Final Report - Part II: Module Details

CSC Students
Department of Computer Science
North Carolina State University
Raleigh, North Carolina 27607

ECE Students
Department of Electrical and Computer Engineering
North Carolina State University
Raleigh, North Carolina 27607

Civil Students
Department of Civil Engineering
North Carolina State University
Raleigh, North Carolina 27607

I. Introduction

In this report the workings of each module are explained in detail. Each team describes their contributions beginning with their goals then mentioning their approach to solving their goals while also mentioning failures and successes in achieving those goals. Each team also discusses how they collaborated with their colleagues on other teams and comments on proposed future work to extend their module.

II. Husky Hardware

A. Goals

The final outcome of this component of the project involved a lot of individual team as well as integration effort. The Husky is the backbone for all the other teams and hence plays a pivotal role in the overall scheme of things. A brief about the goals of the husky hardware is listed as below: Redesign of hardware platform for robustness and better accessibility.

1) GPS, IMU, LIDAR and Encoder based localization
2) Mechanism and Software for blimp tracking (e.g.,pan-tilt mechanism)
3) Robust wireless communication and streaming (between blimps and Husky, and between Husky and central station)
4) Motion planning for Husky and blimps

B. Approaches

The approach taken involved the below steps

1) Understanding Husky platform: :
The husky hardware was already designed to a certain extent by spring 2017 batch of students. Hence, it was imperative that the prior knowledge of the design was understood so that the general approach could be taken forward. Hence, any documentation, codes and designs were studied carefully to see parts where modifications/improvements could be made. The team reverse engineered the system by starting from understanding the goals and then breaking it down into specific tasks that would be required to reach the goal. The tasks were then divided individually so that each team member has equal contribution. A lot of integration efforts would need to be put in and hence, the overall system design must accommodate the requirements from the other teams as well. The major issues overcome during this course, based on feedback from seniors and professors include:

1) Networking
2) Manual control using wired and wireless joystick
3) Integrating the systems and coordinating with various teams.

2) Designing/selecting additional hardware: Due to additional systems and requirements like blimp coming in to the picture, the Husky would have to be redesigned in order to accommodate newer components. Most of the mounting brackets/cases were either 3-D printed or created in the Makerspace. The additional components that were added to the system are:

1) IMU
2) GPS
3) PointGrey Wide angle camera
4) Tilt unit to control the wide angle camera.
5) Mounting brackets for each additional component.

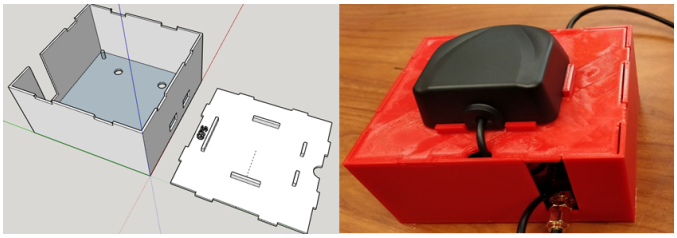

Fig. 1. 3D printed GPS case

3) Autonomous navigation using LIDAR: Since the ZED camera provided Husky with the best information to navigate autonomously, the team was dependent on SLAM to provide the map using the ZED camera's data. Since this was a dependency out of our control, we decided to navigate the Husky autonomously using the LIDAR to explore issues that were pertaining to the autonomous navigation of the Husky. The overall idea of integrating with the LIDAR is as shown below:

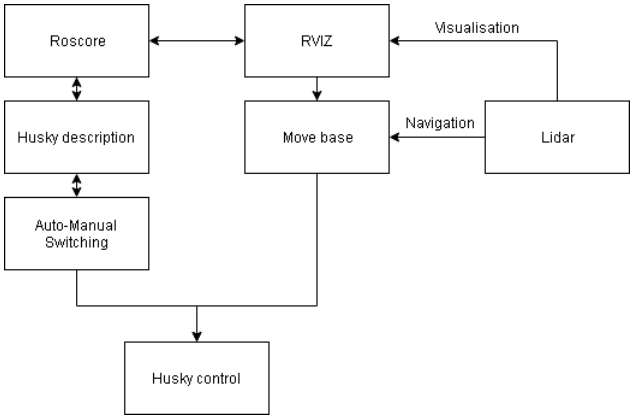

Fig. 2. Autonomous navigation using LIDAR

The LIDAR provides a 2D point cloud to RVIZ. The obstacles that can be successfully mapped are only those that are present at the height at which the LIDAR is mounted. Hence, the map generated is not accurate and will not avoid all obstacles. However, it was a good starting point to test the Husky's autonomous navigation capabilities and to understand the drawbacks of the LIDAR.

4) Autonomous navigation using ZED Camera: This approach we changed the input for generating the map. Instead of using LIDAR, we moved to using the ZED camera since we can obtain a 3 dimensional map of the obstacles. The map also updates real time and hence, provides a better understanding of the environment to the Husky.

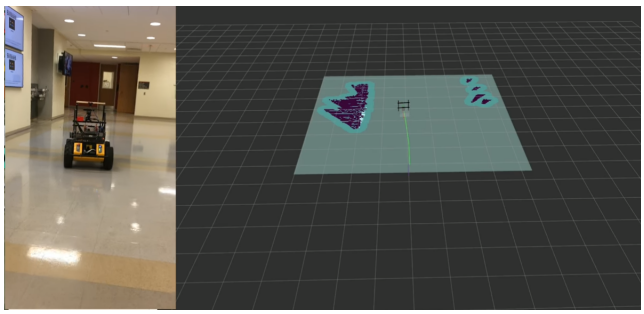

Fig. 3. RVIZ simulation using ZED Camera

5) Optimizing path followed by the Husky: The difference between the previously explained approaches were more from the sensors point of view. However, the algorithm implemented to navigate was not optimal. The path followed by the Husky for goal commands entered on RVIZ was a randomly generated path. It used movebase libraries provided by clearpath robotics. In order to optimize the path, we implemented RRT to calculate the shortest possible path from start to end.

The implementation of RRT seen in Figure 4 was adopted for our navigation.

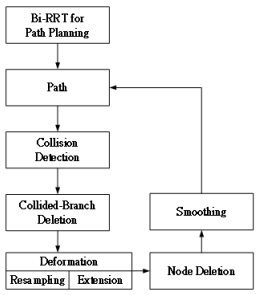

Fig. 4. RRT Algorithm implementation

6) Architecting the final solution: Although the Husky can now autonomously navigate, a lot of other key features need to be added in order to integrate with other teams as well as adding safety features such as manual mode. The ROS messages had to be subscribed to as well. Hence, we developed the below steps The steps to be followed move a Husky involve:

1) Roscore: ROSCore is started by ROS Master.
2) Husky description: Publish husky description through a URDF file which will allow us to visualize husky in RVIZ. Basically it is the dimensions of husky and thus we get a husky model in RVIZ.
3) Auto-Manual: This code was developed to switch between manual mode and auto mode. There are situations were in auto mode husky may behave abruptly and hence as a backup we keep manual mode.
4) Husky control: This code which send the velocity of the wheels to husky. This is done over TCP/IP. This code contains everything it will calculate the speed based on the occupancy grid. This code controls the movement of the husky.
5) Move base: Takes Goal commands from RVIZ and tells us where the husky is supposed to move.
6) This is visualization software present in ROS. Using this software we can see the path that husky is planning and how the map is generated and what is happening inside husky. Just the representation of the internal system visually.

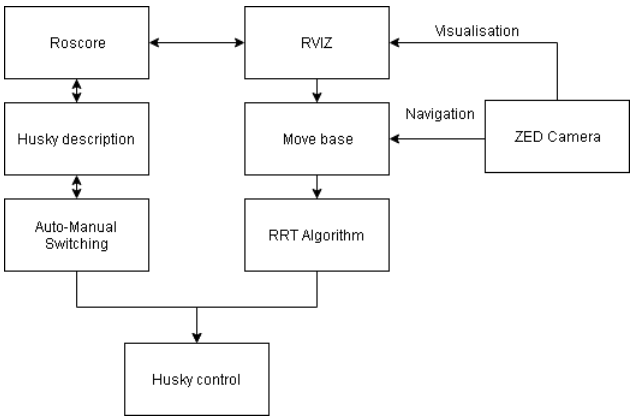

Fig. 5. Autonomous navigation flowchart

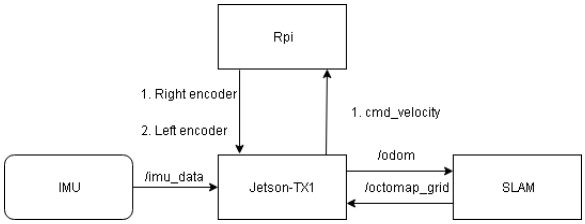

Fig. 6. Software messages

7) Blimp Tracking: The objective of the blimp was to provide aerial footage that was out of the view of the Husky. However, this might involve the blimp to fly to different locations from the construction site. The Husky had to track the blimp so that it can localize the position of the blimp relative to the husky.

The blimp tracking consists of 2 modules. There is a whycon marker attached to the blimp. The camera searches for a whycon marker in each frame as it continuously acquires data of its surroundings. Once it identifies the whycon marker, it tracks it with the help of tilt unit.

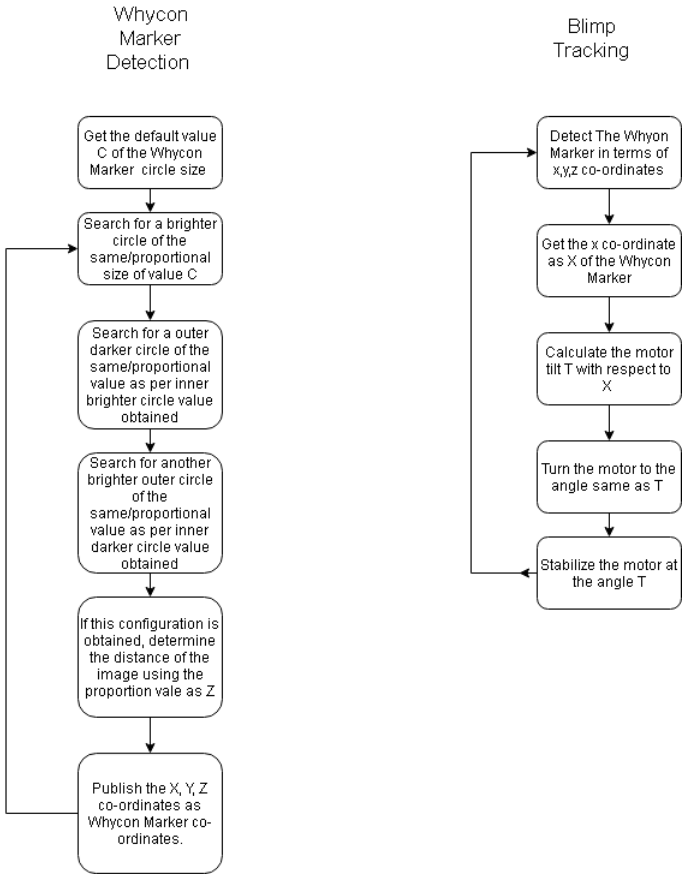

Fig. 7. Blimp Tracking

III. Context A

A. Goals

The major goals for this project are as per the pipeline described in figure **??**. One should note that the pipeline is the general overview of the system. But mainly, the goals in this research can be listed as follows:

1) Investigate semantic segmentation frameworks
2) Implement and modify a semantic segmentation framework for real-time use on the Jetson TX1
3) Integration with SLAM B team to create 2D occupancy grid map

In the Approaches Section each of goals, is explained individually.

B. Approaches

1) Investigate semantic segmentation frameworks: Pixel labeling involves creating network models with considerable number of parameters as each of the pixels need to be labeled. For the model to work on Jetson which has a memory and processor power constrain, a small architecture compromising with the quality should be used. There are many options when it comes to implementing vision based semantic segmentation ENet [**?**], AlexNet [**?**], and Segnet [**?**], are the most widely used algorithms.

In the first demo we decided to experiment with different segmentation algorithms. We had three Context awareness teams and each team decided to try a different segmentation algorithm. We chose SegNet for doing semantic segmentation. But, in the second demo we decided to change to the ENet. ENet segmentation architecture has been designed to work on embedded boards, which provides pixel wise image labeling. this model could run on Jetson TX1 at speed of 7 fps with input image size of 360×640, which is much faster than other models such as SegNet [**?**] , AlexNet [**?**], etc. The model based on Caffe was executed on the embedded board with changes done to accommodate the different class labels.

2) Real-Time navigable space segmentation: ENet takes as input labeled pixel wise information during training. 1000 images, captured from multiple videos, were sampled and labeled manually. An RGB image with pixel information being the class label and size similar to input image size is used for training the network. 3 labels were used for labeling - Obstacles, Ground, Sky (see Figure 8).

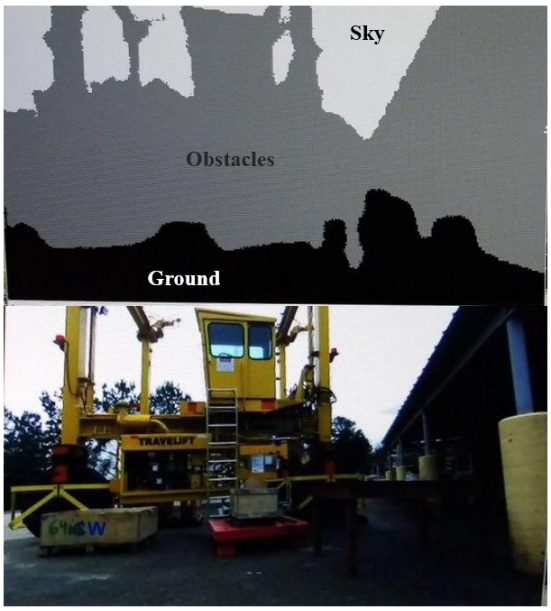

Fig. 8. Labeled classes: ground, obstacles and sky

We trained the ENet model on a server having a Xeon E5, 128GB ram, as the CPU and Tesla k40c and Tesla k20c with 12GB VRAM as GPUs. The Docker container was used to train the ENet model on the server. Docker is a computer program that performs operating-system-level virtualization also known as containerization. We used Docker virtualization tool to overcome different version interdependencies while training on the server. We used the trained model weights on Jetson to carry out the inference process.

The network was trained in 2 steps. First step involved training the encoder part which takes input of size 360 $\times$ 640 and provides label map of size 360 $\times$ 640. The model is trained for 1000 epochs with a batch size of 4. In the second step, the decoder was trained on top of the encoder, to upscale the smaller size intermediate map to full image dimensions.

In the final demo we represented entire images by only two pixel values i.e. black and white. Ground was represented by white and rest of the image is represented by black. This was done because, this segmented image was used for plotting occupancy grid. which is needs image in this format.

Figure 9 shows three rectified color images, segmentation ground truth and their corresponding output segmentation map produced by our trained ENet model.

The computational load put on the Jetson board is presented as a benchmark for future systems and pipelines. Remote logging is performed to record the hardware usage. The GPU usage is measured using the tegrastats utility provided by NVIDIA.

The execution time for the Context-Awareness Module is largely governed by the size of input images. This module runs heavily on the GPU as can be seen in Figure 10. The Context-Awareness Module is triggers by every image frame that takes up some memory. When an image is passed over to the ENet network, a segmented image is produced and is shown as a spike in the GPU usage in Figure 10. This process publishes the boundary-position vector at the end of processing. At the same time, the image is released from the memory after being processed which explains the regular rise and falls in the free memory. Other results relating to CPU and memory usage are shown in Figures 11 and 12.

