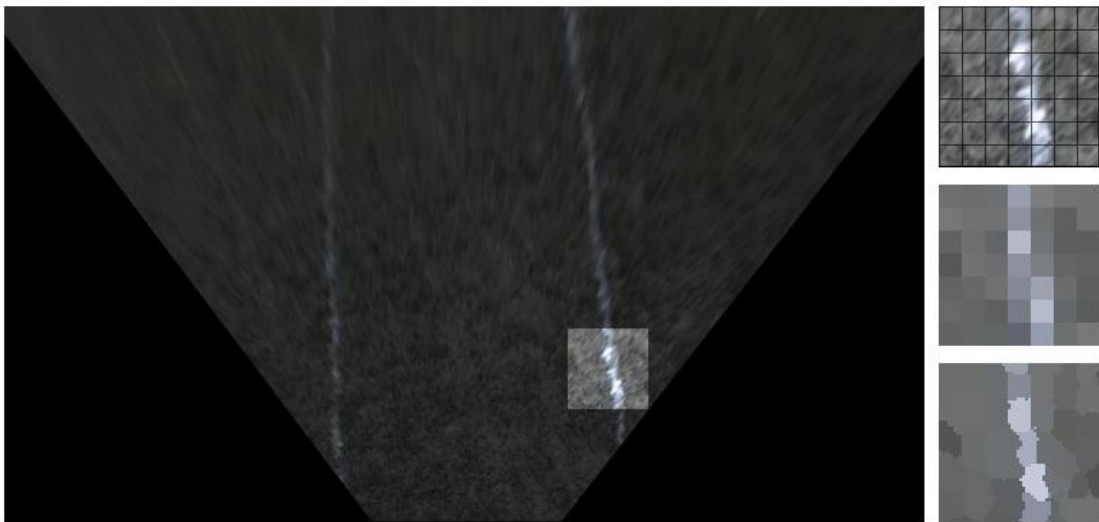


Figure 9 – Box-mean downscaling (center) leads to loss of white color as the grid boundaries (top) rarely coincide with the lane boundaries. *gSLICr* downscaling (bottom) preserves color as the boundaries are obtained by clustering.

- Lane Detection** – One of our earlier approaches to lane detection involved obstacle removal, followed by grass elimination using adaptive thresholding, followed by a linear Hough transform on the pixels that remained (mostly lanes). However, thresholding-based methods do not generalize well to a variety of environments. Besides, the lanes are not always straight lines and require higher order Hough transforms, which are computationally expensive.

For better accuracy, it was decided to use a CNN (convolutional neural network) for lane detection. CNNs are a popular choice for image classification because their architectures force them to relate spatially closer pixels over other pixels (unlike a fully-connected NN). Training a CNN meant that a labeled dataset had to be prepared. A program was developed for this purpose, wherein superspixels were initially labeled using a threshold and then the user could flip the erroneous labels by clicking on the superspixel.

Once trained, the CNN takes a 7x7 grid of superspixels centered at the target superspixel and outputs a predicted label for the target. Note that it only required 3 hidden layers since a lane is a low-level feature. Some details in relation to training are as follows:

- A single image could produce 3600 7x7 grids, each of which could be further flipped and rotated to generate 8 training examples per grid. Thus, a dataset of about 300 images of IGVC-like environments taken at different times of the day was sufficient.
- The architecture was decided by training and testing on MATLAB. An accuracy of 98% was achieved for the architecture shown in figure 10. Keras<sup>16</sup> was used to implement the final architecture as it was simpler to integrate it with our ROS based framework.

<sup>15</sup> See ‘SLIC Superspixels’ by Achanta et al. SLIC stands for Simple Linear Iterative Clustering. It is essentially a fast segmentation algorithm that performs clustering in 5-dimensional space (3 color + 2 spatial dimensions).

<sup>16</sup> Keras is a python based frontend for neural network training and inference. It uses TensorFlow as its backend.



