Final Report - Part I: System Overview

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

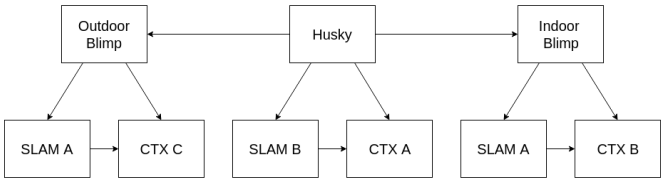

Fig. 1. Sys Overview

I. Introduction

This course aimed for the development of an autonomous mobile robotics system intended for applications such as autonomous management of a construction site. The system consisted of two mobile robots, the Husky, a ground vehicle, and an aerial micro blimp. The aerial system development was further divided into two parts depending on the environment within which it was supposed to work. The Indoor blimp was intended to work inside buildings or facilities, while the outdoor blimp was meant to navigate an exterior environment. The key aspects for the development of autonomous navigation are efficient path planning, localization of the robot, and mapping of the surrounding environment using visual and non-visual sensors. With these aspects in mind the development tasks were separated into three parts: Hardware Development, Simultaneous Localization and Mapping, and Context Awareness. Different teams were assigned to work on a specific aspect and to collaborate to achieve the goal of autonomous navigation as shown in Figure. For the Husky, the teams were Husky Hardware, SLAM-B and Context Awareness-A. For the indoor aerial blimp the teams were the Indoor blimp hardware team, SLAM-A team, and Context Awareness-B. For the outdoor aerial blimp the teams were Outdoor blimp hardware, SLAM-A, and Context Awareness-C. In this report we present an overview of the system which provides a brief summary about various components developed by individual team towards the achievement of the final goal.

II. Husky Hardware

The Husky hardware team worked on the Husky platform. The primary task of the team was to enable the husky to navigate autonomously using information provided by sensors, SLAM and context awareness.

A. System Overview

Husky is a medium sized robotic development platform. The main objective of this robot is to be able to navigate autonomously in a construction area with the help of Stereo camera, LIDAR, GPS, IMUs and Blimp. The Husky uses ROS packages to perform various computations on the processors and share data between various components of the system.