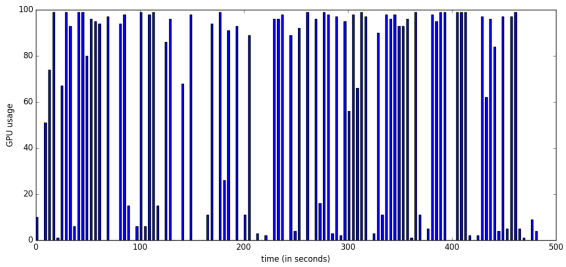

Fig. 10. GPU Usage

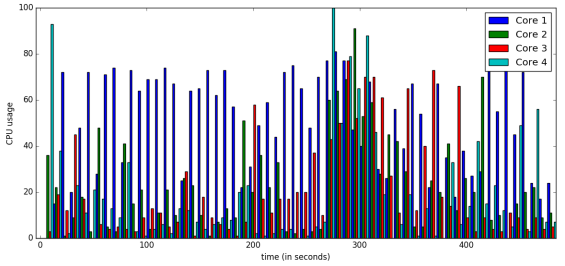

Fig. 11. CPU usage

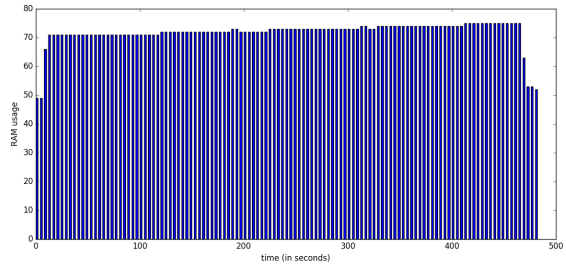

Fig. 12. RAM usage

3) Obstacle vector extraction: Pixels labeled as road are the parts of the image where the robot can traverse safely. A one dimensional vector, giving the first instance of the obstacle when traversed from the bottom is computed. Figure 13 shows the 2D obstacle vector of three sample images. The depth information for all the pixels on the obstacle vector are available, this means that we have the obstacle location in real-word which can be used for the 2D occupancy map creation.

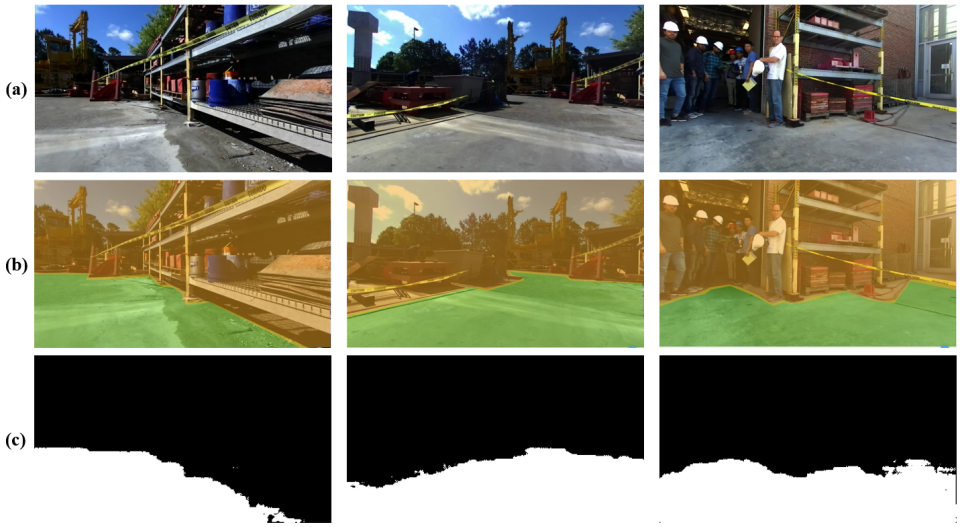

Fig. 9. (a) rectified images, (b)ground truth segmentation, (c) output segmented images

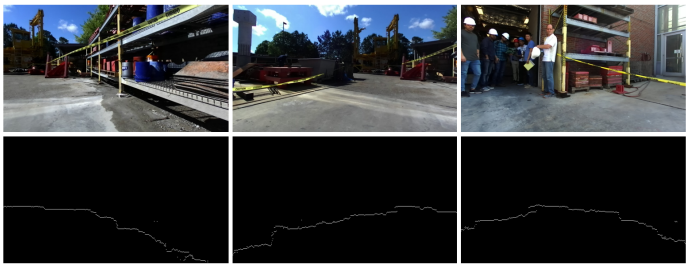

Fig. 13. top: original images, bottom: extracted obstacle vectors

4) Integration with SLAM B team to create 2D occupancy grid map: As mentioned in the introduction, the main goal of this project is to use a stereo camera mounted at a specific height on the Husky and generate the navigable map of the environment using segmented images and depth information. In this project Context Awareness Module provides the segmented images with corresponding depth information to the SLAM B team which need these information in order to create 2D navigable map.

For the Third Demo we decided to concentrate on integration with SLAM-B team. We used ROS kinetic for this purpose. We subscribed to ROS image topics published from the ZED camera by the SLAM-B team. We subscribed to the left camera image topic and carried out the segmentation on that stream of images. We publish our segmented images and the corresponding depth images as an image stream using ROS so that the occupancy grid can be obtained by SLAM-B team.

IV. Context B

A. Goals

The major objective of the integration is the generation of an occupancy grid of the environment which in turn provides a spatial mapping of the obstacles and the navigable area in the environment for the autonomous robotic system.

For the autonomous system to be able to navigate an unknown space, it needs to be aware of its surroundings. This encompasses the ability to simultaneously map the environment and localize the robots location. The two autonomous systems used for the project are a ground based unit - Husky and a floating unit - blimp. The blimps role is to act as a secondary unit that traverses through areas that are otherwise inaccessible to the husky and provide it with the corresponding grid maps.

The purpose of the context awareness module is to act as an auxiliary source of grid maps is done through a series of steps. The images streamed from the blimp are first subscribed to from a master ROS node. These images are then segmented into two classes(ground and obstacles) by the pre-trained network. The segmented images indicates the regions that are traversable and non traversable. Once the incoming image is segmented, the resulting mask is then mapped onto an grid along with the 3D point cloud data published by the SLAM-A team for map visualization.

The major goals or objectives that we identified for our context awareness module to work in collaboration with other teams are are follows:

- Understanding what a construction site looks like?
- Collect data and generate necessary annotations
- Understanding the capacity of the hardware we have?
- Figure the real time segmentation algorithms?
- What frameworks exist for doing segmentation in real-time?
- Train on a suitable hardware for building a model
- Deploy the inference model on Jetson TX1
- Read real-time camera feed from blimp
- Generate the segmentation result
- Merge the point cloud information from SLAM with the segmentation result
- Generate the occupancy map from the above result

The above steps are discussed in further detail in the below

Fig. 14. Sample image collected for training (on the left) and its corresponding annotation (on the right)

subsections.

B. Approaches Taken

Understanding the construction site: Before we start off with the entire module, it is important to understand the typical scene at a construction site. The view of a robot will be markedly different from what a normal person sees. It becomes an even more challenging task when the navigable robot is small in comparison to a human, how to decide on the objects to be classified as safe or unsafe and identifying the major features to be used for deciding the classes for segmentation. Other major challenge is also to figure out accurate depth information given the varied environment scene that one typically environments in a construction setting.

Given that we were going to work in collaboration with the indoor blimp team, we decided on primarily using only two classes for classifying the myriad things we see in a construction site. One was ground which serve as the navigable area for the Husky and the other was obstacles which would encompass any item that would prevent the ground from being navigable for the Husky.

Data collection and description: Collect data and generate necessary annotations: For training the segmentation model, the dataset was collected from the CFL site using a mobile device. The camera was manually taken along an entire path inside the CFL site for segmenting data received from Indoor Blimp. The camera was held at an angle of 30 degree from the vertical keeping in mind the orientation of camera on the blimp. Care had to be taken that every frame collected had ample free space for the husky's movement. An entire video of 11 minutes was taken with a frame rate of 30 frames per second. Manual jitters were explicitly given at regular time intervals in order to replicate the movement of blimp. Finally, 1 frame every second was extracted making it a total of 660 frames for the entire video. Once the frames were extracted, every frame was manually labeled into two subparts as described earlier using an online annotation tool - Supervisely.

Choice of hardware and its capability: Once the data collection and annotation were complete, the next task of paramount importance was understanding the capabilities of the hardware and the rich set of literature and different frameworks available that could help us accomplish the image segmentation. For processing capabilities, we relied on the use of NVIDIA Jetson TX1 boards. It's built around the revolutionary NVIDIA Maxwell architecture with 256 CUDA cores delivering over 1 TeraFLOPs of performance. 64-bit

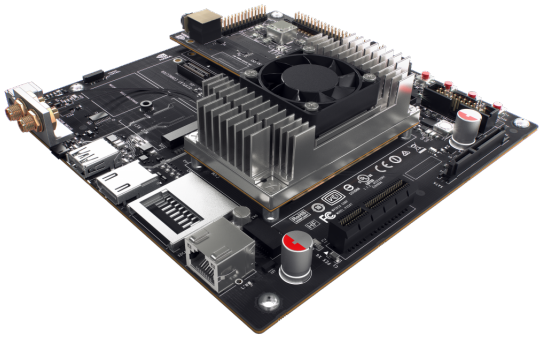

Fig. 15. A NVIDIA TX1 embedded GPU Platform

CPUs, 4K video endcode and decode capabilities, and a camera interface capable of 1400 MPix/s make this the best system for embedded deep learning, computer vision, graphics, and GPU computing. The board also exposes a variety of standard hardware interfaces, enabling a highly flexible and extensible platform. This made it an ideal choice for our applications requiring high computational performance in a low-power envelope.

Real time segmentation techniques: Now that our hardware capabilities have been defined, our next goal was to research different algorithms and frameworks available to help with the actual segmentation task. The ability of a deep learning model to perform semantic segmentation or pixel wise classification in real time is very important for a wide range of applications like autonomous driving and augmented reality with challenging datasets available nowadays. Therefore design of CNNs becomes very vital. Paszke et. al. introduced ENet as an efficient lightweight segmentation network with a bottleneck module. Badrinarayanan et. al. proposed SegNet as an early attempt for end-to-end semantic segmentation in encoder-decoder architecture. Long et. al. proposed the first attempt for fully convolutional segmentation network (FCN) that was trained end to end. He also proposed the skip-net method to utilize higher resolution feature maps in the segmentation in FCN16s and FCN8s architectures. ERFNet uses residual connections and factorized convolutions in order to remain efficient while retaining remarkable accuracy.

Training and deployment of segmentation model: After exploring the above mentioned segmentation models, we finally narrowed down on FCN-Alexnet for semantic segmentation of the images. While Alexnet by itself is an image classification neural network architecture, it is transformed to a fully-convolutional segmentation model capable of per-pixel labeling segmentation network by convolutionalizing a pre-trained Alexnet. Training the model was done using Nvidia DIGITS (a webapp for training deep learning models). The data collected from the CFL site is loaded and using the network prototext, FCN-Alexnet (Caffe model) is retrained for 20 epochs. The performance of the trained model is visually verified by testing its performance on sample test images.

On confirming satisfactory segmentation of test images by

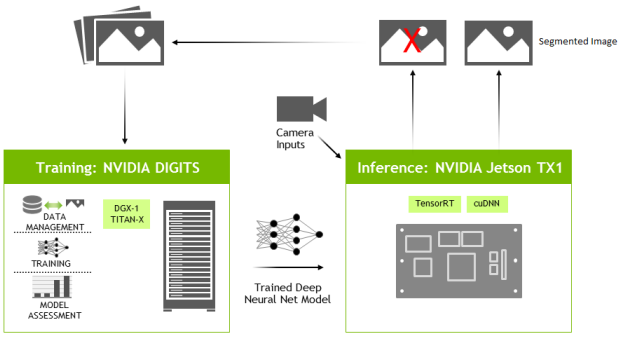

Fig. 16. DIGITs Workflow for Image Segmentation

Fig. 17. A sample image from the CFL site

Fig. 18. Corresponding image segmentation for the above sample

the trained model it's then deployed on Jetson TX1. To ensure compatibility with the TensorRT, deconvolution layer and crop layer are removed from the deploy prototext of the trained model.

Real-time camera feed from blimp: For the real time segmentation, we used ROS to subscribe to topics carrying compressed images from indoor blimp. The images were of the size 410 x 308. To be able to use the compressed image in our model, we had to resize it to 1280 x 720.

Segmentation Results: At every interval, 10 images, that were subscribed to, get saved in the memory of the Jetson. The segmentation model considers the 10th frame saved and segments this frame into two parts, the obstacle and the ground or movable path for the husky. The segmentation takes around 3 Hz on the Jetson TX1. Every segmented image is then saved for further use.

Occupancy map generation and Point Cloud Integration: After segmentation was completed, the next step in the process was to project it onto a occupancy map. The segmented image describes the occupancy state of each cell of the world in the color of the corresponding pixel. In the standard configuration, whiter pixels are free, blacker pixels are occupied, and pixels in between are unknown. A YAML file is used to specify the map meta-data and image specifics that are described below:

- image : Path to the image file containing the occupancy data; can be absolute, or relative to the location of the YAML file
- resolution : Resolution of the map, meters per pixel
- origin : The 2-D pose of the lower left pixel in the map, as (x, y)
- occupied_thresh : Pixels with occupancy probability greater than this threshold are considered completely occupied.
- free_thresh : Pixels with occupancy probability less than this threshold are considered completely free.
- negate : Whether the white or black free or occupied semantics should be reversed

The 'map_server' ROS package provides a command line utility to perform the above and the 2D occupancy grid can then be visualized on the RViz platform. The 3D point cloud information generated using VINS-MONO technique was provided by the SLAM-B team which could be projected on the 2D occupancy grid and visualized with the RViz platform.

C. What worked and didn't work

Table I highlights the different segmentation techniques along with their corresponding frameworks that we used for

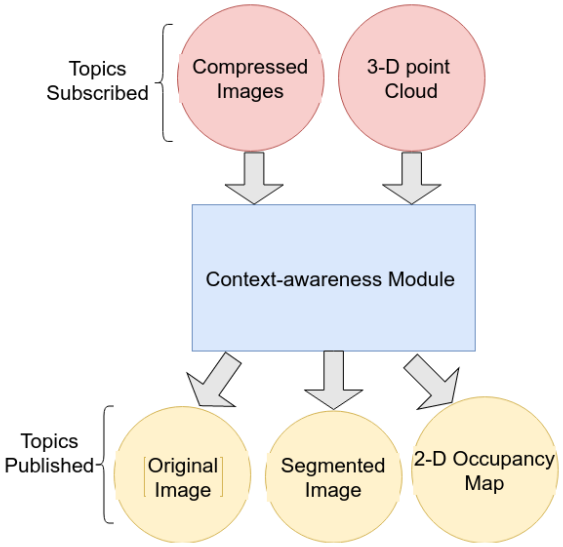

Fig. 19. ROS Workflow for CTX-B team

TABLE I
OVERVIEW OF SEGMENTATION ALGORITHMS AND FRAMEWORKS

Segmentation Technique	DL Framework	Issues Experienced
ERFnet	PyTorch	Unable to install Pytorch on Jetson
ENet	Torch and Lua	Out of Memory issue while running latest Torch/Lua framework
Scene Labeling using RCNN	Tensorflow	Unable to run Tensorflow due to insufficient RAM
AlexNet FCN	Caffe	Worked successfully

this project work.