It was found that the CNN was also able to classify lanes without the need for obstacle detection (if trained accordingly). Thus, obstacle detection was made optional (see figure 7) to reduce computational load and battery consumption.

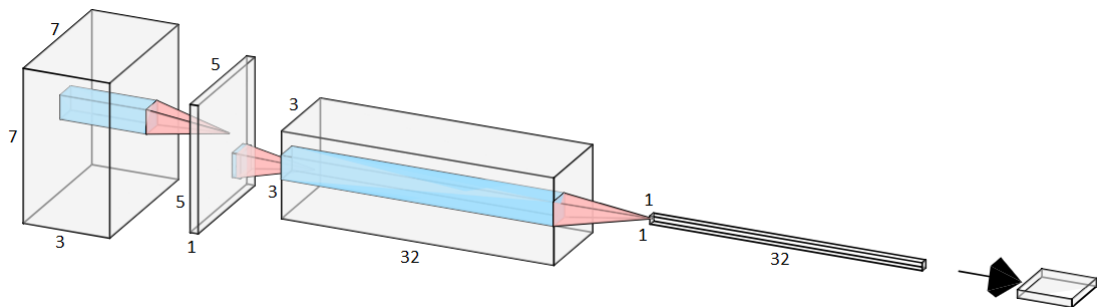


Figure 10 - Architecture Diagram of the CNN Used for Lane Detection

## 5.2 Simultaneous Localization and Mapping

As can be seen in figure 7, most of the nodes in this module are drivers for a particular sensor (encoder, laser scanner, and IMU + GPS). These drivers provide sensor data in a ROS message format. We have made several modifications based on our use case. For instance, the laser scanner driver has been modified to neglect points on the robot's chassis itself. Also, the latitude-longitude data given by the GPS module is converted to UTM coordinates so that it can be used for localization. The non-driver nodes have been explained in detail below:

- **UKF Localization** – An open-source implementation of UKF (Unscented Kalman Filter) based localization, called *robot\_localization*, has been used. The working principle is as follows:
  - A state vector of length 15 is maintained at all times. It consists of position, orientation, linear velocity, angular velocity and linear acceleration – each of which is a vector of length 3.
  - The prediction step is performed at 20 Hz assuming a 3D non-holonomic unicycle model, using the control input and the current state.
  - Sensor data arrives asynchronously from each sensor at about 10 Hz. A correction step is performed for each measurement. If a sensor is disconnected at any point, the algorithm still functions as the measurement update is asynchronous anyway.
  - From wheel odometry, the speed and 2D angular velocity are used. From the IMU, 3D orientation and angular rate (from the gyroscope) is used. IMU acceleration is ignored due to high covariance. The horizontal position from GPS is used but not the altitude.
  - Optionally, 3D position and orientation data from visual odometry is also used.

For outdoor use-cases, the mapping algorithm may not always work well (especially in sparse environments). Therefore, we needed to improve the localization accuracy. This was achieved by experimentally measuring the covariances of each sensor (measurement noise) and the control characteristics by analyzing raw data on MATLAB. The process noise covariance matrix was also tuned. Finally, we were able to repeatedly achieve a localization error of less than 3% for a 500 m (1640 ft) outdoor run, without GPS.

- **Mapping** – With very low localization errors, it is possible to generate a global map even without SLAM. Therefore, we have developed a ROS node that directly uses the sensor-fused odometry and transforms the laser scan and lane data into the global frame, inserting it into a map.

We have also tried using SLAM and performed a qualitative comparison between the two (see figure 11). For SLAM, we have used *Cartographer*, an open-source SLAM package by Google. It uses a gradient descent based squared error minimization approach for local scan-to-map matching. Small local maps are generated this way. It then develops a graph network of these local maps and looks for loop-closure constraints in this global graph in real-time using a slightly more efficient approach for map-to-map matching. In practice, we found that the results were slightly better than our earlier approach to mapping. However, in sparse environments with almost no laser scan data, the algorithm's behavior was unstable, and therefore it was decided not to use it just yet. The team plans to modify the package later for better performance in sparse outdoor environments.

