Indian Institute of Technology Kanpur
IGVC Design Report
May 15, 2019

Team Members

Name (Department, Batch)	e-mail (@iitk.ac.in)
Prof. Mangal Kothari (AE, Faculty Advisor)	mangal
Harsh Sinha (AE + EE, Y14)	harshsin
Aalap Shah (AE + EE, Y15)	aalap
Hemanth Bollamreddi (EE, Y15)	blmhemu
Abhishek Yadav (EE, Y16)	abhiyad
Vaibhav Agarwal (PHY + EE, Y16)	vaibag
Vardhan Gupta (ME, Y16)	vardhang
Apurva Nandan (AE, Y17)	apurvan
Aryan Choudhary (EE, Y17)	aryanc
Faizan Siddiqui (MSE, Y17)	faizan
Kishan Shukla (CHE, Y17)	kshukla
Nitik Jain (ME, Y17)	nitik
Shivam Kumar (EE, Y17)	krshivam
Tushar Singhal (CE, Y17)	tushars

FACULTY ADVISOR CERTIFICATION:

I hereby certify that the design and development of the vehicle **Daksh II**, 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.

Dr. Mangal Kothari
Department of Aerospace Engineering
Indian Institute of Technology Kanpur
Email: mangal@iitk.ac.in

Table of Contents

1 Team Organization and Design Process

1.1 Introduction

The IGVC team from Indian Institute of Technology, Kanpur will be participating in IGVC for their second time in 2019. Many of the design changes this year have been driven by the challenges faced by the team during IGVC 2018, where the team stood 5th in the Design Competition.

The vehicle, codenamed **Daksh II,** is a modified version of our previous year's vehicle, with minor improvements to the main chassis and major changes to the sensor mounts (and consequently, their configuration). The core electronics (motor control and power system) have been largely unchanged, though some additional systems (status indicators, visualization screen) have been added to facilitate easier testing. The software stacks for mapping, motion planning, and control received updates mainly targeted at improving their efficiency. However, most of the software for the computer vision module was re-written from scratch.

1.2 Team Organization

As usual, the team consists of both old and new members – with the new members being inducted into the team through a screening test followed by an interview. Lectures were conducted for the new members, focusing on key theoretical aspects such as sensors, computer vision, and SLAM. Since directly working on improving the already developed software would be difficult for them, they were initially assigned some short projects outside the scope of IGVC (such as voice-based control and navigation using internet services such as Google Maps).

By December 2018, a list of necessary changes was created, and work on improving the current vehicle commenced. Since a lot of the work involved improvements on the software already developed last year, most of the members of the team worked on multiple modules. These details have been summarized in the following table:

Name (Dept., Batch)	Role in Team	Name (Dept., Batch)	Role in Team
Harsh Sinha	SLAM, Cyber Challenge	Aryan Choudhary	Vision
Aalap Shah	Vision, SLAM, Motion Planning	Apurva Nandan	Electronics
Hemanth Bollamreddi	Vision, Electronics	Faizan Siddiqui	Miscellaneous
Vaibhav Agarwal	(Leader) Logistics, Mechanical	Abhishek Yadav	Miscellaneous
Kishan Shukla	Cyber Challenge	Vardhan Gupta	Miscellaneous
Nitik Jain	SLAM, Vision	Tushar Singhal	Miscellaneous
Shivam Kumar	IOP, Vision		

1.3 Design Process

Like the previous year, ROS[1] (Robot Operating System) is still being used as a means of enabling modularity and asynchronous inter-process communication. All the code used and developed by the team is still open-source, and available through our team's GitHub repositories[2]. The major design ideologies followed this year, and the motivations behind each one have been listed below:

- **Efficient Software Implementation** – One of the key issues with our vehicle in 2018 was that it had a long reaction time, which essentially restricted its speed. This was the main driving factor for most of the software changes made this year. The performance of each ROS node was profiled, and all the bottlenecks were located. For instance, transmitting the entire map

[1] ROS is a popular open-source tool that provides a C++ library and several packages for robotics development.
[2] All of these repositories are available at https://github.com/igvc-iitk, our team's GitHub organization.

from the mapping node to the path planning node resulted in a pretty long delay in generating paths. This problem was resolved this year (as discussed later in the section on mapping). As another major example, the entire computer vision pipeline was written from scratch this year, keeping the lane detection latency in mind.

- **Easier System Debugging** – Two new electronic systems have been added to the vehicle specifically to facilitate easier system debugging while testing. The first is a set of status indicators, and the second is a visualization screen mounted at the rear of the vehicle (both discussed in the upcoming sections). Within the software too, it was ensured that each ROS node outputs some information regarding its current state so that it is easier to find which process is behaving unexpectedly.

2 Innovations

Most of the innovations introduced in the previous year's vehicle have been carried forward to this year. These include our custom-made adjustable camera mount, seamless manual to autonomous switching, LiDAR-based safety braking, the mission control dashboard, etc. A few new innovations have also been introduced this year. These have been listed below:

2.1 Status Indicators

As shown in the adjacent figure, an LED strip has been attached to the E-stop pole. Different sections of the LED strip represent the status of various software modules. Since these can be visible from a considerable distance, it helps in pinpointing problems in navigation to a particular module, in real time. This makes testing considerably simpler. The legend for interpreting these indicators is described below:

Section	Legend
Mode	This is the (very bright) red safety light as required by the AutoNav competition. It shows a solid red color in manual mode and blinks in autonomous mode.
Lane	It shows white and green LEDs proportional to the percentage of lanes and grass (respectively) in the image, as detected by the computer vision algorithm.
Planning	It is yellow when the path planner is re-planning and orange when it is idle.
Control	It changes color gradually as the position controller's error increases from 0 (blue) to 200mm (red).
Braking	It becomes red whenever the vehicle suddenly tries to stop because of an immediate obstacle appearing along the currently planned path.