D. Future Work and Extensions

In future, we plan to get a well defined Occupancy Grid through an image being publish by a monocular camera. We have looked into a paper by Andrew Ng which calculates a depth in an image. Using the technique mentioned in his paper, we can get a rough idea about the depth of an obstacle and get a perfect occupancy grid which can be directly used by husky to navigate. We also need to figure out how to increase the segmentation speed by trying newer approaches so that appreciable lag cannot be noted and the system works in near real time.

V. CONTEXT C

***Abstract*—This report describes the contributions of Context Awareness - C to add Computer vision system and help in the navigation of husky and outdoor blimp in construction environment. We provided segmented output of the given input image and help blimp and husky navigate safely through the environment. We have described in detail the effort we put in building contextual awareness for the robot.**

A. Introduction

Vision is one of the most crucial module in the design of an autonomous robotic system. The main objective of our project was to make the robot contextually aware of the surroundings. This can help the robot avoid obstacles in its path. In our case the robot is an outdoor blimp and have built a system to help the blimp and clear path husky a ground robot to navigate. The computation is carried out on Nvidia Jetson Tx1. Jetsons are embedded GPU devices and are very handy for mobile platforms.

The larger goal of this project is to achieve automation in construction by sending robots namely husky- a ground bot and two blimps. For the autonomous navigation of these robots we have developed a system to localize and map the surroundings.

Our main task was to carry out semantic segmentation on the images we received from the blimp. This report discusses in length the tasks carried out by the context awareness team for outdoor blimp. We have organized the report as follows: The next section is subdivided into installations and the how the actual segmentation was carried out on digits. The part B gives in depth insight of all the steps followed by us to carry out segmentation successfully.

B. Semantic segmentation

We tried different models for semantic segmentation. Most of the models failed to build on Jetson. We finally used Nvidia digits, a platform specially built for Jetson devices.

1) Installations:

- Caffe- A deep learning framework used by digits. Installing caffe was a major task. For NVidia digits we had to install a modified version of caffe known as nvcaffe. There were many dependencies that were needed to be installed first before building caffe. This was a cumbersome task and it took a lot of time and effort to get it right after many trials. After the successful installation of caffe it was easy to add and trian models on digits.
- Nvidia digits- Installation of Digits was comparatively simple. Digits tasks is to manage data, design and train neural networks on multi-GPU systems. It also monitors performance in real time with advanced visualizations, and helps in selecting the best performing model from the results browser for deployment. There were no issues faced while installing digits. The main challenge faced was while adding new dataset as this has to be in the same format as required by digits.

2) Segmentation with FCN Alexnet: Segmentation is based on image recognition, except the classifications occur at the pixel level as opposed to classifying entire images as with image recognition. We used fully convolutional network (FCN) Alexnet is the network topology for segmentation models with DIGITS and TensorRT. Digits has support to a fully-integrated segmentation workflow, which allowed us to create image segmentation datasets and visualize the output of a segmentation network. We also had DIGITS model store, a public on-line repository from which we could download network descriptions and pre-trained models. We followed the following steps for segmentation -

1) Dataset - We collected our own dataset for training. We captured images in CFL for the same. We captured images in different weather conditions including sunny, cloudy etc. We also captured images at different time of the day. Figures 20 21 and 22 are few of the images that we collected-

Fig. 20. dataset image1

Fig. 21. dataset image2

Fig. 22. dataset image3

2) Data augmentation - We augmented the data by adding fake shadows, histogram and contrast normalization etc.
3) Data annotation - We used the tool named 'supervise.ly' to annotate the images. We annotated the images into three classes namely:

   - Sky
   - Ground
   - Obstacle

   Sky was represented by blue ((0 0 255) on RGB color scale). Ground with green (0 255 0) and obstacle with red (255 0 0). Here are the sample annotated images. Figure 23 and 24 are few annotated images.

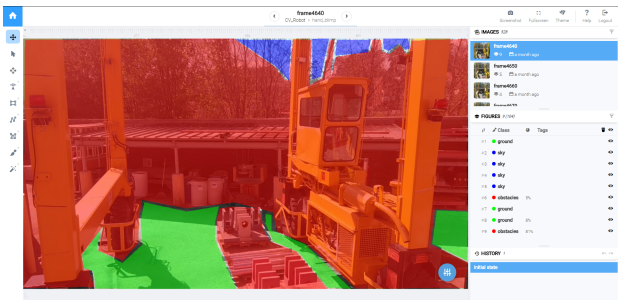

Fig. 23. Annotated image 1

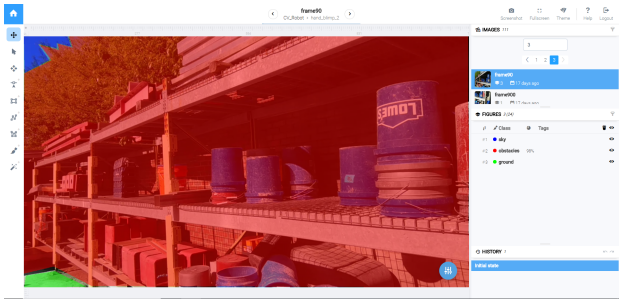

Fig. 24. Annotated image 2

4) Training with Digits- For training with digits we had to convert all the .json annotated images from supervisely to .png images. For this we found a python script that was run on Nvidia GTX-1060 laptop. The training screenshots are shown in figure 25 and 26.

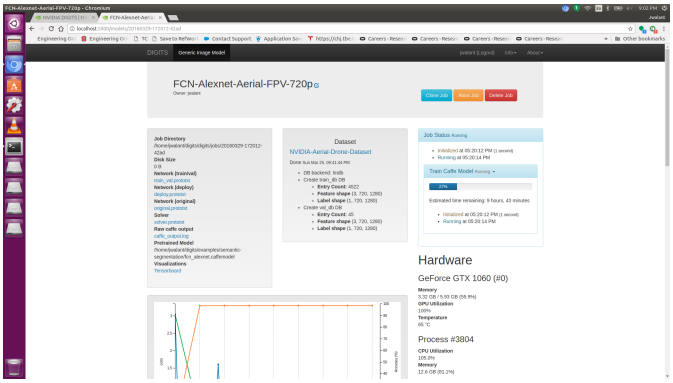

Fig. 25. Training on host

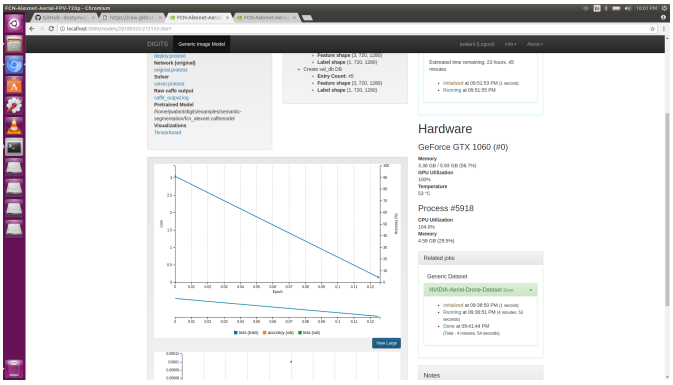

Fig. 26. Training on host

5) Inferencing with host- After training, digits gives an option to test or infer images on the newly trained model. This can help in determining the quality of the model before deploying it on the Jetson. The result that we obtained with our test images were quite satisfactory and hence we decided to move forward with Jetson. An image inferenced on host is shown in figure 27

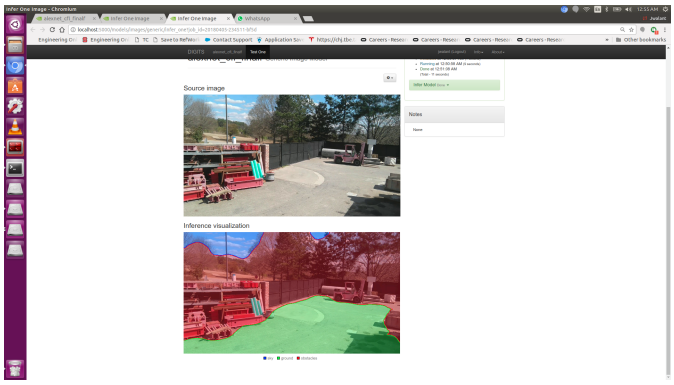

Fig. 27. Inferencing on host

Some images gave not so good results. As you can see in figure 28

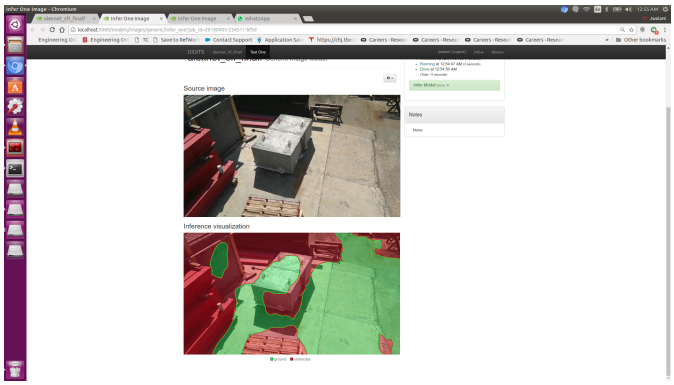

Fig. 28. Inferencing on host

6) Inferencing on Jetson - To test a custom segmentation network model snapshot on the Jetson, we used the command line interface to test the segnet-console program. We had to manually copy the model that was built on Digits that was installed on the host machine. We used ROS to subscribe to the images published by outdoor blimp and then used those images for segmentation. These are some of the segmentation outputs
7) Conversion of segmented output to binary The segmented output is to be projected on rviz to visualize the traversable area. Hence, we wrote a script to convert segmented output to binary. The sky and the obstacle is not the traversable area whereas, the ground is. Hence, we wrote a script to convert segmented output accordingly to binary. These are the binary outputs -
8) Projection on rviz We used map server, a ros package to project the binary image on rviz. We need to specify the binary image, orientation of the image and projection location for the map server in a .yaml file. The projected output will be as follows -

This ends the segmentation pipeline. We publish the segmented image and the map that is projected on rviz through ROS. Outdoor blimp/Husky can subscribe to them if they decide to use it for navigation.

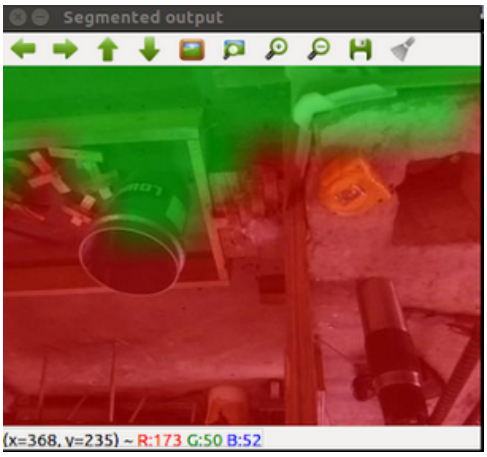

Fig. 29. Segmented output1

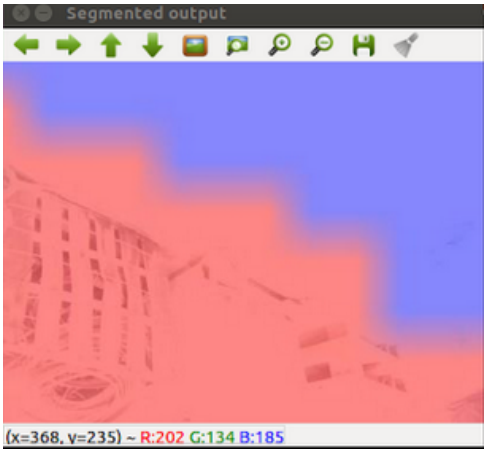

Fig. 30. Segmented output2

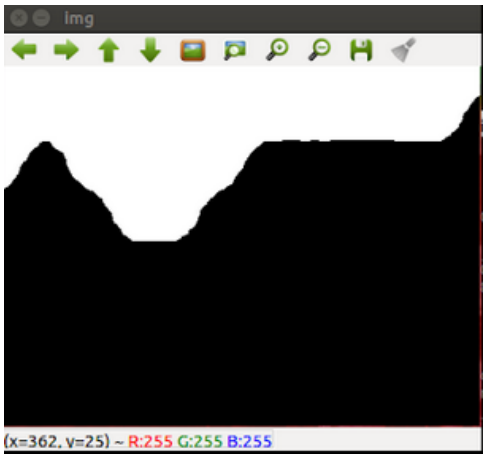

Fig. 31. Binary image

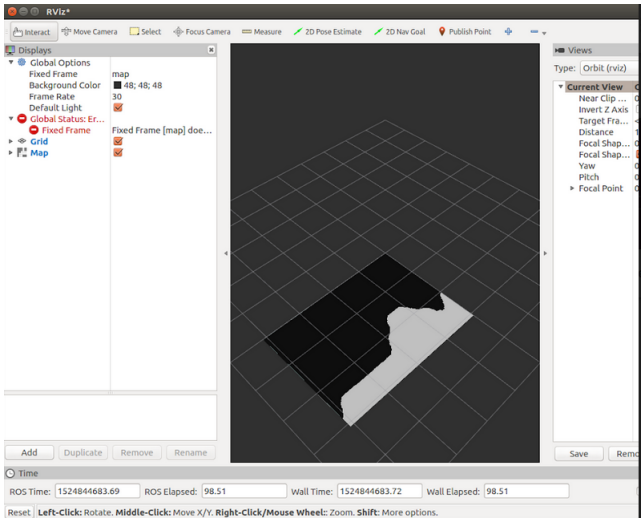

Fig. 32. Projection on rviz

3) Conclusion: We successfully created a pipeline which can be used in autonomous navigation of robots. This effort was integrated with other teams to realize the larger picture of automation on construction site. The segmentation results were good when compared to the ground truth. This work could be considered as a founding steps towards making the robots contextually aware in construction environments.

4) Future Work: The current work carried out by context awareness team could be continued in many ways.

- Training Data: The data collected currently is only from one construction site. This can be improved by collecting the data from multiple locations adding on to the current dataset.
- Depth Information: The current model does not consider depth information. This can be added in future to improve the projection of occupancy map.
- Speed: The computational speed of the model is quite slow. This can be improved by making the model lighter.

VI. SLAM A

A. Objective

As the primary goal of this course project was to map a construction site which will enable autonomous navigation of robotic agents, it was imperative that the localization and mapping algorithm was able to process the input in real time, with minimal lag and robust feature extraction and initialization strategy. Following are the main goals for the Monocular SLAM based on the Blimp teams

1) Create a mapping of the environment using Visual and Inertial.
2) Set up concrete localization of robotic agent.
3) Set up input data stream using ROS.
4) Synchronize input data streams.
5) Republish pose graph, and pointcloud to be used by the context awareness teams.
6) Integrate with both Blimp teams for end-to-end testing of the entire system.

B. Setting Up Environment

Even before we could start with our SLAM algorithm, it was important for us to install all the necessary libraries and softwares. Primarily, the important libraries included OpenCV(for Image Processing) and PCL(for generating Point Cloud) and ROS. As we were working on Jetson, we desired to install OpenCV with CUDA so that the GPU capabilities of Jetson could be utilized. Hence, we installed OpenCV 3.1.0 with CUDA support. However, on installing ROS, the openCV package of ROS overwrites the pre-installed OpenCV library in Python. Also, we installed full desktop version of ROS on Jetson TX1. For installing PCL library, we had to cross-compile it as it could not be directly installed. This involved installation of Eigen, Flann, Boost and VTK libraries with specific versions and building the library from source. Also, for VINs-Mono algorithm, Ceres Solver is required. Considering the dependency issues on the Jetson TX1 arch64 architecture, correct installation took upto 3 weeks.