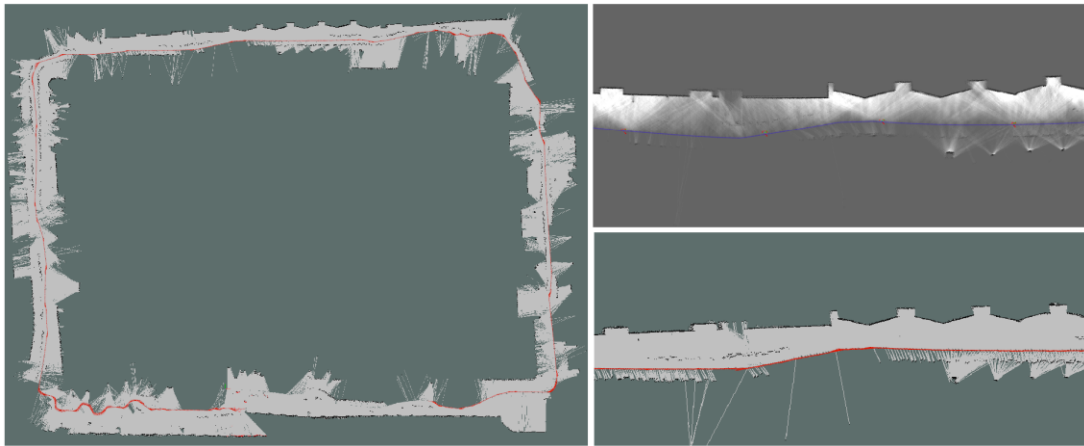


Figure 11 - A map generated within our campus over a 500 m (1640 ft) trajectory using our approach (left). Zoomed in sections of a part of the same map generated using *Cartographer* (right top) and using our method (right bottom).

### 5.3 Motion Planning and Control

- **Path Planning** – Most of the work on path planning was done by the team in the year 2016-17 itself. After testing a range of algorithms such as A\*, D\* Lite, PRM (Probabilistic Road Maps), etc. over MATLAB simulations, it was decided to use a modified version of RRT (Rapidly Exploring Random Tree) implemented from scratch by our team in C++. The RRT algorithm works as follows:
  - A tree is initialized with the current position as the root node.
  - Samples are randomly drawn from free space and connected to the nearest node by a straight line segment. If the line segment completely lies in free space too, a node corresponding to a fixed step size in the same direction is added.
  - The tree is allowed to 'grow' until a fixed number of nodes or iterations are reached.
  - The shortest path is chosen by tracing back a path from the nodes which lie close enough to the goal and comparing their total times.

The reasons why RRT was chosen over other algorithms are:

- It is easier to introduce non-holonomic constraints within the path by simply rejecting those nodes which lead to large turning angles. Actually, the vehicle can even turn on-point, but this adversely affects speed.
- It is fast enough so that we can simply re-run the algorithm every time a new map is generated.

The path planner also performs some additional tasks – the map is converted to a configuration space at the beginning, and the path is smoothed at the end.

- **Control** – For the high-level path following controller, it was decided to use a non-linear control law<sup>17</sup> specifically developed for unicycle model vehicle, rather than a PID controller. Once we developed a ROS node for it from scratch, it was found that tuning this controller was relatively easier. The control law used is:

$$\begin{bmatrix} v_d \\ \omega_d \end{bmatrix} = \begin{bmatrix} K_s \\ u + \frac{v \cos \tilde{\theta} c(s)}{1 - c(s)l} \end{bmatrix}$$

$$u = -k_a v l \operatorname{sinc} \tilde{\theta} - k_b v \tilde{\theta}$$

Here,  $v_d$  and  $\omega_d$  are the output velocity and angular speed given by the high-level controller to the low-level controller. The speed is fixed at a set point  $K_s$  (the controller reduces the speed before sharp turns though).  $c(s)$  is the local curvature of the path,  $\tilde{\theta}$  is the orientation error and  $l$  is the signed distance error.