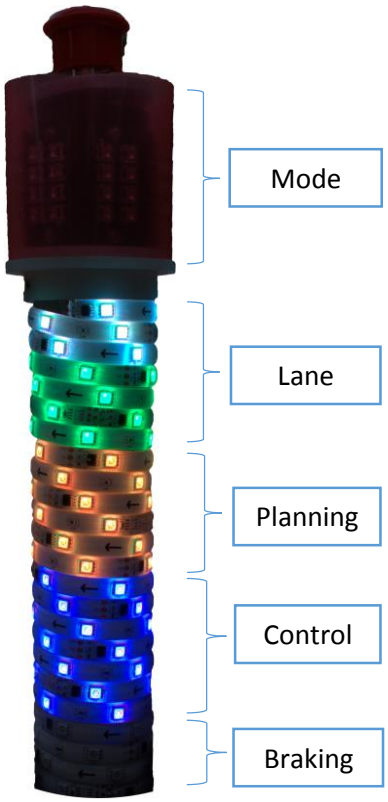

Figure 1 - Status Indicators

Additionally, the lights for a section turn off if the corresponding ROS node has been inactive for more than 1 second. The brightness of the LEDs can be controlled via the RC transmitter used for the wireless E-stop. Also, the status indicator (all sections combined) changes to a moving rainbow-colored pattern when the vehicle is being driven in manual mode (using a remote control).

2.2 Vertical Laser Scanner

Besides having a high reaction time in IGVC 2018, our vehicle also faced the problem of improper mapping because of small variations in the vehicle's orientation (because of a non-flat ground profile). The cause of this problem is illustrated in the figure below:

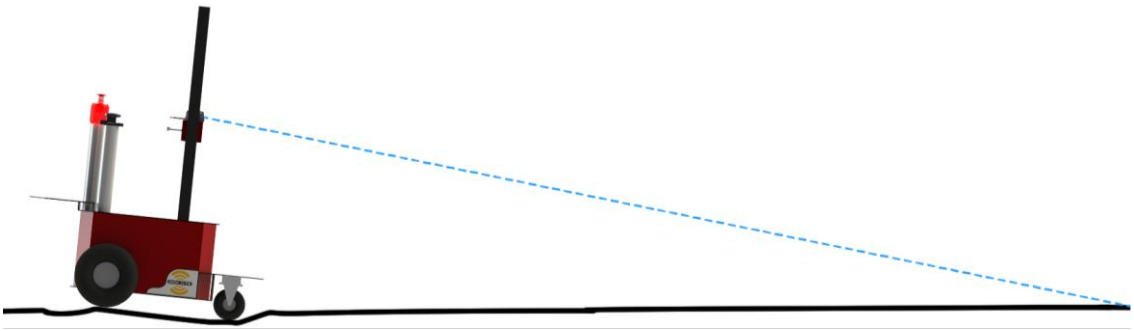

Figure 2 - Mapping problems caused by the non-flat ground profile. Note that the horizontal laser scanner will cause the points on the ground to appear as obstacles, which will lead to an incorrect map and incorrect motion planning.

This problem originates from the fact that our laser scanners can only scan along a 2D plane. Our solution to the problem is to use two laser scanners instead of one – one horizontal and one vertical. The vertical laser scanner obtains a set of points essentially approximating the ground profile ahead of the vehicle, which is then processed to determine which points scanned by the horizontal laser scanner are true obstacles. The algorithm for this is described below, along with two illustrations.

Algorithm for filtering horizontal laser scan

```
for each point in vertical_scan and horizontal_scan
    convert point from polar to cartesian coordinates
for each point in vertical_scan (cartesian)
    calculate slope m1 from previous_point and m2 to next_point
    if (atan(m2)-atan(m1) < t1) & (abs(point.height-horizontal_scanner_height) < t2)
        add point.x to false_positives
for each point in horizontal_scan
    if point.x is within delta of any x in false_positives
        remove point from horizontal_scan
```

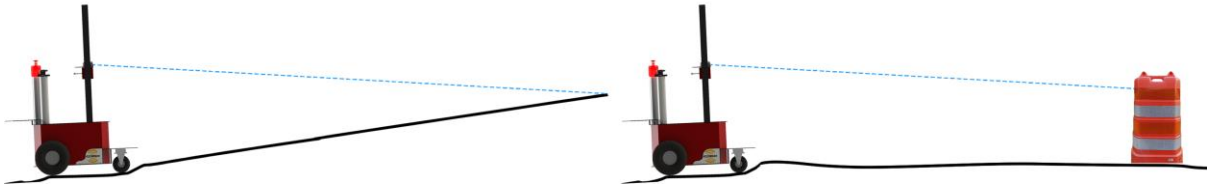

Figure 3 - Left: False obstacle avoided by the algorithm, Right: True obstacle unchanged by the algorithm

In essence, the vertical laser scanner is used to identify fault-generating locations, i.e., those above the horizontal scanner height (except where the slope difference is large, meaning it is actually an obstacle). These are then used to filter the points from the horizontal laser scan. This configuration partially makes up for not having a 3D laser scanner, without the cost overhead. The assumption here is of course, that the ground profile in front of the vehicle does not vary very much in the direction perpendicular to the profile measurement direction. This assumption is valid unless the terrain is particularly hilly. Since the vehicle is holonomic, it only has a non-zero velocity in the forward direction, which is where the map will be most accurate. Note that the vertical laser scanner can also be used for avoiding cliffs, holes, etc., which cannot be detected by a horizontal laser scanner irrespective of the mounting height. Ramps can be detected too.

3 Description of Mechanical Design

Daksh II is a 3-wheel differential drive vehicle with a front mounted castor wheel. The major mechanical changes in the vehicle this year include a separate frame for mounting all the sensors, a larger castor wheel, better weatherproofing, a proper payload holder, and general improvements for present-ability. The weight of the vehicle was also reduced to 44 kg (20% lower than last year).