C. Approach

The first step is to explore existing frameworks which align with our aforementioned objectives. Among the numerous SLAM implementations that we investigated initially, ORB-SLAM2 [1] seemed most promising for our monocular approach. Later, however, after we faced many speed issues with it, we shifted our focus to the monocular VINS-MONO [3] algorithm.

There are obvious pitfalls of using a monocular camera:

1) No depth information available for scale estimation.
2) Spatial aliasing and temporal aliasing due to rolling shutter camera.

Having said that, it is more challenging and interesting to implement a localization and mapping algorithm. The next section details the implementation of ORB-SLAM2 on Jetson TX1.

*1) **Camera Calibration**:* Camera calibration is the process of estimating the intrinsic parameters of the camera using images of a special calibration pattern. Intrinsic parameters deal with the camera's internal characteristics, such as focal length, skew, distortion and image center. The extrinsic parameters describe its position and orientation in the world.
The ROS camera_calibration package was used to perform calibration of the monocular camera as shown in figure 33.
The calibration output provided us with the distortion matrix, the intrinsic parameters and the camera matrix. These parameters were incorporated in the YAML file of the VINS Mono SLAM algorithm. The extrinsic parameters were calculated on the go and thus were not required to be precomputed for running the code.

*2) **ORB-SLAM2**:* ORB-SLAM2 is a real-time SLAM library for Monocular camera that computes the camera trajectory and a sparse 3D reconstruction. It is able to detect loops and re-localize the camera in real time. ORB-SLAM2 uses

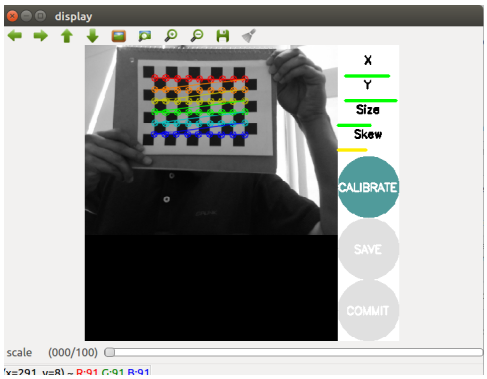

Fig. 33. ROS Camera Calibration

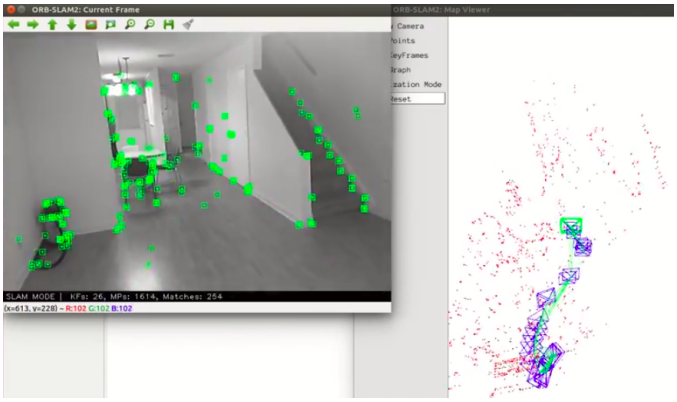

Fig. 35. ORB-SLAM2 Localization and Mapping

Oriented Fast and Rotated Brief features which are oriented multi-scale FAST corners with a 256 bits descriptor associated. They are relatively fast to compute and match, while they have good invariance to viewpoint.

Figure 34 provides a basic work-flow for ORB SLAM2 Algorithm. System consists of following important steps [2]:

1) Tracking and localizing the camera with respect to every frame using the matched features in local map.
2) Mapping and optimizing the local map using map points and local key frames.
3) Loop closure for detecting large loops and correcting the accumulated drift using pose-graph estimation.

The system has Vocabulary and Recognition Database for relocalization, in case of tracking failure (e.g., an occlusion) or for reinitialization in an already mapped scene, and for loop detection. The system uses the ORB features [2] for tracking, mapping, and place recognition tasks. These features are robust to rotation and scale and present a good invariance to camera autogain and autoexposure, and illumination changes.

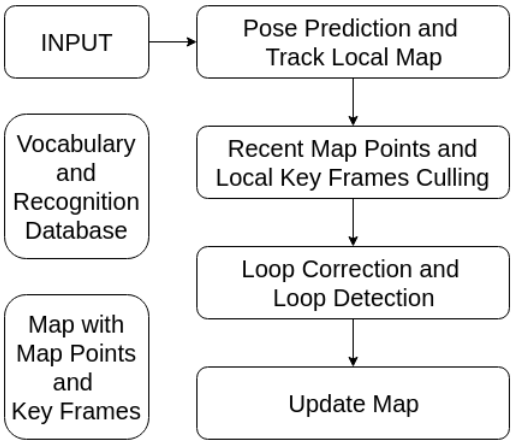

Fig. 34. ORB-SLAM2 Block Diagram

1) **Successes:** ORB-SLAM2 provided a robust localization path and a sparse point cloud when running on a workstation other than the Jetson TX1. Figure 35 shows the robustness of the algorithm with just Visual data when running on a personal workstation.
2) **Failures:** ORB-SLAM2 provided a robust localization and mapping strategy for a monocular camera. However, it underperformed on a realtime basis. The primary reason for this is the slow initialization of ORB features. The goal of initialization is to compute the relative movement between two frames to triangulate and extract an initial set of map points. These are then used to extract the relative scale of the scene with respect to the camera. This needs to be independent of the scene and should not require human intervention. ORB SLAM requires a feature rich scene, which has adequate lighting and almost zero lateral movement. Both of these conditions are too strict for a real time scenario where the blimps were looking down mostly on a a surface which have very few features. Also, while testing this algorithm with the Blimp team, it was not possible to have near zero motion, which caused a lot of image blur. Consequently, the time to initialization was no where near ideal on the TX1.
3) **Integration:** The integration part included subscribing to Image topic. Given the fact that ORB-SLAM2 has intensive computations, the throughput achieved was less than a frame a second. Hence, as a future scope of improvement, if the heavy ORB SLAM computations are processed on the on-board Maxwell GPU, we could potentially see a speed up of more than 3 times.

*3) **VINS Mono**:* As discussed above, ORB-SLAM2 was not scalable for our real-time requirements on the TX1. Further, it was susceptible to the initial camera state. Hence, a visual-inertial approach was required so that a feedback loop is created, making the estimation robust. After investigating further on SLAM algorithms that use visual as well as inertial data, we settled on VINS-Mono.

As VINS-Mono uses a loosely-coupled sensor fusion where IMU is treated as an independent module to assist vision-only pose estimates obtained from the visual structure from motion. Hence, the coordinate frames of the sensors have to

be calibrated for proper results. We consider that the direction of gravity is aligned with the z axis of the world frame.The body frame is defined to be the same as the IMU frame. The camera frame is defined to be different than IMU frame, hence we use both rotation matrices R and Hamilton quaternions q to represent rotation between body(IMU) and camera frame.

For each new incoming frame, previously tracked features are tracked using an innovative tracking algorithm called KLT method, This method assumes a local neighborhood of pixels for each key point, tracks the change in intensity value by calculating intensity gradient in x and y direction. It calculates the optical flow between neighboring frames by calculating a velocity vector (V_x, V_y), which satisfies:

$$I_x(q_n)V_x + I_y(q_n)V_y = -I_t(q_n) \tag{1}$$

where q_n are the pixels inside a neighborhood window, and $I_x(q_i), I_y(q_i), I_t(q_i)$ are the gradients of the image intensity with respect to $x, y,$ and t respectively. For extracting new features, a Harris corner detector is used for each new frame.

VINS also offers robust loop detection and closure when a scene is visited again. This is done on the basis of the history points accumulated by the estimator. When new Harris features match the history points, the estimator immediately senses a loop closure and continues further with this new information.

The raw gyroscope and accelerometer measurements from IMU, ŵ and â, are given by:

$$\hat{a_t} = a_t + b_{a_t} + n_a \tag{2}$$

$$\hat{w_t} = w_t + b_{w_t} + n_w \tag{3}$$

where, b_a = acceleration bias b_w = gyroscope bias and n_a and n_w are the additive Gaussian noise components for acceleration and gyroscope measurements respectively.

We assumed a stationary initial condition to start the monocular VINS estimator. However, this assumption proved inappropriate as initialization under motion is necessary in real-world application. Additionally, as the extrinsic parameters of the camera were being calibrated on the fly, it was required to provide initial rotation movement. This was a win-win situation as the initialization was fast and smooth as compared to ORB-SLAM2.

1) **Successes:** VINS Mono gave a very robust localization and a sparse point cloud along with the camera orientation. It was able to detect the history points and complete a loop closure. Also, the system was able to eliminate drifts in the path, in contrast to ORB SLAM 2, by utilizing a tightly-coupled re-localization module that seamlessly integrates the Visual-Inertial Odometry (VIO). VINS Mono incorporates IMU pre-integration for Gyroscope Bias Calibration which is then subsequently used for other metric initializations. This helps it to initialize faster compared to ORB SLAM 2. Another advantage of using VINS Mono was that it had the ability to calibrate the extrinsic parameters of the camera on the go. While the algorithm provides an option to the user to manually specify the extrinsic parameters, it also had the ability to calibrate it while taking the live video feed.
2) **Failures:** The VINS Mono algorithm best performs when the system uses a global shutter camera and a synchronized high-end IMU. In our case, the system lacked both. The Raspberry Pi camera is a rolling shutter camera because of which faster movements create distortion in the image captured. Also the IMU sensor is unsynchronized with the image feed. These two factors combined to give a poor performance by the algorithm. This hardware constraint, resulted in the localization to go wrong occasionally and subsequently affect the future point cloud generation.
   The only possible remedy to this issue was to use a sophisticated sensor which combined the capabilities of both the global shutter camera and a synchronized high-end IMU.
3) **Integration:** The final demonstration incorporated a complete integration between the SLAM and the Blimp teams. The camera and the IMU sensor were fixed together on both the Blimps. When the motion of the Blimp started, the camera feed and the IMU stream were initiated. This allowed the VINS Mono algorithm to start the localization and mapping as soon as the estimator completed its initialization step. As soon as the initialization was completed, the algorithm started to publish the generated point cloud for use by the Context Awareness teams as shown in Figure 40. The white points signify all the "history" points it accumulates as we move the camera around. The red points signify the points in the camera frame at the current time step.
   Figures 36 to 39 show the working demonstration of integration between the Blimp and the SLAM. As the orientation of the blimp changes, the corresponding orientation also changes in the SLAM output. Also, it can be seen that the surrounding area is getting mapped and the current location of the blimp is also being updated.

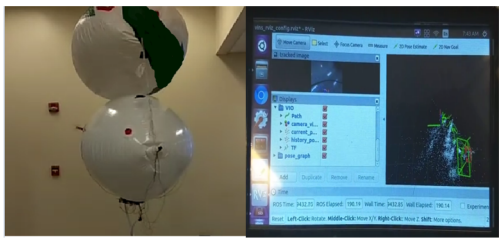

Fig. 36. Orientation-1

*4) **Future Work**:* For the future scope, few improvements are proposed.

- Replacing the Raspberry Pi (Rolling Shutter) camera with a global shutter camera. The rolling shutter camera adds distortion to the image when the camera is moved too

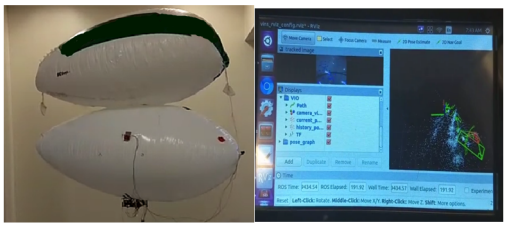

Fig. 37. Orientation-2

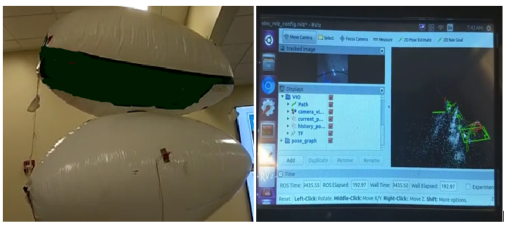

Fig. 38. Orientation-3

fast. This can be prevented by replacing it with a global shutter camera.

- The IMU and the camera feed are not synchronized as they originate from two distinct units. This can be avoided by using a high-end sensor which integrates both the IMU and the camera, thus removing the delay as much as possible.
- The code is currently not optimized to use CUDA on Jetson. The CPU version of the code adds latency during runtime owing to processing in the backend. This can be improved by incorporating the

VII. SLAM B

A. Goals

The goal of simultaneous localization and mapping is to take a mobile robot placed in an unknown environment and build a map of its environment while simultaneously tracking its location within the generated map.

Due to the division of labor within the class where our SLAM team focused on the Husky we had the following goals:

1) Implement a visual based SLAM algorithm for use with the Husky
    a) Investigate different types of visual SLAM (eg. monocular/stereo)
    b) Investigate sensors typically coupled with visual SLAM
    c) Investigate open source SLAM frameworks
    d) Implement and modify a SLAM framework for use on the Husky
    e) Generate a 2D occupancy grid map of the environment
    f) Generate a 3D point cloud of the environment
    g) Accurately localize the Husky within the map
    h) Investigate loop closure as a method of restoring a previously mapped environment and re-localizing the Husky within that stored environment

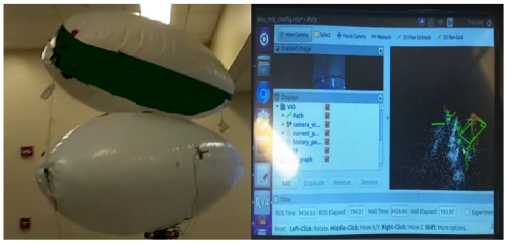

Fig. 39. Orientation-4

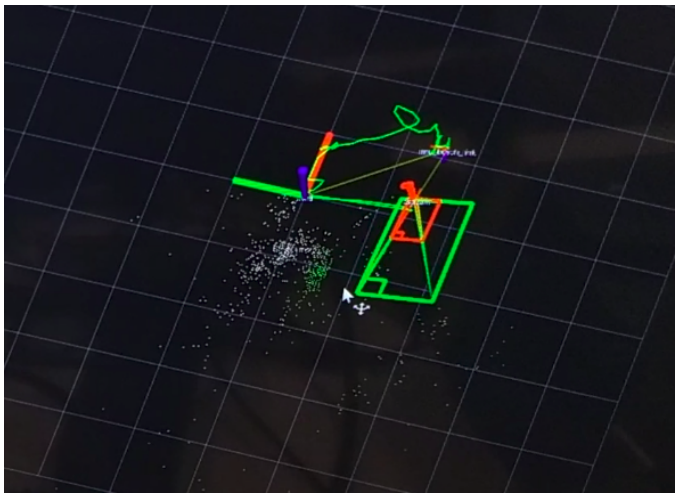

Fig. 40. Pointcloud

2) Ensure integrability of SLAM algorithms with other teams on this project
    a) Ensure we work fine as the ROS master of this project
    b) Ensure we can subscribe to the odometry data generated by the Husky team and use it in real time for our SLAM algorithm
    c) Ensure the generated 2D occupancy grid and 3D point cloud are being published as ROS topics by our JETSON
    d) Ensure the frame rate of our ROS topics meet the other teams' requirements
    e) Ensure other teams are capable of subscribing to our ROS topics
    f) Ensure generated mappings are suitable for navigation purposes

These goals are separated into two categories: one being tasks that we needed to perform for our module to function properly and the second being tasks needed for our module to integrate with other teams. In the next section we will describe the actions taken to achieve the goals and subgoals listed above.