## 5.4 Interoperability Profiles

The IOP problem is being tackled by using an open source software, JTS (Jaus Tool Set). The IOP problem is broken down into an FSM which is then implemented using the JTS GUI. The GUI further generates the corresponding C++ code. The code is edited for some minor user-based changes like printing some message, initializing a state automatically, etc. The IOP-ROS bridge is being used for interfacing ROS with JTS generated code. An overview of the IOP implementation by JTS is as follows<sup>18</sup>:

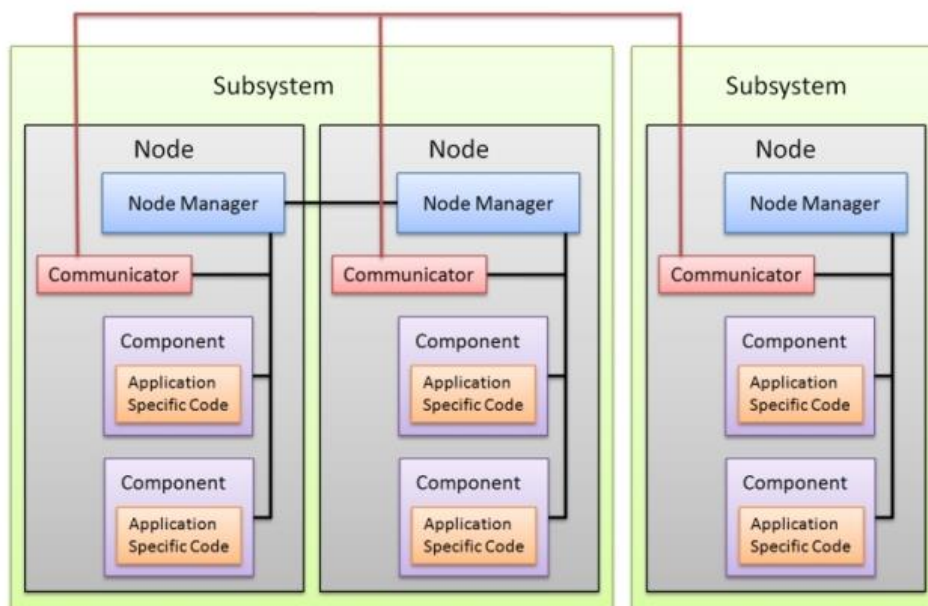


Figure 12 - IOP Implementation by JTS

## 6 Failure Modes and Resolutions

Identification of failure modes is a very crucial component of testing. While some problems can only be debugged by recording the entire data at then analyzing it later, some more common issues can usually be identified at the testing site itself with the help of a GUI. Therefore, the team has developed a Mission Control Dashboard for easy and quick identification of failure points. It shows information about the entire state of all 3 modules, as seen in figure 13.

<sup>17</sup> See ‘Nonlinear Control Design for Mobile Robots’, by C. Canudas de Wit., in The International Journal of Robotics Research.

<sup>18</sup> Image Source: the SourceForge page of the ACTIVE-IST Lab at the University of Central Florida