3.1 Chassis

The central compartment of the chassis is a rigid iron frame covered with steel and aluminum plates. It is divided internally into two levels, with the motors, the motor drivers, the power system and controller at the lower level and the computer and batteries at the higher level. The top level can be accessed via a lid, while the lower level has been securely tightened with bolts. The top level also has a few vents for the ventilation of the computer. The payload is held using Velcro straps attached to the chassis. The ground clearance is around 12 cm (5”).

Figure 4 - The complete chassis of Daksh II (CAD Model towards the left, actual vehicle on the right)

3.2 Sensor Mounting

A sensor frame is attached to the chassis to provide more flexibility in changing the heights and orientations of the sensors without modifying the central compartment. The horizontal & vertical laser scanners and the stereo camera (with the same adjustable mount as used in the previous year) are mounted on the sensor frame at heights of 0.70m (2’ 4”) and 1.32m (4’ 4”) respectively above the ground. The heights have been increased from earlier to increase the field-of-view of the camera and to ensure that the laser scanners get a decent range even when the vehicle is slightly tilted. The inertial measurement unit and the GPS receiver are mounted on the top of a pole attached to the chassis.

3.3 Suspension System

Until the last year, a spring-based suspension was used for the plastic caster wheel, and pneumatic tires were used for the rear wheels. This year, the hard castor wheel was replaced with a slightly larger, pneumatic caster wheel. The tubes within all 3 tires are maintained at a pressure of 2 bar (30 psi). Thus, the spring-based suspension was removed.

3.4 Weatherproofing

The central compartment of the chassis encloses all of the electrical compartments and the main computer within it (which was earlier outside, hence vulnerable). Thus, only the laser scanners, IMU, and GPS receiver need to be covered in case of rainy weather. For this purpose, a flexible sheet is attached from the top of the sensor frame to the rear plate using Velcro straps when required. The stereo camera at the top is waterproof by itself. The entire chassis has been re-painted to provide further corrosion resistance. We have also tested the vehicle for extended periods in bright sunlight.

4 Description of Electrical System

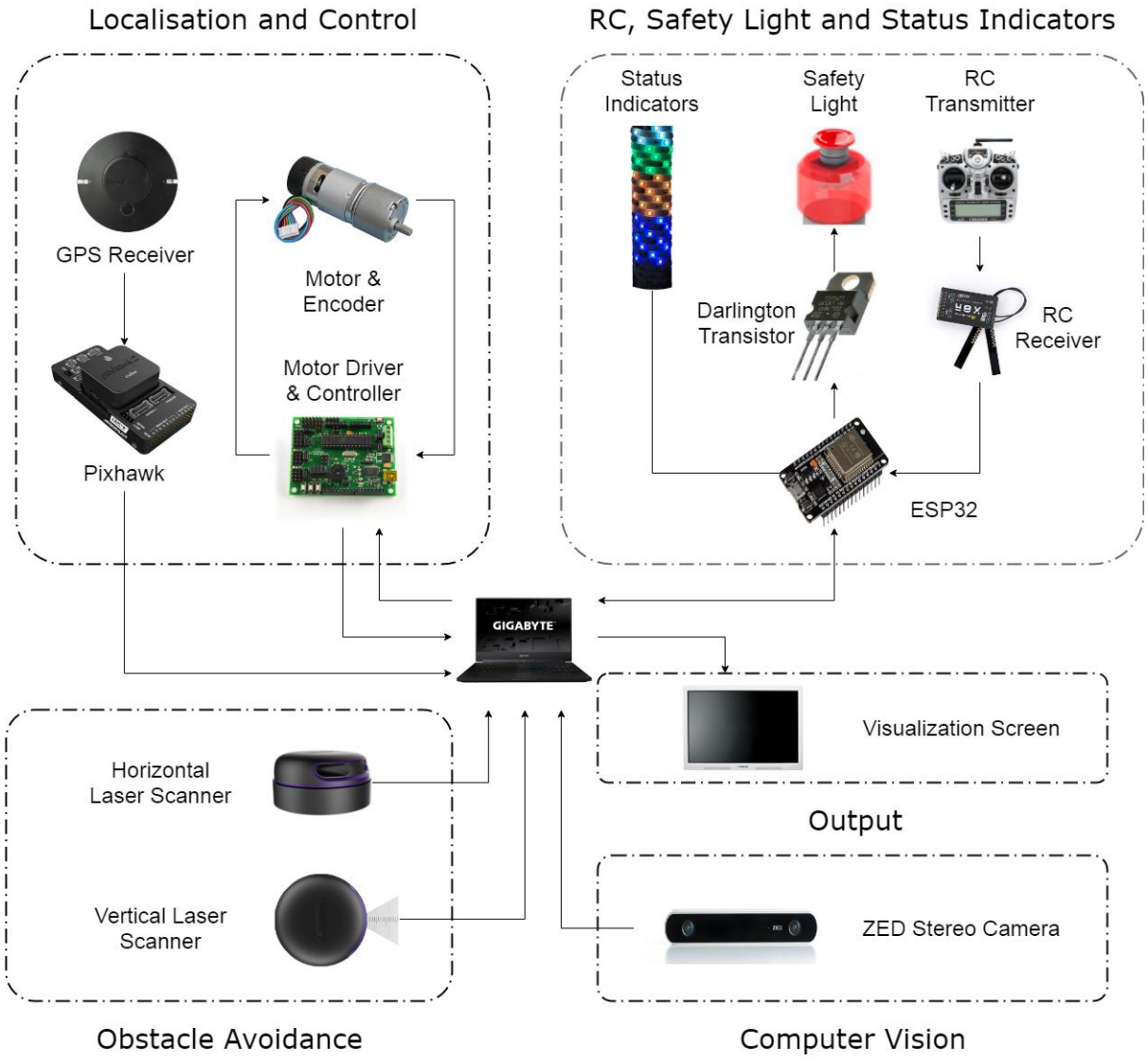

Figure 5 - Overview of Electrical System

4.1 Sensors

We have upgraded our laser scanners to a newer model with greater range. However, as explained in an earlier section, we have used a horizontal and a vertical laser scanner – both of which are 2D. This has been done in the interest of lowering the cost (compared to using a 3D laser scanner). Detailed sensor specifications are as follows:

- **Laser Scanners** – The horizontal and vertical laser scanners are both RPLIDAR[3] A3 2D laser scanners. The angular resolution is 0.25° when operating at 16000 samples/sec. The maximum range of each scanner is 25 m (82 ft), but it may reduce to around 8 m (26 ft) in cases of extreme sunlight (the interference makes detecting the reflected laser pulse more difficult). In such a case, we operate it at 4000 samples/sec. This makes the individual pulses stronger (easier to detect) and restores the full range again but degrades the angular resolution to 1°.
- **Stereo Camera** – A ZED[4] 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. While depth information can be obtained by stereo matching, we are not using it for the competition.
- **Encoders** – The integrated encoders coupled to the motors have an accuracy of 2048 counts per revolution. They are quadrature encoders with complementary-output noise cancellation.
- **IMU and GPS Receiver** – A Pixhawk[5] 2 Cube is used which has two IMUs. These are internally fused by the Pixhawk itself using an EKF algorithm. A 'Here GNSS'[6] is connected to the Pixhawk for obtaining globally-referenced position data required for waypoint navigation.
- **RC Receiver** – An FrSKY X8R 8-channel receiver is used for binding to the remote that consists of the wireless e-stop, the script execution switch and the joysticks for manual mode driving.

4.2 Computers and Microcontrollers

Along with the main computer, some minor processing is done by the Pixhawk and ESP32 microcontroller too. The exact processes run by each and the device specifications are listed below.

Device Model	Processor	RAM	GPU	Purpose
Gigabyte Aero 15X	i7-8750H (4.0 GHz)	16 GB	NVIDIA GTX 1070 (8 GB)	Computer Vision, SLAM, Path Planning and Control
Pixhawk 2 Cube	STM32 (168 MHz)	256 KB	-	Fusion and filtering of internal IMUs
ESP32	Xtensa LX6 (240 MHz)	520 KB	-	RC interface, E-Stop, LED Control

4.3 Power Distribution System

The main power system is contained within a box powered by a single 10,000 mAh, 6-cell, 22.2 V LiPo battery. It gives out various voltage levels (24 V, 12 V, 5 V), which are used to power the motors, encoders, status indicators, the safety light, the RC receiver, and the visualization screen. The 2 RPLIDAR A3 2D laser scanners, the ZED stereo camera, the Pixhawk (with the GPS receiver) and the ESP32 are powered using the USB power from the Laptop, which is powered by a 94 Wh Li-ion battery.

[3] RPLIDAR A3 is a low-cost 2-dimensional laser scanner model manufactured by SlamTec Co.
[4] The ZED camera is a stereo camera manufactured by Stereolabs Inc., intended specifically for SLAM usage.
[5] Pixhawk is a well-known, low-cost flight controller hardware project.
[6] 'Here GNSS' is a low-cost GPS sensor compatible with the Pixhawk, manufactured by ProfiCNC.

4.4 Safety Devices

The safety light is powered by the main battery and controlled via the ESP32 microcontroller through a Darlington transistor. It is brighter than all the other LEDs on the vehicle and is visible from a long distance even in bright sunlight.

The mechanical e-stop is directly attached in series with one of the wires from the main battery (through the power system box). The wireless e-stop works through a tri-state on the RC transmitter. When the switch is at the top position, the ESP32 microcontroller (connected to the RC receiver) sends a 5 V kill signal to the main drive system that isolates the battery from the drive system via a Darlington transistor.

Further, separate, removable fuses are attached for every system that is connected to the main battery. Also, when the vehicle is pushed manually in the off state, there is a back-EMF generated because of the motors. If the current is high enough, it can damage the motor drivers. For this reason, a small buzzer is attached to the wires carrying the current from the motors in order to warn the person pushing the vehicle.

4.5 Debugging Devices

A visualization screen is attached to the rear of the vehicle. It uses RViz[7] to display the current state of the map, the vehicle's current location within it, and the currently planned path. The input video stream and the detected lanes can also be seen. The visualization screen is connected to the main computer via an HDMI cable.

The status indicators also aid in debugging (especially when one is too far from the screen to check on the current state of the vehicle). Their functions have been described in an earlier section. They are controlled via the ESP32 microcontroller.

Figure 6 - The visualization screen and status indicators attached to the rear of the vehicle

[7] RViz is a 3D environment visualization tool compatible with ROS.

5 Description of Software Strategy

The software has been divided into 3 modules – Computer Vision, SLAM, and Motion Planning & Control. The biggest change this year was the introduction of a much more robust and faster computer vision pipeline. The other modules mostly received updates that contributed towards improving the speed of the vehicle.