B. Approaches Taken

First, we will cover a summary of our investigations into different visual SLAM algorithms. There are two primary visual SLAM techniques: monocular and stereo SLAM. The aim of

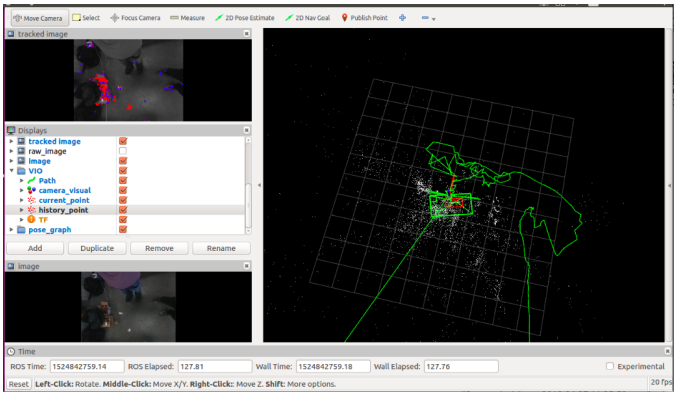

Fig. 41. VINS-Mono Localization and Mapping

both of these algorithms is to generate a 3D representation of the environment using visual data. Stereo SLAM algorithms use two cameras to generate the depth information through triangulation using the fixed distance between the two cameras. Monocular SLAM algorithms extract depth through indirect calculations based on features extracted from a single camera image.

Pros - Monocular

- The most important advantage that pure monocular SLAM has is the implementation simplicity of the hardware. This means that monocular SLAM is both cheaper and physically smaller than a similar stereo SLAM.
- Although the lack of depth information is one disadvantage of monocular SLAM, it still provides an advantage, which is the scale flexibility. Since there is no scale information in monocular camera, the SLAM system can switch smoothly from one scene to another without matching their scale information.

Cons - Monocular

- The software implementation in monocular SLAM systems is much more complicated than stereo SLAM. This is all because the hardware simplicity which provides no depth information. As a result, the algorithms for monocular SLAM are much more complex since they need to propose some methods to obtain depth information from a single image.
- Another serious problem of monocular SLAM is the pure rotation situation. Pure rotations will lead to large camera pose errors in SLAM system. When the camera rotates at its center, it will appear very similar to a large translation movement. Then the SLAM system will be confused about what the true 3D motion is, which will result in the computation of an incorrect camera pose.

Pros - Stereo

- The algorithm of for stereo SLAM system is much simpler than monocular SLAM. That is because the depth information is already obtained by the stereo camera. Stereo camera can directly calculate the depth information by comparing the difference between images from two cameras.

TABLE II
OVERVIEW OF SLAM FRAMEWORKS

	ORB-SLAM2	RTAB-MAP
Ease of Implementation	Compatibility Issues	Easy
Operation	C++/GUI	C++/GUI
Functionality	Good	Good

- Another advantage is the implementation of IMU sensor fusion. In the past visual SLAM + IMU information has been used as a method to improve performance over the basic visual SLAM algorithm. With the development of current technologies, the stereo camera can provide RGB-D images with IMU integrated data simultaneously. This is a marked improvement over using an additional IMU unit because it can provide both sensors' data in the same reference frame without any integration conflicts.

Cons - Stereo

- One problem of stereo camera is the baseline problem. To obtain depth information, the scene needs a decent baseline which is the spatial distance between the two cameras in the system. A larger space between the 2 cameras is required to set up a wider baseline which may not always be available.
- There are also some common problems for all visual SLAM algorithms such as the moving object problem, the blocking problem, and the light source interference problem. These problems arise because both monocular and stereo cameras are based on feature extraction from images. If the algorithm cannot get enough features from the image it will not work and the blocking, moving, and lighting will all cause some visual problem of the image which may lead to no or wrong features.

Sensor fusion of stereo SLAM with IMU sensors is particularly prevalent. We ended up implementing this technique to help improve the localization accuracy of our SLAM algorithm. Similarly, raw obstacle information associated with LIDAR has been demonstrated in many builds. This sensor could be integrated as well however LIDAR has issues in outdoor spaces which are the proposed target of this navigation system so integrating this sensor was not a high priority for our module. Finally, external odometry is typically paired with visual sensors to help localize a robot. In particular, the combination of IMU plus encoder information from the Husky team was integrated in our SLAM algorithm.

We investigated two SLAM frameworks primarily. These were the ORB-SLAM and RTAB-MAP open source implementations of visual SLAM algorithms. We primarily focused on functionality as well as ease of implementation which led us to select the RTAB-MAP

Table II shows a comparison between the frameworks. The pros and cons of each framework ultimately led us to utilize the RTAB-MAP framework for implementing visual SLAM due to its lack of compatibility issues that we discovered in ORB SLAM2 with our Jetson.

Figure 42 shows a still of a single image's point cloud

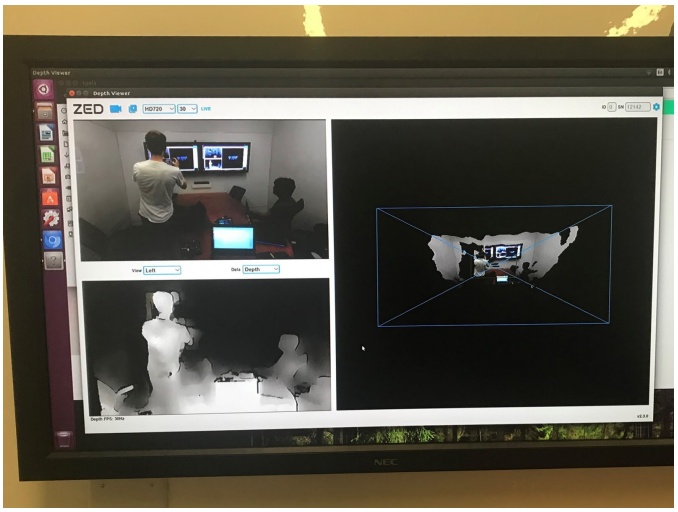

Fig. 42. Visualization of single image point cloud

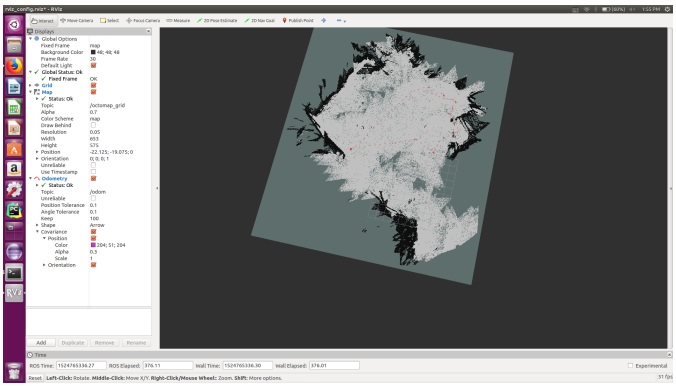

Fig. 43. Visualization of occupancy grid generated after mapping the construction site at CFL during testing.

where depth information is extracted via the stereo camera demonstrating the functionality of being able to generate 2D occupancy grids which are merely a sparse version of 3D point clouds. Figure 43 demonstrates a completed mapping of the CFL construction site during a testing run in conjunction with the Husky team. The red path seen in the image describes the path taken by the Husky during the generation of the map and shows that the husky was accurately localized within the mapping. This path was semi-supervised where the autonomous navigation was allowed to run for the most part except for cases where we needed to direct the Husky out of the path of oncoming vehicles to avoid any potential for disasters.

C. Software and libraries Used

We employed the ROS-kinetic (full desktop install) build with $rtabmap\_ros$ (a ROS package to run RTAB-MAP inside ros), $ZED\ SDK$ (official ZED camera software) and $zed\_ros\_wrapper$ (package to integrate ZED camera with ROS). Installation of these packages is performed within the Ubuntu 16.04 operating system.

D. Installation Instructions

First the installation of ROS full desktop version will go followed by creation of catkin workspace and the installation of $ZED\ SDK$. Then download the $zed\_ros\_wrapper$ package, which will help integrating ZED camera with ROS and $rtabmap\_ros$, copy the extracted package into src folder of catkin workspace folder and run $catkin\_make$, which will compile the whole catkin workspace with the $zed\_ros\_wrapper$.

One of the problems faced at the time of installation was that all the libraries used were open source and all are in continuous development which makes it hard to figure out which library versions are compatible with each other. This can lead to incompatible $zed\_ros\_wrapper$ with CUDA version. The latest $zed\_ros\_wrapper$ in GitHub requires a system with CUDA9.0 and applying Jetpack3.2 is the only way to install CUDA9.0 on Jetson TX1. However, the newest $zed\_ros\_wrapper$ is incompatible with the Jetpack3.2 system. So the best way is to use the combination of Jetpack3.1 and $zed\_ros\_wrapper$ published before 20th DEC, 2017.

To ensure full transparency the detailed instructions of a fresh installation process for our software is described below. It is unlikely you will need to undergo this process if you utilize the Jetson from last year as we have set up the system to function out of the box.

1) Flash Jetpack 3.1 onto the Jetson which will install Ubuntu 16.04 plus CUDA
2) Install ROS desktop full install and create a catkin workspace and compile it.
3) Install $rtabmap\_ros$ package which will be used to run rtabmap software
4) Install ZED SDK which will be used to run ZED camera
5) Download the $zed\_ros\_wrapper$ v1.0.0 or commit 05e47ff on github and extract it into the src folder of catkin workspace and compile the whole workspace again.
6) If the last compilation is completed without any errors then we have verified that the installation was successful and the tools can begin to be used.

E. Procedure

Once all the libraries are installed and working properly, ROS can be fired up to launch ZED camera and publish all the ZED topic ranging from $left\ camera\ data$ to $right\ camera\ data$ to $odometry\ data$. Then $rtabmap$ can be used to subscribe to those topics to create an occupancy grid map (2D) and point cloud (3D) and publish those topics too. These topics were used with $rviz$ to create and visualize those data and launch command for HUSKY navigation. Detailed use cases are shown below for transferability to future students.

- Setting up the Husky in Mapping mode
  - Power the Husky and Jetsons on
  - Connect to the Husky Wifi on your laptop

  - ssh into our Jetson using IP address: '192.168.0.11' with username and password of 'nvidia'
  - Run 'roscore' in the terminal to setup our Jetson as the ROS master
  - Initialize the Zed Camera by running 'roslaunch init_zed init_zed.launch' in the terminal
  - Initialize rtab-map in mapping mode by running 'roslaunch rtab_mapping rtab_mapping.launch' in the terminal
  - Mapping mode should now be enabled
- Setting up the Husky in Localization mode
  - Power the Husky and Jetsons on
  - Connect to the Husky Wifi on your laptop
  - ssh into our Jetson using IP address: '192.168.0.11' with username and password of 'nvidia'
  - Run 'roscore' in the terminal to setup our Jetson as the ROS master
  - Initialize the Zed Camera by running 'roslaunch init_zed init_zed.launch' in the terminal
  - Initialize rtab-map in localization mode by running 'roslaunch rtab_localize rtab_localize.launch' in the terminal
  - Localization mode will now be enabled and the script will begin comparing current frames to the stored database in an attempt to perform loop closure
- Visualizing the mapping
  - Setup the Husky in Mapping or Localization mode as shown above
  - Run 'rosrun rviz rviz' to open rviz on your laptop to visualize the results of the mapping/visualization
  - Use the GUI to visualize the SLAM mapping

F. End product

After setting up the Jetson as described above, we were able to create a very crisp 3-D point cloud and occupancy grid of the environment and were able to publish this data to other teams as a ROS topic. The Husky team then used this data to visualize an occupancy grid on $rviz$ and employ it in the husky to autonomously navigate from one point A to point B.

G. Failures and Successes

One of the main failures that we encountered was the inability to integrate with the context awareness team. Our plan was to send our video data to context team which will further divide it into frames and segment those images and identify paths and hurdles and then convert those final images into videos and send us back which will be used by us to form point cloud and occupancy grid map. The problem we faced was as follows:

- There was a considerable time delay between the data sent and data received.
- The frame rate we employed was 60 fps but the frame rate used by them was of 7 fps because of low computational power of Jetson when running their computer vision algorithms.
- There was a problem in integrating the processed frames with their adjacent depth value which was needed to form the point cloud and occupancy grid map.

The second failure that we encountered was that we were unable to read LIDAR sensor into our Jetson. Our Jetson was not able to detect the LIDAR even as an external detachable drive.

Now if we talk about successes then the main success was we were able to figure out the libraries and dependencies which were required to run CUDA and be compatible with each other. Successfully integrating with husky team was our second success and we were able to publish relevant data as the ros master for all the other teams as well. Finally, the SLAM algorithms in RTAB-MAP we implemented tended to have quite good performance and required very little tuning needed to

H. Future Work

In the future our goal is to combine the results shown at our final demonstration with segmented image data. To be more precise, we were planning to send our visual data to the context team which will take the video data, divide it into frames, and then segment those frames. These segmented frames will mark hurdles and paths with different colors, convert those segmented frames into video, and return the result for integration into the SLAM algorithm. Using this result we will make a new point cloud and occupancy grid using the segmented video data and the previously generated maps to provide the husky with a more accurate representation of areas to avoid which will aid its path finding.

VIII. Indoor Blimp

A. Goals

The main goal of the indoor blimp unit was to build an unmanned aerial vehicle which will navigate autonomously and monitor civil infrastructure systems while simultaneously interacting with other agents. The indoor blimp was intended to provide an aerial view of the site and feed the data to Husky, SLAM and context awareness teams to facilitate mapping and inference. As the task involved the development of a completely new system, thorough planning was done and stagewise targets were set . While some tasks were undertaken in parallel, some had a prerequisite of complete system and was scheduled for the end. The first step towards building an aerial blimp was to design the blimp from scratch considering the mechanical aspects. Secondly, the approach aimed towards development of sensors and actuator system while taking into consideration the electrical as well as software driver aspects of it. Finally, the development also targeted a basic navigation system that imparted the blimp the ability to navigate in the required environment. The tasks that were targeted by the Indoor blimp team can be summarized as follows:

1. Designing the flying mechanism consisting of blimp envelope and gondola and selecting the embedded platform for control and video streaming

2. Vision system design which deals with video streaming and pan-tilt mechanism
3. Non-vision based localization System design which deals with the sensor selection
4. Building an Indoor control and navigation pipeline .

B. Flying mechanism module

1) Envelope:

a) Approaches: For the aerial blimp it was essential to select an envelope size that would carry the payload but at the same time keeping its size small, to navigate the construction site and to keep the air drag as small as possible. The first stage of payload carrying estimation started with an Mylar envelope with a diameter of 36 inches. This envelope was filled with Helium and the weight carrying capability was seen to be 83 grams. Thus, in the first stage of design we planned to stack the envelopes to lift the desired payload. In the second stage of the development we planned either to built the envelope or find an envelope from a vendor of bigger dimension. We selected an envelope of polyurathene material (Fig. 44) which is ellipsoid in shape, diameter of 0.8 meters and a length of 1.7meters. When tested for payload capacity it was able to carry a payload around 230 grams

Fig. 44. Indoor Blimp Envelope

b) Failures:

- The envelope that the Indoor blimp team finalized was not able to navigate the doors given the size of the envelope. However this constraint was inevitable as the primary requirement was to meet the payload carrying capacity which demanded a bigger size envelope.
- The envelope was stacked to carry the payload. However this was required as the envelope was prone to leakage and deflation

c) Success: The Indoor Blimp team finally decided to carry the payload with two envelopes stacked.