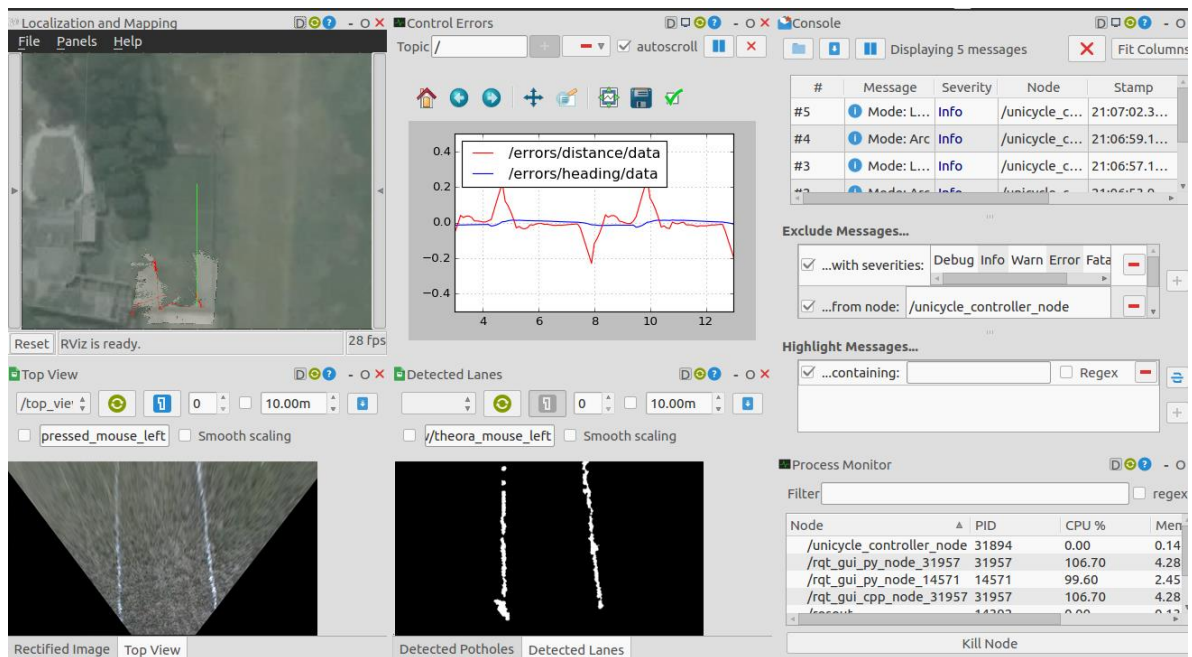


Figure 13 - Mission Control Dashboard. Mapping data is overlaid on top of satellite imagery (downloaded in real time).

Common Failure Modes	Identification/Resolution
General mechanical or electrical failure (loose nuts, broken wires)	Mechanical and electrical toolkit available (contains fastening and soldering tools, plus some spare nuts, bolts, wires and ICs)
Castor wheel wear and tear	Spare tire grip available
Battery drainage	Battery monitor attached to all batteries
GPS fix not found	Incorrect position will be seen on the dashboard
Controller not tuned well	Control errors can be seen on the dashboard
Poor lane detection	Seen on the dashboard, solved by training CNN on new data

## 7 Miscellaneous

- Performance Testing** – Subsystem tests were performed almost every alternate weekend, and all the sensor data was recorded. The same data was then used for testing, debugging and improving our algorithms. Although some simulations were made using MATLAB and Gazebo, testing on real-world data was preferred as it models the errors better. Towards the end, an IGVC-like test track was made for testing. The vehicle can travel at a top speed of 2.1 m/s (6.9 ft/s), but it is driven autonomously at an average speed of 0.5 m/s (1.6 ft/s). It can easily climb inclines of up to 30° with the payload. The reaction time was found to be less than 1 second.
- Cost Estimate** – The total rebuild cost of the robot was calculated to be \$7,500. This includes all the sensors, both onboard computers and the entire mechanical and electrical subsystem. All the software is free and open-source. Throughout the design, lowering costs was a priority.
- Planned Future Improvements** – Besides improving the current system, the team also intends to develop the vehicle beyond the requirements for IGVC. One of the main goals of the team is to develop a culture of robotics related learning within the institute and to develop a low-cost vehicle for an end user. Keeping these points in mind, we plan to experiment with other sensors such as radar and features such as pre-loaded maps, voice control, etc.
- Acknowledgments** – The work done by us would not have been possible without our 2016-17 team members (Animesh Shashtry, Deepak Gangwar, Shubh Gupta, Swati Gupta, Bhuvu Gupta, Hardik Maheshwari, Nishkarsh Aggrawal, Pritesh Kumbhare, Anay Mehrotra, Anav Prasad, Ashish Patel, Siddharth Srivastav, and Yash Chandnani). Open-source repositories and research papers, wherever used, have received due mention within the report.