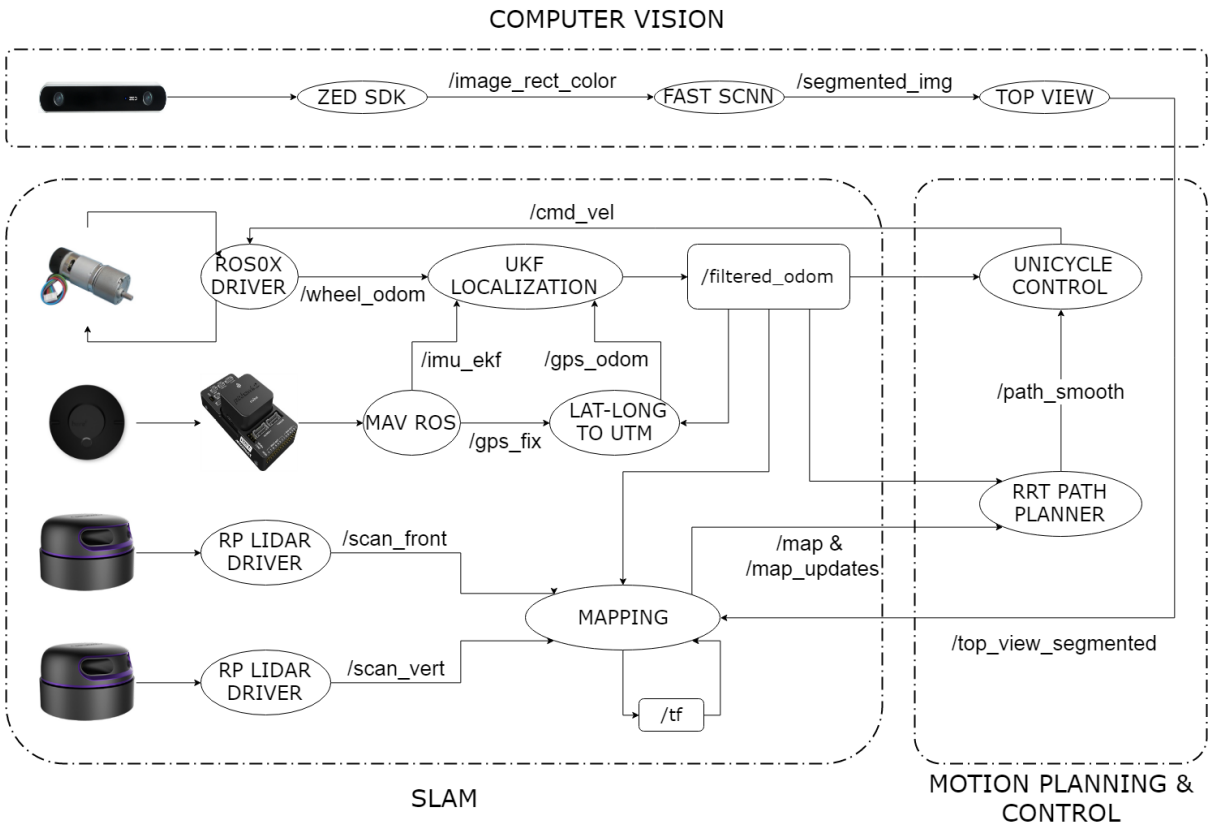

Figure 7 - Overview of Software Strategy

5.1 Computer Vision

Until last year, our lane detection architecture consisted of depth-based obstacle detection[8], followed by a top-view transform, followed by superpixel segmentation on the GPU, followed by a shallow convolutional network for predicting whether there is a lane at the center of each 7x7 superpixel grid, followed by a remapping from each superpixel to its pixels. This turned out to be pretty slow in practice, mainly because of the large number of ROS nodes carrying data back and forth between the CPU and GPU, and also because of inefficient implementation. Thus, we decided to use a fully convolutional semantic segmentation network for lane detection this year:

Dataset Creation – The first step was to create a labeled dataset for training the network. Three IGVC-like test tracks were created across our institute's campus. Then, multiple video sequences were recorded from each track at different times of the day. Rather than the size of the dataset, the variety was emphasized – so that the network would learn a general representation of painted lanes on grass (instead of over-fitting to particular times of the day or a particular type of grass). Then, a few frames were extracted from each video sequence in a way that they don't overlap much (to avoid redundant

[8] Note that it is not necessary to perform obstacle detection using computer vision (since it is done better using laser scanners). However, removing obstacles (esp. white striped) from images improves lane detection itself.

data). Finally, pixel-wise labels with three classes – lane markers, grass, and unknown (obstacles, people, etc.) – were created for each image. Adobe Photoshop was used to simplify this process (a set of automatic actions were created for each step of labeling) so that it would take about 2 minutes for an inexperienced person to label an image. A total of 552 images were labelled this way by the team.

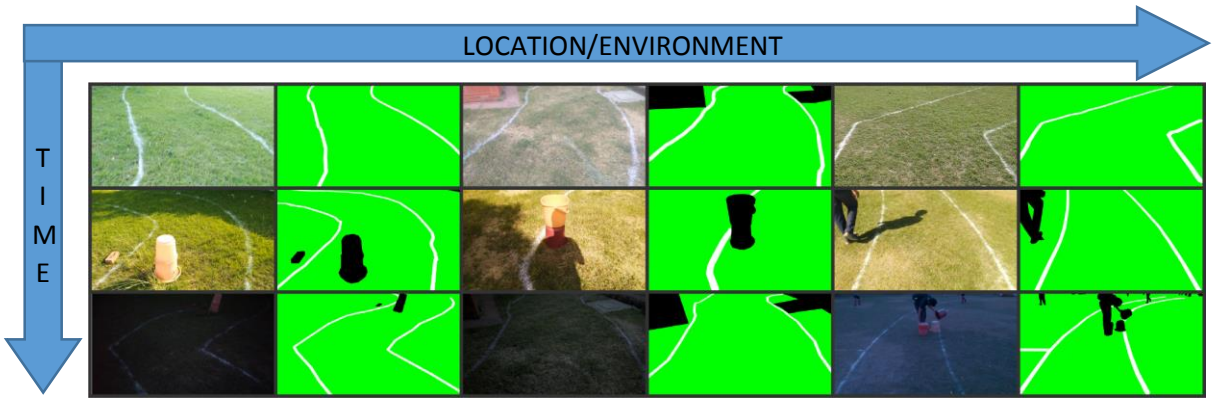

Figure 8 - Sample images from our labeled dataset (input on left, labels on right). White pixels represent lane, green pixels represent grass, and black pixels represent unknown. Note that the dataset consists of different lighting conditions (cloudy, bright sunlight with shadows, low-light) and many types of grass (dense green, patchy, yellowish).