2) Payload Carrier:

a) Approaches: In order to make it easier for blimp to carry all the components along with processor we decided to have a payload carrier which will serve as basic housing for all the parts. As weight was the main constraint for the team it was important to design the payload carrier as light as possible. For this, we 3D printed the module (Fig. 45) and motor holders (Fig. 46) . By this, we tried to achieve the required payload constraint. We tried two different versions for this one for Raspberry Pi 3 and other for Raspberry Zero. The overall payload including all the components and gondola for these two systems were 240 gms and 205 respectively. As Raspberry Pi Zero was not used due to reasons mentioned in other sections we dropped design for that model.

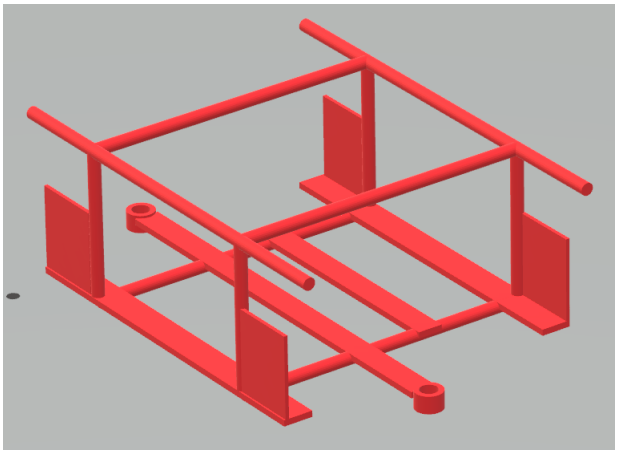

Fig. 45. Indoor Blimp Gondola CAD model

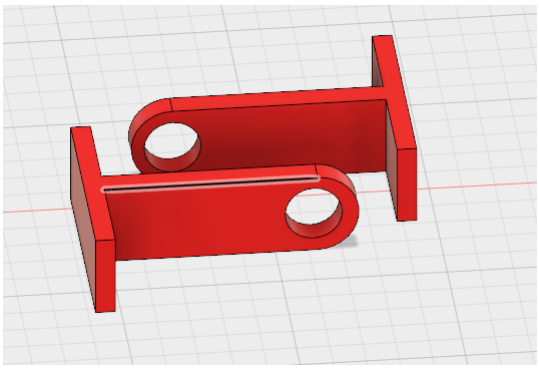

Fig. 46. Indoor blimp Motor Holders

b) Failures: Initially we built the model which had all four motors in it. But using this module didn't give us more rotational power. So we changed our model and incorporated only two motors in it and the other two motors were placed on blimp itself.

c) Success: Successfully designed the payload carrier 3D model with the maximum weight of only 27 grams.

C. Vision System module

1) Pi CAM:

a) Approaches: The vision system is an integral part of the indoor blimp unit which helps in providing data for the SLAM and Context awareness units to work. Selecting a camera which can support the other units by providing data with the right format at the same time which will not hinder the payload capacity of the blimp was a challenging

task. A stereo camera is always the best choice for a SLAM algorithm to generate accurate results since it captures the three-dimensional images with the depth information. But due to the limited payload capacity of the blimp, lifting a stereo camera was difficult since the weight of a stereo camera is coming up-to 150gm. Because of this reason, we were forced to use a monocular camera which could be lifted by the blimp. Finally, a Raspberrypi Camera Module V2 was selected for streaming video which was used by the other modules. The weight of the camera module V2 is 19gm. The image of the camera module is shown in Fig. 47

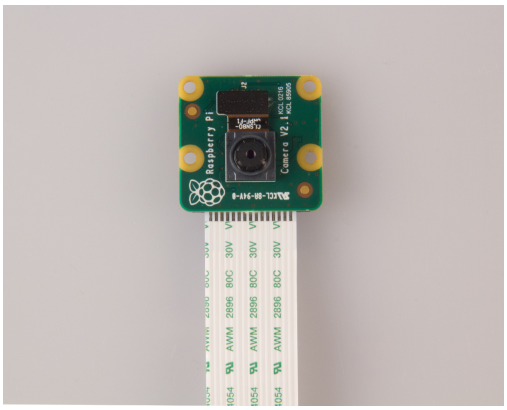

Fig. 47. Camera module V2

b) Failures: Initially, Raspberry pi zero was selected as the controller because of it's reduced weight. But the ros package which was selected in accordance with SLAM and context awareness units was not compatible with raspberry pi zero. So we had to shift to raspberry pi three.

c) Success: After shifting to raspberry pi three, we were able to achieve successful video transmission with the resolution of 410 X 308 with 30 frames per second and SLAM and context awareness units were able to use this data for their purpose.

2) Software Package:

a) Approaches: To begin with the video streaming we developed programs that allowed us to stream videos wirelessly using server client programs, like, netcat and VLC.The issue with this mechanism was the difficulty to extract the streamed data required for further processing by SLAM and CA team. Hence we decided to go for a ROS based videos streaming approach. The ROS package used for video streaming is raspicam_node with the node name 'Compressed_Image' and the topic name 'image_indoor_blimp_compressed'

b) Failures: We did not face any issues from the software side for transmitting the video frames.

c) Success: We were successfully able to publish video frames using the 'raspicam_node' package and SLAM and Context awareness units were able to use this data for creating the map and image segmentation respectively.

D. Pan and tilt module

1) Servo gimbal mechanism:

a) Approaches: In-order to give SLAM and CTX team images at required desired angle we used servo gimbal mechanism. This not only gave us freedom to set the camera at an appropriate angle required to get perfect image but also it kept camera stabilized. First we mechanically assemble various free parts including two servos of 9g each to get a module with which we can play around. Then we implemented only pan mechanism with the angle increment of about 10 degrees in each step. Same logic was followed for tilt mechanism. However as we were able to maintain constant angle required by the SLAM and CTX team with the servo gimbal mechanism, we in agreement with the other two teams excluded pan-tilt mechanism from the final demo.

b) Failures: It was observed that once the gimbal completed yaw and pitch movement as per the given angle and reached steady state, it kept vibrating with some noise.

c) Success: After optimizing hardware assembly of gimbal we were able to get smooth motion for pan and tilt mechanism.

2) Software Package:

a) Description: A ROS package was developed for the Pan-Tilt mechanism. It basically consists of two program. The first program publishes the command for increment and decrement of the angle while the second program subscribes to the command and executes the action within a call-back function.

The package 'servo' subscribes to the commands for controlling the Servo motor for pan and tilt using the node 'servo_control' with the topic name '/indoor_blimp/servo/'.

b) Failures: There were no issues with the developed software package for servo-gimbal.

c) Success: The Indoor Blimp team successfully implemented the ROS package for Servo-gimbal and published the respective topics.

E. Localization module

1) IMU:

a) Approaches: Accurate localization is necessary for self navigation of blimp and precise mapping of environment using SLAM. As GPS results are not accurate in indoor environment, it was decided to use inertial measurement unit (IMU) for localizing the blimp. Additionally, the inertial measurements from IMU together with the camera data when fused with visual data aids better localization and mapping using SLAM. Bosch BNO055 IMU has 9 degrees of freedom and performs sensor fusion within to provide accurate results. The image of BNO055 is shown in figure 48. BNO055 provides absolute orientation (Euler and quaternion angle), angular velocity and linear acceleration at 100 Hz rate suiting the needs for blimp systems. It uses high speed ARM Cortex-M0 based processor with minimal power consumption and is recommended by Adafruit industries for usage on raspberry pi systems.

b) Failures: IMU was configured in UART mode and at times the port failed to open on first program call. However, the issue was observed only during the initial execution and a rerun resolves the issue.

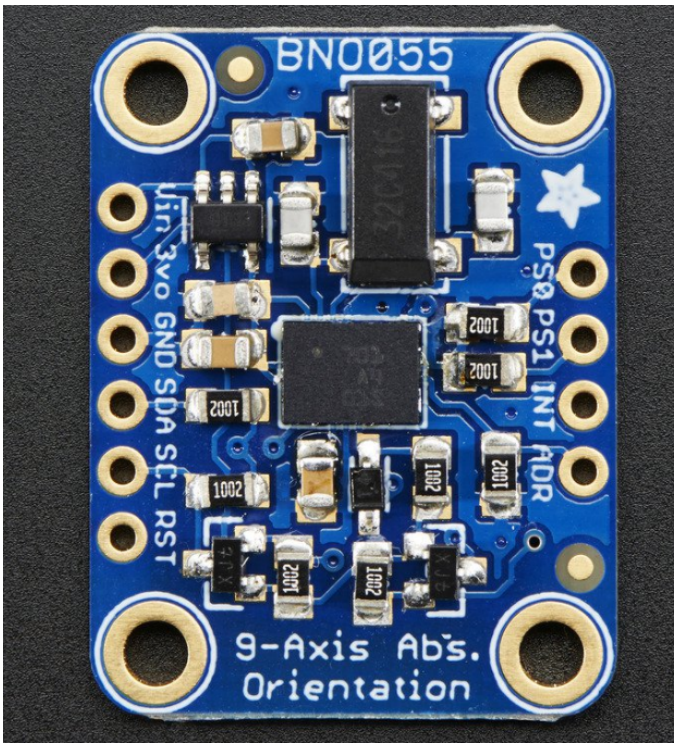

Fig. 48. IMU Adafruit BNO055

c) Success: BNO055 was stable and perfectly apt for indoor blimp system. The output data are reliable and accurate.

2) Software Package:

a) Approaches: Initially, a python script was implemented to test and calibrate IMU. Following the decision to use ROS as the common platform to ease integration tasks, a new ROS package was made for IMU. The IMU node publishes orientation, angular velocity, linear acceleration, yaw angle(Euler) temperature and magnetic field data. Although, RViz provides a good visualization for IMU orientation,another program with better visualization for IMU orientation in a 3D layout was implemented for demonstration purpose.

b) Failures: Initially, the data published by IMU was not in sync with camera data and hence was affecting the performance of SLAM system. A new package was created to launch both IMU and camera nodes simultaneously to publish synchronized data. Also, the calibration had to be performed every time system is restarted. However, a program was found in Adafruit tutorial to save calibration data to system and load the same thereafter.

c) Success: The published IMU orientation data was highly accurate and real time. The data accuracy was compared by rotating the imu physically to different angles and comparing the results in published data, external compass and visualization tool. The synchronous data feed from IMU and camera was tested in coordination with the SLAM team. The data transmission was real-time, synchronous and with negligible delay. The precise yaw control mechanism, wherein the IMU yaw data is subscribed, further confirmed the accuracy of the data.

F. Obstacle detection module

1) Ultrasonic sensor:

a) Approaches: Obstacle detection is achieved using ultrasonic sensors which are attached to the envelope surface. The main objective of this module is to detect obstacles which are coming in-front of the blimp and to prevent the blimp from moving further when an obstacle is detected. We selected LV-MaxSonar-EZ1 Ultrasonic Range Finder for detecting obstacles. It measures distance by sending out sound waves and listening the same waves after reflection form the obstacle. By recording the elapsed time between the transmitted and received waves, it is possible to calculate the distance between the sonar sensor and the object. We tried different methodology for this like UART, using RC circuit, interfacing it with Arduino to get results. But we finally implemented PWM based distance measurement using PWM pin of ultrasonic sensor.

b) Failures: If there are more then one ultrasonic sensors in vicinity of each other within some specified range they can interfere with each other and can give erroneous reading. With the power change the accuracy of ultrasonic ranger decreases. Also if gave wrong readings if the obstacle was in dead-band of the sensor.

c) Success: By placing the ultrasonic apart carefully and powering them equally we were able to get almost perfect results.

2) Software Package:

a) Description: The package 'ultrasonic' publishes the ultrasonic measurements using the node' talker' with the topic name 'ultrasonic'.

b) Failures: For publishing the ultrasonic sensor measurements using ROS packages, we did not face any issues.

c) Success: The ultrasonic values which are measured are published using a simple ROS publisher code which was used for controlling the motors.

G. Supply regulation and electrical circuitry modules

1) PCB:

a) Approaches: To make system more robust and clean we went on designing a PCB with EAGLE software. Our plan was to incorporate all the components on it along with voltage regulator circuit to build a stand alone independent system. First we tried using general purpose board (no printing) and soldering to bring together all components. We then moved on to printing the board for our second try. Finally we also incorporated voltage regulator to complete our design. These are shown in Fig.49 and Fig.50.

b) Failures: While working with general purpose board soldering was difficult as there were more connections than expected. The copper layer on the PCB was very thin so we had to do multiple iteration of it as a slight mistake soldering would rip off the copper layer.

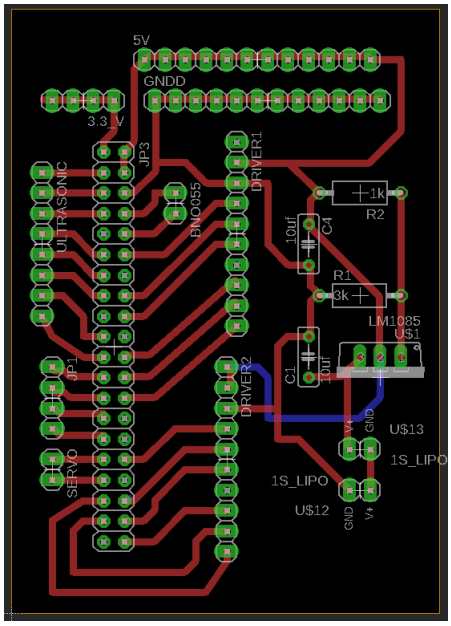

Fig. 49. Indoor blimp PCB

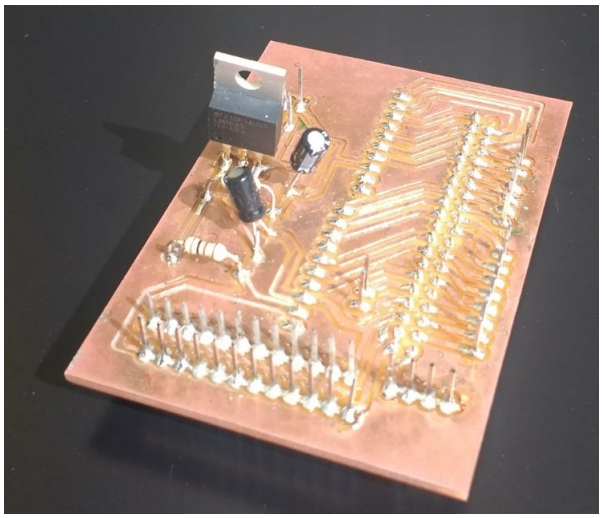

Fig. 50. Final soldered PCB

c) Success: Got a working PCB designed in college makerspace lab itself with reliable and consistent performance.

2) Supply Regulation:

a) Approaches: Initially, we thought of supplying the Raspberry Pi with battery and use this for all the other components. But this did not supply sufficient power when all the components were used at the same time and there was decrease in performance. So we decided to go regulated supply using voltage regulator IC.

b) Failures: Initially chose wrong IC for our voltage regulator, but apart from that things were easy and smooth.

c) Success: Successfully designed a voltage regulator circuit with 5V and 3A output property.

H. Motor Propeller system

1) Motor controller:

a) Approaches: For motor control we first tried using L293D IC. Then we switched to TB6612 breakout board as shown in Fig.51, it gave 1.2 A of current capacity and worked very well for motors 4.5V and above. Also this board is easy to use and plug and play type of device which is much better than using the through hole IC's.

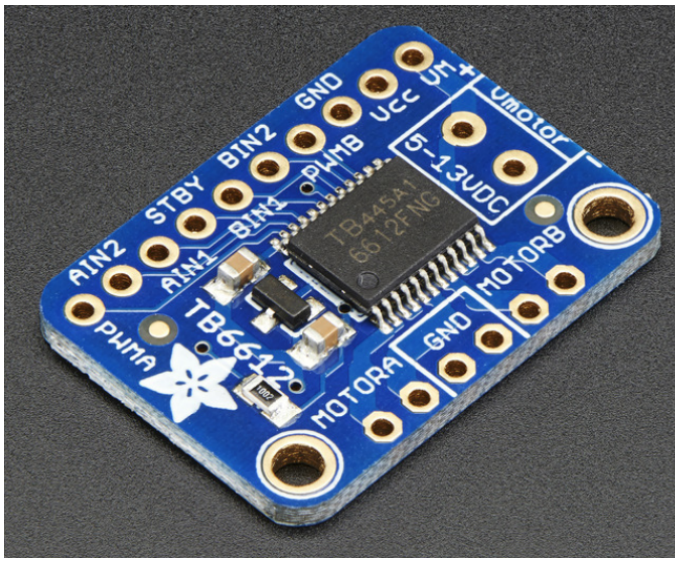

Fig. 51. TB6612 Motor Driver IC

b) Failures: Due to faulty voltage given at the input of motor driver we didn't get required result at its output and also it damaged our breakout board.

c) Success: With careful application of input voltage and proper configuration with PCB we were able to manipulate motor by running PWM signal (from Raspberry Pi) through TB6612 board.

2) Motor and propellers:

a) Approaches: In the initial design stage two motor-propellers were selected for forward-backward motion and one for the upward downward motion. However given the weight of payload and the size of envelope we selected two motor-propellers for the upward-downward motion as well. The aerial blimp system is not suppose to perform any high dynamic maneuver. Hence low KV motors were selected which draws less current for a certain rpm. Accordingly, the propellers were also selected with small pitch angle. The resulting motor-propeller system was sufficient to provide desired navigation but still being lightweight not to increase the payload weight.

b) Failures: We did not face any failures for controlling the blimp using the motors.

c) Success: The designed motor-propeller system was successfully able to navigate the Blimp

I. Control and navigation module

1) Velocity control:

a) Approaches: The blimp navigation control were implemented for yaw motion and up-down motion. The yaw motion was imitated from the differential drive mechanism of a two-wheel robot that allowed the system to make a right or left turn. The yaw motion and the up down motion were implemented using the open loop velocity control.

b) Failures: For the blimp closed loop velocity control was not implemented. This was due to lack of an additional data apart from the linear acceleration from the IMU, like GPS or feedback from SLAM that actually fuses the IMU and camera data to estimate actual velocity. Also as we were not able to implement close loop control hence we were not able to compensate the motion due to inertia. Like giving a stop command failed to halt the system immediately.

c) Success: The Indoor blimp team implemented the navigation with open-loop velocity control.

2) Yaw control:

a) Approaches: During the initial stage we implemented the yaw motion with open loop velocity control. But, our need was to implement the control were blimp needs to turn by a specific desired angle (for example turn right by 90 degree). For this we designed a PID controller were values from the IMU were taken to control motors which helped blimp to turn by specified angle.

b) Failures: Initial tuning of PID gave a the jerking movement to blimp. Also the tuning was difficult since slight change in dynamics of system would have different values of PID parameters and system will then not give same results for previous tuned values.

c) Success: After fine tuning of PID were able to achieve both absolute and relative yaw control for the blimp.

3) Software Package:

a) Approaches: The indoor blimp unit subscribes to the topic '/indoor_Blimp/setpoint_position' which gives a set point for the yaw control and the open loop control commands.

b) Failures: We were not able to implement the software for controlling the forward-backward navigation movements using a closed loop approach due to lack of time and other software issues

c) Success: With the help of the commands received from the above mentioned ROS topic, the indoor blimp unit was able to succesfully navigate using PID based yaw control and open loop based forward-backward, left-right, up-down motions.

J. Future Works

- For the envelope design, we can modify the dimension which meets the size requirement of indoor navigation and get more payload capacity.
- Can implement velocity control in closed loop system by integrating values with various different data (from GPS, SLAM).
- Try to get the system working with Raspberry Pi Zero so that the overall system weight reduces which is one of the most vital thing for blimp.

K. Integration

The major integration for indoor blimp unit is with the SLAM and Context awareness. Indoor blimp unit publishes video frames in the form of compressed and raw format which is subscribed by the SLAM unit for generating the map and localization.

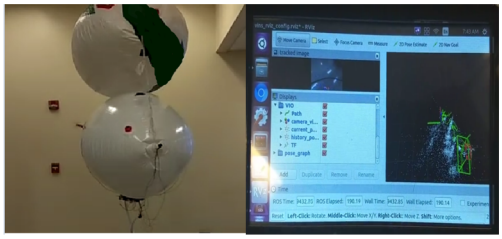

Fig. 52.

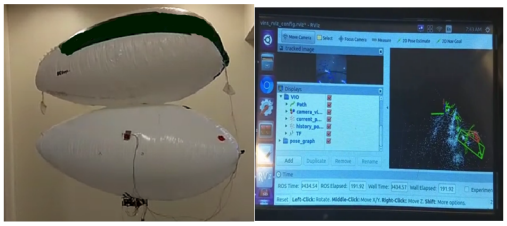

Fig. 53.

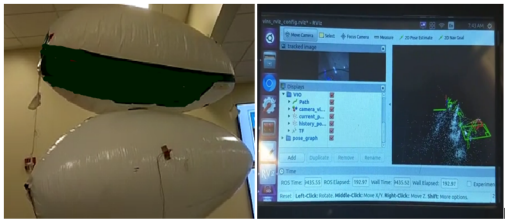

Fig. 54.

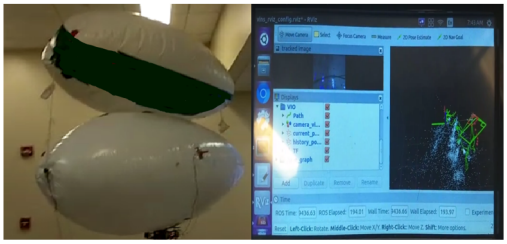

Fig. 55.

This is shown in figure 52 to 55 The same data is used by the context awareness unit for segmenting the images and detecting obstacles on them.

IX. Outdoor Blimp

A. Goals

The overall vision of the outdoor blimp system was to navigate autonomously to monitor projects and activities in a construction site. The goals of the team were the following:

- Design of the hardware and software of the system.
- Be able to operate for 30 minutes.
- Have a vision system to stream data from the system to other modules.
- Have a non-vision system for localize its location.
- Move and navigate autonomously.

The sections below discuss the various modules that were used in attempt to achieve these goals.

B. Flying mechanism module

1) Envelope:

a) Approaches: In order to be able to lift the payload carrier and parts, it is necessary to find the correct dimensions for the envelope. If the envelope is bigger than the required size, it is hard to navigate it in construction site and it will not be stable in windy conditions. If the envelope is smaller than the required size, the envelope is not able to lift the payload carrier. It was one of the hardest challenges of the outdoor blimp team because of high numbers of variables such as helium purity, payload carrier weight, and the shape and material of the envelope itself.

At the beginning the outdoor blimp team decided to find the right material and weld it in order to make a custom-shape envelope. finally, through trial-and-error the outdoor blimp team decided to buy a professionally-made envelope from a vendor.

b) Failures: The outdoor blimp team tried to find the correct weldable material for the envelope, but after several tries they could not find the correct material. Materials had the following issues.

- Material was not weldable.
- Material was too much heavy.
- Material was too thin.
- Envelope leakage after welding.

c) Success: Finally, outdoor blimp team decided to buy the blimp envelope from an online vendor. Since a single envelope was not able to provide enough lift, the outdoor blimp team stacked two balloons in order to generate the required lift.

2) Payload Carrier:

a) Approaches: In order to make attach all of the components such as the Raspberry Pi 3 and other sensors, it is necessary to have a light payload carrier that can simply attach all the components to the blimp. In order to have a light-weight and durable carrier, the outdoor blimp team decided to design and 3D print the payload carrier.

The reason behind having separated side motor modules is to generate maximum torque. Increasing the distance between side motors can simply generate more rotational power.

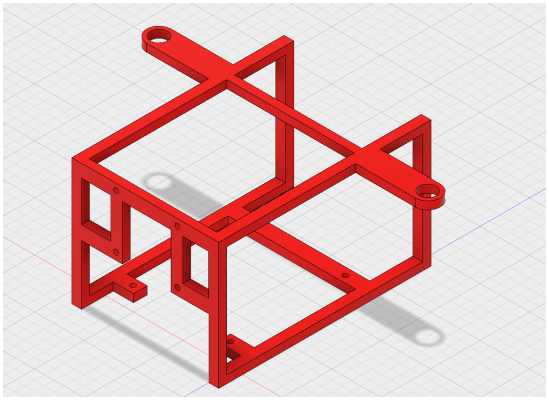

Fig. 56. Payload carrier CAD model.

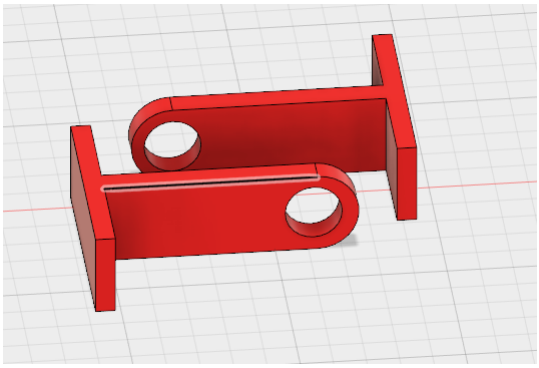

Fig. 57. Side motor holders.

b) Failures: The payload carrier design went through multiple iterations and prints in order to achieve the lightest weight while still retaining its strength. As well, the side motor holders had issues with the length of the stem which was causing the propeller to contact the envelope. These issues were resolved through iterative designs of the system.

c) Success: The final payload carrier was designed and printed to maximize the following characteristics: ease of attachment, lightweight, and strength. The final payload carrier weighed 20 grams.

C. Processors

1) Microcontroller:

a) Approaches: For the outdoor blimp system, a Raspberry Pi Zero and a Raspberry Pi 3 were tested to act as the microcontroller on-board the blimp system. Although the Raspberry Pi Zero weighs less and consumes less power, the Raspberry Pi 3 has double the SDRAM, has a higher clock speed, and is able to be configured with ROS. Both systems have in-build WIFI and Bluetooth modules, micro SD card ports, GPIO ports, and camera ports.

b) Failures: The failure of the microcontroller came when using the Raspberry Pi Zero. The system did not have enough RAM in order to run all the necessary threads to publish images, IMU, and GPS data. As well, the Raspberry Pi Zero was not able to be installed easily with the Robotic

Operating System. The architecture of the system was not compatible with the default ROS software.

c) Success: The success of the on-board microcontroller came from switching to the Raspberry Pi 3. A pre-built image of ROS allowed the outdoor blimp team to begin development almost immediately. As well, the 1 GB RAM allowed the microcontroller to handle most threads with little latency. The threads are still slow for images at around 10 FPS, but the system is more reliable than the Raspberry Pi Zero.

2) Graphics Processors:

a) Approaches: The Jetson TX1 was utilized to handle running the roscore and republishing the images from the microcontroller on the blimp. The Jetson TX1 has the processing power to service all nodes subscribed to the published images.

b) Success: The Jetson TX1 was able to utilize ROS and OpenCV without any issues. The GPU was able to handle the load of the images from the blimp system.

D. Vision System module

1) Camera:

a) Approaches: An ArduCam and RaspiCam were both tested for the system. Both cameras were able to meet the requirements for capturing images at specified frame rates and resolution.

b) Failures: The main failure for the hardware of the system was camera port of the Raspberry Pi. The outdoor blimp team faced an issue multiple times where the microcontroller could not interface wit the camera due to hardware issues on the port. This could be caused due to short-circuiting the port when attaching and detaching the camera from the Raspberry Pi.

c) Success: The system was able to capture images that met the requirements of both CTX and SLAM teams.

2) Software Package:

a) Approaches: Two separate ROS packages were tested for the imaging system. A raw image node and an OpenCV node were tested and integrated into the system. These imaging systems allowed publishing of both compressed and uncompressed images for CTX and SLAM systems to use. As well, the parameters for the camera were calculated and stored in a .yaml file in the file system. The final system used the Jetson TX1 to republish the images to ensure that processing power was not wasted on publishing multiple images for multiple subscribers. The nodes for the Raspberry Pi were /Outdoor_Blimp/temporary/image_raw and /Outdoor_Blimp/temporary/image_compressed. The republished nodes from the Jetson TX1 were /Outdoor_Blimp/image_raw and /Outdoor_Blimp/image_compressed.

b) Failures: One of the failures of the system was the processing power of transmitting the images concurrently with other processes. The frame rate was dropped to around 10 FPS at 640x480 resolution for the image capture to ensure images did not back up and stall the pipeline when transmitting the images. This failure was due to the limits of the processor to handle image transmission concurrently with other processes and sensor transmissions.

Another failure was network issues in transmitting the images. With the number of nodes subscribed to the system, the Raspberry Pi was not able to successfully transmit multiple copies of the images to all the subscribers. This issue was solved by utilizing the Jetson TX1 to republish the images from the Raspberry Pi and have all the nodes in the network subscribe to the Jetson for obtaining blimp images.

c) Success: The success of the software nodes were that they were able to transmit the images in near real-time. The latency of the images were low for the compressed images. As well, another success was that the format of the node allowed for the capture and transmission of the images directly through OpenCV. This allowed other teams and systems to utilize OpenCV directly when using the system's images.

E. Pan and Tilt Module

1) Servo Gimbal Mechanism:

a) Approaches: In order to provide SLAM and CTX teams with footage at various angles and in order to stabilize the camera from wind effects, the outdoor blimp used a pan-and-tilt servo gimbal. The gimbal was required to respond rapidly to changes in attitude and maintain a heading within 5 degrees of the setpoint in pitch and roll. It used two 9g micro servos, one to control the pitch of the camera and the other to control roll. In order to provide accurate information about the heading of the camera, the outdoor blimp included an IMU on the pan-and-tilt mechanism in-axis with the camera. Control of the servo gimbal was done through PWM signals directly from the Raspberry Pi 3. The outdoor blimp team determined that in order to maintain the accuracy parameters given by the SLAM team, a feed-forward control loop based off the IMU located on the payload carrier was sufficient. The outdoor blimp team created a calibration curve mapping IMU readings to the appropriate servo pulse to keep the servo gimbaled to the setpoint.

b) Failures: The primary failure of the pan-tilt mechanism was its lack of integration into the rest of the system and lack of testing for setpoints other than straight down. Although there was a ROS message available to control the setpoint, outdoor blimp was not able to verify its accuracy and SLAM team was not able to make use of it, so that functionality was removed for the final demo.