Network Architecture, Training and Results – Since we wanted to optimize for lane detection latency, we needed a semantic segmentation network with very low inference time. Therefore, we made an implementation of Fast-SCNN[9] using PyTorch[10]. It utilizes ideas such as depth-wise separable convolutions (which are an approximate but much faster replacement for convolutional layers) from MobileNets[11] and inverted residual blocks from MobileNetsV2[12]. Further, it uses the pyramid pooling layers introduced by PSPNet[13], which essentially pool high-level features at different scales from the entire image, adding global context to pixel-wise predictions. The network architecture is as shown:

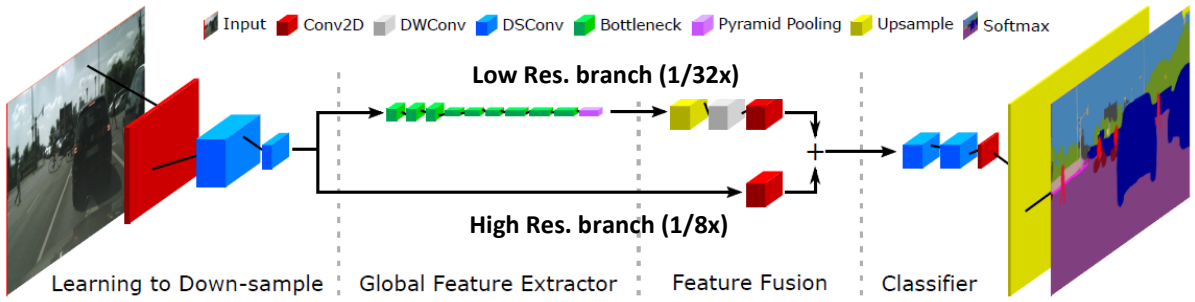

Figure 9 - The architecture of Fast-SCNN. Source: Fast-SCNN paper referenced above.

For training, the dataset was split into training, validation, and test parts. To prevent over-fitting, the input data was heavily augmented (including random crops, flips, rotation, and even slight color jitter) and dropout layers were applied. The optimization algorithm used was SGD (stochastic gradient descent) on cross-entropy loss. After training, the IOUs (intersection over union) obtained on the test set were 40% for lane markers, 90% for grass, and 30% for unknown. Note that although the values do not seem impressive, the reason for this is the lack of a well-defined (high-contrast) boundary between the lanes and the grass, meaning that the label boundaries themselves might not be perfect.

[9] See '**Fast-SCNN: Fast Semantic Segmentation Network**' by Rudra P K Poudel, Stephan Liwicki, Roberto Cipolla.
[10] PyTorch is an open-source python framework for prototyping deep neural networks.
[11] See 'MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications' by A. Howard et al.
[12] See 'MobileNetV2: Inverted Residuals and Linear Bottlenecks' by Mark Sandler et al.
[13] See 'Pyramid Scene Parsing Network' by Hengshuang Zhao, Jianping Shi, Xiaojuan Qi, Xiaogang Wang, Jiaya Jia

However, we can analyze the performance of the network qualitatively, as shown below:

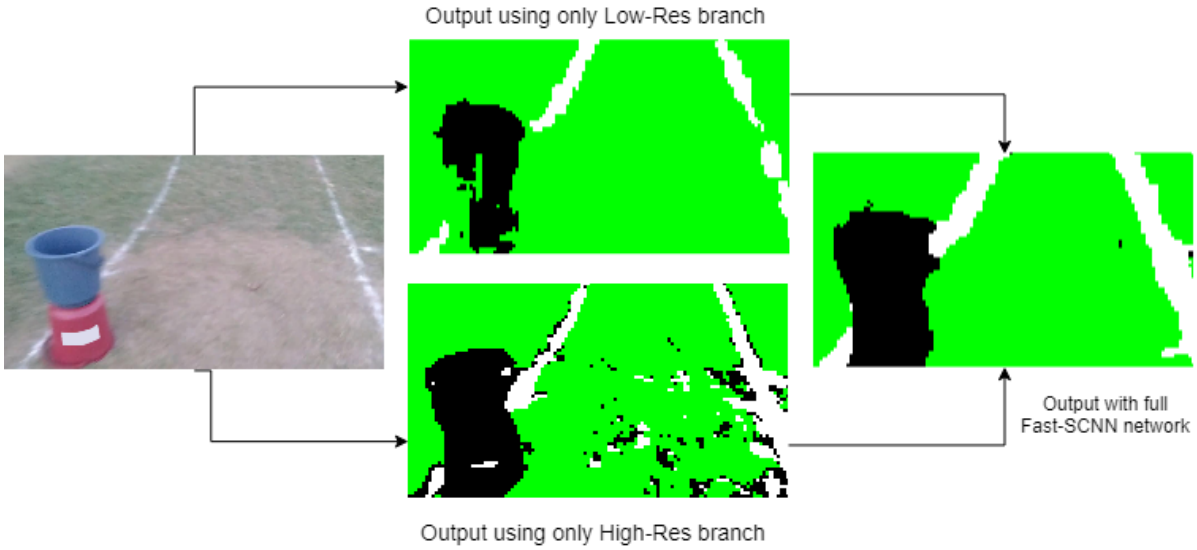

Figure 10 - Qualitative performance of FAST-SCNN on IGVC-like data

As expected, the high-res branch detects mostly groups of white colored pixels as lanes without getting any global features (note the white sticker on the bucket). The low-res branch classifies objects better in general but has inaccurate boundaries (note the lower bucket). When both the branch weights are activated, the output has neither of these problems. This explains the motivation behind this architecture.