Another failure of the system was in the response rate. Due to only being able to poll the IMU at 10 Hz, the pan-tilt mechanism was only able to respond at 10 Hz. Another improvement could have been made to the gimbal by utilizing feedback from the IMU on the camera itself to keep the camera pointed exactly at a set point.

c) Success: The gimbal functioned as specified when directed to point straight down. It was able to respond to the full range of motion of the blimp and responded to those changes. It kept pointed down within a tight margin. The outdoor blimp team was successfully able to send setpoints through the ROS node to send the desired attitude of the camera, and were able to successfully publish the attitude of the camera through the IMU.

2) Software Package:

a) Approaches: Two ROS nodes were created for use with the pan-tilt mechanism. The input for setpoint control was shared with the yaw setpoint control in /Outdoor_Blimp/setpoint_position, and the IMU node was created at /Outdoor_Blimp/camera/imu/pose and /Outdoor_Blimp/camera/imu/rpy.

b) Failures: These ROS nodes were never integrated with the SLAM team, and so were never tested in integration.

c) Success: The IMU on the camera published data smoothly and responded to input from the setpoint ROS node.

F. Sensors

1) IMU:

a) Approaches: In order for the SLAM teams to localize the blimp and the images, IMU sensors were necessary on the blimp to understand the orientation of the blimp in space. The outdoor blimp team researched and tested the Digilent 9-DOF with barometer PMOD Nav and the Adafruit BNO055 9-DOF IMU. The system needed 9 degrees of freedom from an accelerometer, gyroscope, and magnetometer in order to obtain the most accurate IMU values.

b) Failures: In research and testing, the outdoor blimp team discovered that the Digilent 9-DOF IMU did not have an on-board digital filter. The Adafruit BNO055 has a built-in ARM M0 processor that filters the signal and stores the values in data registers. Since the team did not have enough development time to create a digital filter, the team decided to use the BNO055 for the system. Another issue with the IMU system was calibration. Since the sensor needed to be calibrated every time the device was turned on, the outdoor blimp team calibrated the system and loaded the values into the system at run-time. This way, the calibration of the accelerometer, gyroscope, and magnetometer would already be completed.

c) Success: The success of the IMU was found with the Adafruit BNO055. The system was able to calculate Euler angles, quaternions, and self-calibrate. The IMU gave the system the accuracy necessary for the SLAM and CTX algorithms. Two IMU sensors were utilized for the system, one for the payload carrier, and another for the camera system.

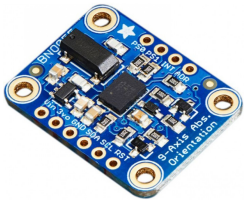

Fig. 58. Adafruit BNO055 Inertial Measurement Unit.

2) GPS:

a) Approaches: The outdoor blimp system utilized the Adafruit GPS system which uses a PA6H1F1702 GPS receiver. The system was tested and utilized in the system to obtain the GPS coordinates of the blimp while navigating outside.

b) Failures: A failure of this system was that the GPS values would sometimes not obtain valid GPS lock, and the GPS would return GPS values of zero. This needed to be parsed in order to ensure that the system was not given invalid GPS coordinates.

As well, since the CTX and SLAM teams did not require GPS data, the system was disabled for the final demo.

c) Success: The success of the system was that the blimp was able to localize its position on earth up to 3 meters, and the receiver was reliable enough for our system.

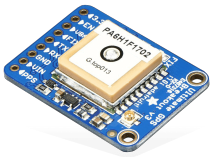

Fig. 59. Adafruit GPS Unit.

3) Barometer:

a) Approaches: A Digilent 9-DOF with Barometer PMOD Nav and a Diymore BMP280 were tested for the barometer system. Since, the Adafruit BNO055 IMU was selected for the IMU, the Digilent system was not necessary for the outdoor blimp system. As well, since the Diymore board was smaller, the team decided to use the BMP280 barometer for altitude measurements.

b) Failures: Since none of the CTX and SLAM algorithms did not require a barometer, the software for the system was not fully integrated into the outdoor blimp system. The hardware is present on the payload carrier, but no ROS messages were created for the barometer.

c) Success: The BMP280 allows the outdoor blimp system to be accurate to 1 m. As well, since the BMP280 uses an I2C interface, it was simple to integrate into the system on the I2C bus of the Raspberry Pi.

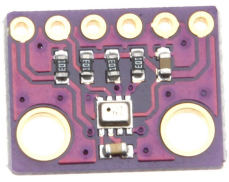

Fig. 60. Diymore BMP280 Barometer Unit.

4) Software Packages:

a) Approaches: The two IMU systems were integrated into the ROS system. Their nodes were published as the following nodes /Outdoor_Blimp/imu/pose, /Outdoor_Blimp/imu/twist, and /Outdoor_Blimp/imu/orientation. One set of messages were published for the IMU mounted on the payload carrier, and one set of messages were published for the IMU mounted on the camera system. The

GPS published /Outdoor_Blimp/gps/general_info and /Outdoor_Blimp/gps/velocity. The barometer was not integrated into the software of the system, and so it did not publish any nodes.

b) Failures: The main failures of the sensors were the refresh rate. The refresh rate of the IMU systems needed to be reduced to 10 Hertz to allow for the system to properly obtain and filter the values. If the refresh rate was increased, invalid data values would be published from the system because the registers of the IMU would not update with the correct values.

c) Success: The success of the sensor system was that valid quaternion and Euler angles were published from the IMU and valid GPS values were published from the GPS. Another success of the system was that it allows for future expansion of the system over the I2C bus. If another sensor needs to be integrated, the I2C bus is available to expand to another device.

G. Supply Regulation and Electrical Circuitry Modules

1) PCB:

a) Approaches: With the first version of the outdoor blimp, the system used jumper cables to connect every peripheral to the Raspberry Pi 3. The issue with this system was that the wiring became tedious and difficult to manage. In order to simplify the system, a custom PCB was designed and manufactured in order to integrate all of the individual components of the system.

b) Failures: The system designed the PCB though multiple iterations and then prototyped the PCB on the Othermill Pro. The issue with the Othermill Pro Milling machine is that the PCB copper is exposed, and so, human error during soldering can cause shorts in the circuitry which can damage components if not carefully soldered and checked. Therefore, the design was manufactured by a professional PCB fabricator to reduce the potential issues of milling our own PCB. An issue with the professional PCB was a human error of placing the PCB incorrectly on the Raspberry Pi. If the PCB is placed correctly on the microcontroller, then the power from the PCB could fry GPIO pins.

c) Success: The PCB simplified the entirety of the payload carrier while reducing the weight of the system. As well, modules could easily be added and separated from the system without the need to attach and re-attach jumper wires. The PCB also allowed the use of a voltage regulator circuit, reducing the weight of the system by 25 grams and creating a reliable power system for the blimp.

2) Supply Regulation:

a) Approaches: The first iteration of the blimp utilized a 1000 mAh power pack to supply voltage to the microcontroller, sensors, and gimbal system. Two single cell LiPo batteries were used to power the motor systems. The second iteration of the power system used a proper voltage regulator circuit located on the PCB. The voltage regulator on the circuit was a LM1085-ADJ adjustable voltage, three Amperes voltage regulator.

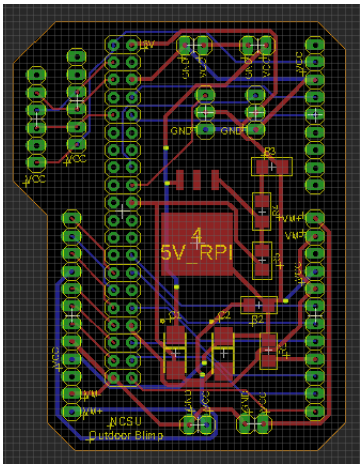

Fig. 61. Layout of the custom PCB design.

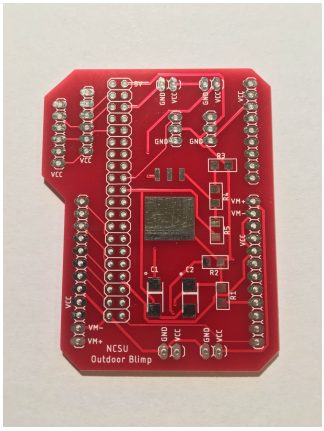

Fig. 62. Image of the custom manufactured design.

b) Failures: In the first iteration of the power circuit, the power pack could not supply enough current to operate both servo motors on the pan-and-tilt system. The capacity of the power pack was not enough to move both servo motors at the same time. Therefore, the second iteration of the power system increased the capacity of the system as well as the mAh supplied to the system. An issue with the final power system was that unbalanced LiPo batteries could drop the supplied voltage below 5 volts, causing the microcontroller to hit a brownout condition. Using two batteries with the same voltage levels would reduce the chance that one battery would drain before the other. Another issue with the system was heat. In the next iteration of the PCB, a heat sink on the bottom side should be included to help dissipate the heat from the voltage regulator.

c) Success: The final iteration of the system supplied a constant 5 volts to the Raspberry Pi for a maximum operation time of 45 minutes. The motor system can operate at 30 minutes if the motors are run at 30% of full power throughout the flight. Another success of the system was that it allowed for adjustable voltage regulation. By changing the resistor values in the circuitry, the voltage could be changed to supply a

different voltage level.

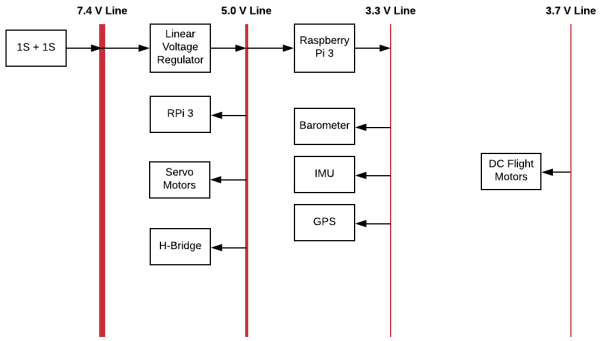

Fig. 63. Block diagram of the outdoor blimp power system.

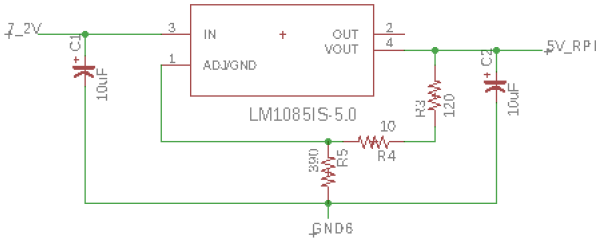

Fig. 64. Schematic of the Power Regulator Circuit.

H. Motor Propeller system

1) Motor controller:

a) Approaches: The outdoor blimp team used the Adafruit H-Bridge module, powered by a Li-Po battery and controlled by the GPIO and PWM pins on the Raspberry Pi. The initial iteration at the first demo included two H-Bridge units to control two motor pairs: one for yaw and one for forward/backward translation. However, once it was determined that the blimp would not control its own translation, one H-Bridge was removed.

b) Success: The H-Bridges functions as expected, and the H-Bridge system was able to be controlled by the GPIO and PWM pins on the Raspberry Pi 3 in order to control the motors. The H-Bridges were able to provide bi-directional control for in-place yaw control.

2) Motor and propellers:

a) Approaches: The outdoor blimp system used two Hubsan-X4 motors and propellers to control the motion of the blimp. At the beginning of the project, the motors were tested to determine thrust capability and ideal power output. The peak operating voltage and current was confirmed to be 3.7V and 1.5A, which the outdoor blimp team used to determine battery capacity.

b) Success: The motors and propellers functioned as expected. Indoors, the motor controllers and motors were able to control the translation and yaw of the blimp. Outdoors, the system was able to control yaw while tethered due to the effects of wind on the system.

I. Control and Navigation Module

1) Velocity control:

a) Approaches: The outdoor blimp team used the motor and propeller system to control the three-axis motion of the blimp. A ROS node was created to provide open-loop directional input to the blimp, which was converted into control signals from the GPIO pins on the Raspberry Pi to the H-Bridge.

b) Failures: Translational control was removed from the system as it was determined that the blimp would stay closely tethered to the husky and only control its own yaw. In addition, closed-loop control was never implemented due to lack of velocity feedback from SLAM teams and from our own GPS. When testing indoors and near buildings, our GPS unit had difficulty obtaining GPS lock.

c) Success: In the first demo, the blimp was able to receive ROS commands and had full translational ability (left/right, up/down, forwards/backwards).

2) Yaw control:

a) Approaches: The outdoor blimp team used a PID controller with the two side motors on the payload carrier to control yaw. The motors had bi-directional input, allowing for smooth yaw correction. The yaw was provided by the BNO055 sensor at a frequency of 10 Hz, and output was provided to the motors at 10 Hz. A ROS node was created to control the yaw setpoint.

b) Success: The yaw was set through the web interface, and the blimp was able to respond and face at the appropriate setpoint. It was able to respond in a slight amount of wind while tethered to the Husky.

3) Software Package:

a) Approaches: The system subscribed to a setpoint node to determine its heading. The setpoint node was set by the user, but could be integrated into an autonomous functionality in the future.

b) Failures: The failure of this system was the logical error seen during demo II. The issue was that the error calculation was incorrect, causing the PID control loop to be unstable.

c) Success: When the PID error issue was fixed, the control system allowed for the system to turn and hold a specific heading.

J. System Failures

a) Path Planning: The primary failure from the initial plan was the blimp's incapability to follow a specified plan given by the Husky. This was mainly caused by a lack of control outdoors in the wind. The motors did not have enough power to compensate for more than a slight amount of wind. Due to this, outdoor blimp team had to add multiple tether points, restricting the translational ability of the blimp while maintaining yaw control.

b) Outdoor to Indoor Transition: The initial stretch goal for outdoor blimp was creating a system for transitioning the outdoor blimp to indoor. Due to time constraints, outdoor blimp team was not able to work on this.

K. Future Work

There are several ways to improve the current design. Some of the hardware improvements are as follows:

- Increasing the envelope size to allow for more lift.
- Creating control systems to stabilize the system in the wind.
- Using light-weight servo motors for the gimbal system.
- Redesign the method the payload and motor stands are attached to the blimp envelope.

A large potential change to the system would be to have the outdoor system be a drone. The effects of wind on the blimp were extreme even in windspeeds as little as 5 miles per hour. A drone would allow for a more stable platform in outdoor conditions since motors are more powerful and can hold positions in space.

Some of the software improvements are below:

- Design and implement autonomous movement and path planning.
- Integrate altitude control, barometer, and GPS back into the system.
- Allow for manual control using a controller.

L. Collaboration

For the collaboration with SLAM and context awareness teams, the outdoor blimp publishes all the necessary data in ROS such as the compressed and raw camera images in the required resolution and at almost real-time and IMU data. In addition to publishing the data, the blimp can rotate to the given azimuth using the PID control system and point to an exact direction using camera and IMU. The requirements for the blimp nodes were given by the context awareness and SLAM teams.

REFERENCES

[1] Mur-Artal, Raul, and Juan D. Tards. "Orb-slam2: An open-source slam system for monocular, stereo, and rgb-d cameras." IEEE Transactions on Robotics 33.5 (2017): 1255-1262.

[2] Mur-Artal, Raul, Jose Maria Martinez Montiel, and Juan D. Tardos. "ORB-SLAM: a versatile and accurate monocular SLAM system." IEEE Transactions on Robotics 31.5 (2015): 1147-1163.

[3] Qin, Tong, Peiliang Li, and Shaojie Shen. "Vins-mono: A robust and versatile monocular visual-inertial state estimator." arXiv preprint arXiv:1708.03852 (2017).