Real-Time Inference – While we were obtaining output frame rates of around 20 FPS with the PyTorch implementation, we wanted to improve the speed even further to reduce power consumption when we run it at our desired FPS. That would also ensure that it can run with all other ROS nodes running in parallel. Thus, NVIDIA TensorRT[14] was used to increase the inference speed. It makes use of optimizations such as pre-allocating memory for layers, performing calculations at lower precision, etc. for optimizing performance. Eventually, with some more optimizations in our code, we were able to achieve a full 60 FPS, which is the maximum frame rate of the ZED stereo camera.

Once the segmented image has been obtained, the top view node (not modified from last year) is used to transform the image for mapping.

5.2 Simultaneous Localization and Mapping

The Pixhawk + GPS Receiver and the laser scanners both come with open-source ROS drivers. Thus, we directly receive data in the form of ROS messages from these drivers.

UKF (Unscented Kalman Filter) Localization - An open-source implementation of the UKF algorithm for ROS, named robot_localization has been used to fuse the wheel odometry, GPS and inertial data (orientation and angular velocity only, not acceleration). The package internally maintains a state vector of length 15. It has been set up so that it performs a prediction update at 20 Hz, assuming a 3D non-holonomic unicycle model, using the control input and the current state. A measurement update is performed each time a new measurement arrives. This part of our software has remained unchanged since the last year.

[14] NVIDIA TensorRT is a platform for high-performance deep learning inference.

Mapping – As mentioned earlier, the inclusion of the vertical laser scanner for filtering bad points detected by the horizontal laser scanner is one of the key changes to this year's mapping strategy. Further, as mentioned earlier, transmitting the entire map between ROS nodes resulted in a pretty long delay in generating paths. This problem was fixed with a very simple modification. Instead of passing the entire map to the motion planning node each time, the mapping node only sends data within a small neighborhood of the current location. Since the perception sensors all have a limited range, the map outside this region is unchanged. Thus, instead of sending the entire map to the motion planning node, only the updated part is sent. The motion planning node maintains its own copy of the map and keeps on updating it using the updated parts sent to it. This clears off a huge bottleneck as copying a large map from one node to another takes a lot of time.

Also, the ICP (Iterative Closest Point) algorithm has been implemented for mapping too. Earlier, the map was generated by simply transforming the laser scan and lanes to the map frame.

5.3 Motion Planning and Control

The motion planning and control stacks have one major change this year – the paths generated by the path planning node and those followed by the high-level controller are cubic splines instead of a combination of straight lines and circular arcs like the previous year. When arcs and lines are used, the second order derivate of the path is discontinuous (angular velocity changes instantaneously). Cubic splines ensure that the path is smoother. This makes it easier for the controller to follow the path, reducing the control error (at the same speed). This allows us to increase our vehicle's speed.

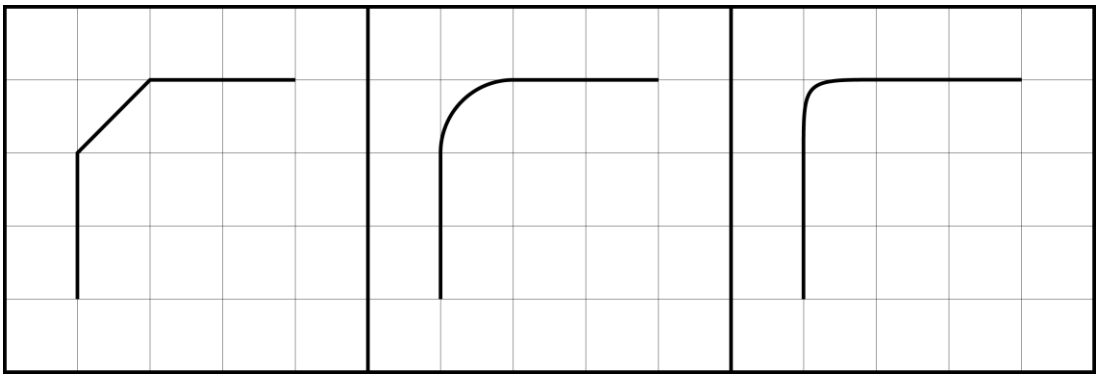

Figure 11 - A comparison between continuous paths (lines, left), 1st order differentiable paths (lines and arcs, middle) and 2nd order differentiable paths (cubic splines, right).

Path Planning – The RRT algorithm was chosen because it is fast, and it is possible to apply non-holonomic constraints to it. The algorithm used in our implementation can briefly be summarized as:

Algorithm for RRT

```
1   rrt = a tree with the current location as the root node
2   while rrt.num_nodes < max_nodes and num_iterations < max_iterations
3      sample = random sample from the configuration space
4      parent = nearest neighbor of sample from all the nodes in rrt
5      if abs(turning angle to go from parent to sample) < max_angle
6         if the straight line joining parent and sample is collision-free
7            if its length is ≥ delta_step
8               truncate the line length to delta_step
9               update sample accordingly
10           add sample to rrt
11  find the nearest neighbor of goal in rrt and trace a path to the root
```

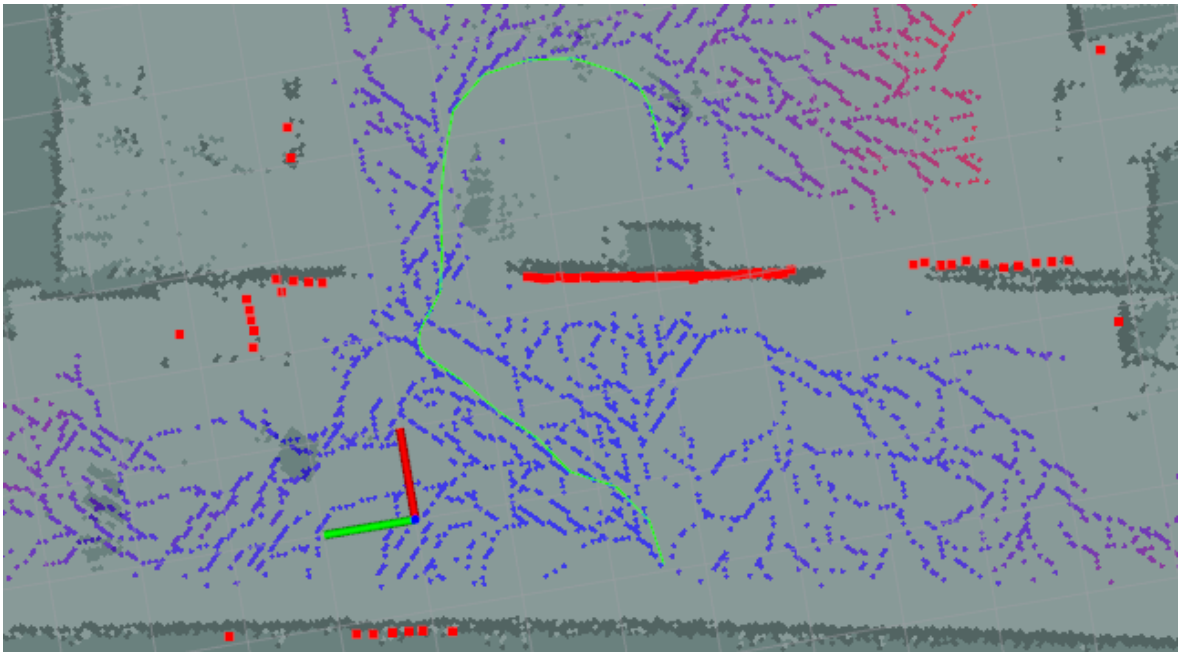

Figure 12 - A sample path (green) generated by our RRT implementation. The blue nodes on the map show the full tree.

Control - The control law used is the same as the previous year. It has been mentioned again for reference:

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

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

Here, v_d and ω_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

We implement IOP using the open-source FKIE IOP-Core[15] package. The code for various services like discovery, liveness, health monitor, etc. are pre-implemented using Jaus Tool Set (JTS). JTS is an open-source implementation of JAUS published by SAE. It also provides a GUI for designing JAUS software in terms of a Finite State Machine (FSM).

The repository provides an IOP-ROS Bridge for communicating the ROS messages to JAUS and vice versa. All services of the ROS-IOP Bridge are implemented as a plugin.

In JAUS/IOP a robot represents a subsystem with all payloads and computers. For the whole subsystem, only one component, namely 'platform_manager,' will manage the discovering of all IOP/JAUS services. Various services and the parameters can be set using the launch files.

[15] Available at https://github.com/fkie/iop_core/

6 Failure Modes and Resolutions

Many failure modes were identified during our performance tests. The following table lists these modes and their resolutions (or at least identification):

Failure Mode	Resolution/Identification
A software module is misbehaving	Status Indicators
Ground profile not flat/cliff/hole	Vertical Laser Scanner
Wireless e-stop out of range	Velocity set to 0 automatically
Laptop overheating	Ventilation inside the central compartment
Battery drainage	Battery monitor attached to all batteries
High current through any component	Removable fuses attached
Wires getting tangled in wheels	All wires have been moved to sleeves inside the vehicle
High back-EMF	Buzzer attached to alert the person
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)

7 Miscellaneous

7.1 Simulations and Performance Testing

Our team uses ROS bag files[16] to store sensor data during testing. This data is then replayed later to test our algorithms. Since testing on real data is more reliable than testing, we have always preferred this method against simulations.

Three IGVC-like courses were built around our institute's campus. This allowed us to collect enough data for training the semantic segmentation network. These courses were later used for performance testing. Many errors were found in the full system integration, but they were fixed, and our vehicle was able to navigate the course autonomously. As of now, we are able to stay above the minimum speed limit of 1 mph and are trying to improve it. The vehicle can easily climb ramps of up to 30° while carrying the payload. The endurance of the system (computer + vehicle) is above 60 minutes (usually, the computer runs out of power first) but the exact amount varies based on a lot of factors.

7.2 Cost Estimate

A cost estimate for the vehicle, were it to be built again, has been given below. Note that manufacturing cost has also been included in the estimate.

Item	Cost (USD)	Item	Cost (USD)
Manufacturing and raw materials (chassis and sensor mounts)	2500	Camera (ZED)	450
Caster	90	IMU (Pixhawk 2 Cube)	435
Wheels	30 x 2	Here GNSS	75
Motors (NEX)	110 x 2	Gigabyte Aero 15X	1800
Laser Scanners (RPLIDAR A3)	560 x 2	Miscellaneous	50
Total	**6800**		

[16] ROS bag files are recordings of sensor data and generated messages stored in a streamable format on the disk.

7.3 Planned Future Improvements

Since a semantic segmentation network provides a general framework for all detecting all kinds of lanes, buildings, trees, etc., we plan to use the vehicle beyond IGVC, in more varied scenarios. Given that our semantic segmentation network already works in low-light environments, it is possible to navigate even at night if an additional headlight is provided.

We also plan to work with pre-loaded maps so that we can navigate faster in known areas. This is also useful for AutoNav, where some data from a previous run can be used in the next.

7.4 Acknowledgments

We would like to thank our faculty advisor, Prof. Mangal Kothari, and our Institute, for their support. We would like to acknowledge the work of all of our previous team members who have contributed immensely to the development of our vehicle. Our progress would not have been possible if not for the contributions of the authors of the research papers that we have implemented, as well as the developers of all the open-source projects that have been used in the software for our vehicle.