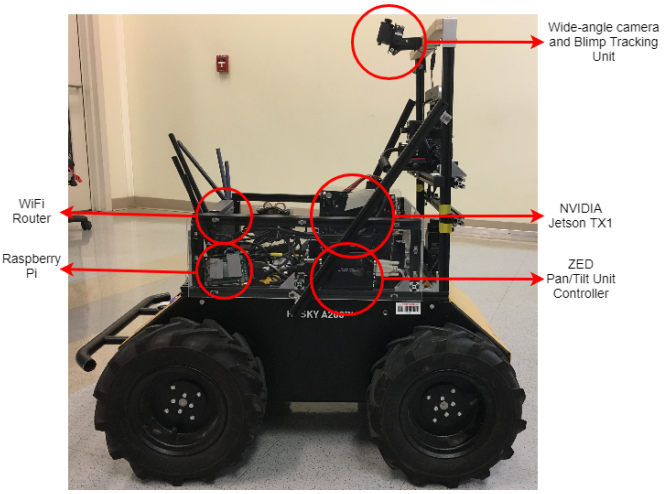

Fig. 2. Side view of Husky

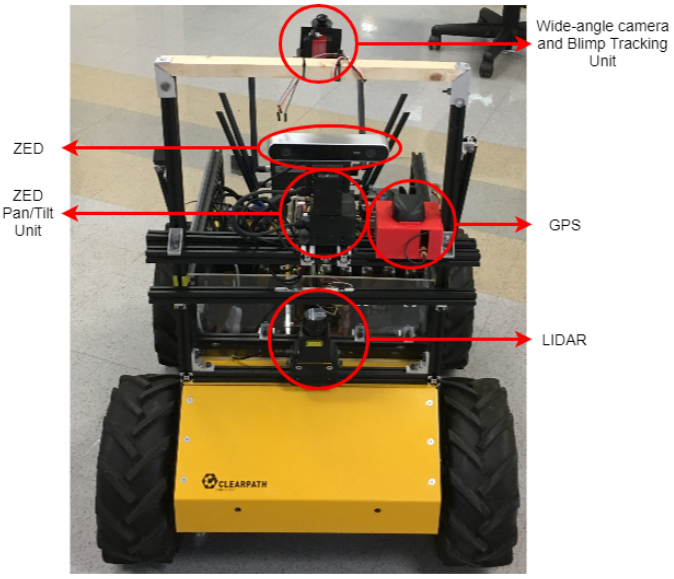

Fig. 3. Front view of Husky

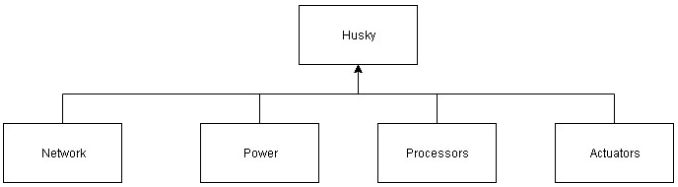

Fig. 4. Husky Block Diagram

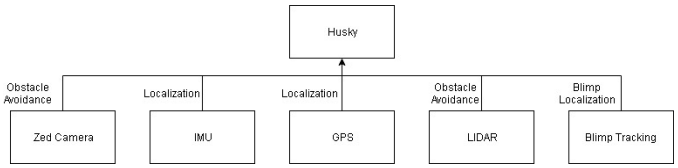

Fig. 5. Husky Sensors Diagram

B. Hardware design

The hardware platform was inherited from Spring 2017 students. The older hardware designs were revisited. The hardware consisted of the below modules:

1) Sensors
2) Actuators
3) Power
4) Network
5) Processors

1) Sensors: Sensors are provide input to the on-board Jetsons to run their computation. The function of each sensor is highlighted in Figure 5. The sensors are:

1) Zed camera
2) LIDAR
3) GPS
4) IMU
5) Blimp Tracking camera

2) Actuators: The actuators are to transport the husky from the current location to the fed location. The Husky moves at a max speed of 1 m/s. The Husky was also added with Pan and Tilt units to control the Zed camera as well as blimp-tracker.

3) Power: The Husky is powered by a 24 V, 20 Ah battery. The battery provides 192W of power and runs for an approximate 3 hours. The battery is used by the Husky for locomotion as well as to power the Jetsons, Router and Sensors. Hence, power from the battery is distributed at 24V, 12V and 5V for other devices.

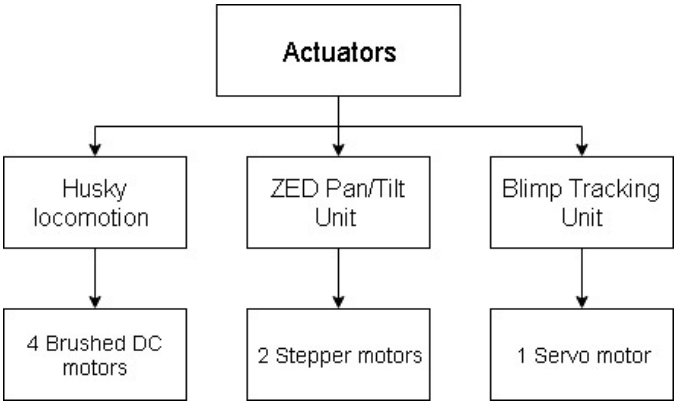

Fig. 6. Husky Actuators Diagram

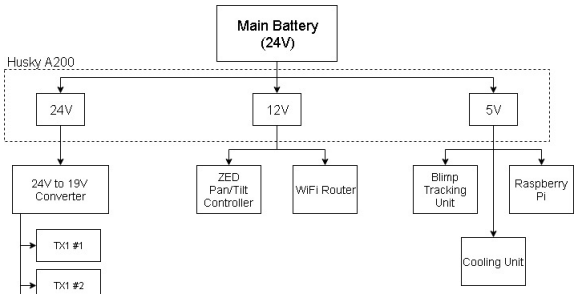

Fig. 7. Husky Power Distribution Diagram

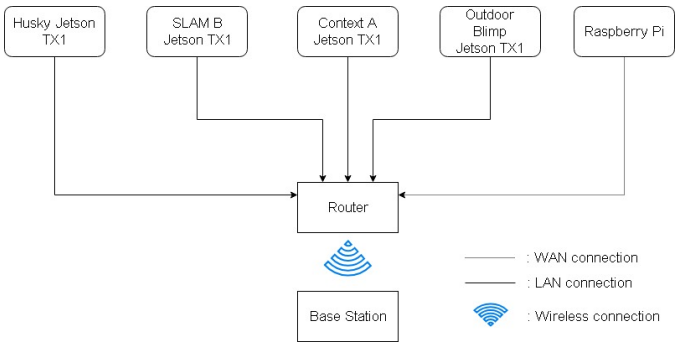

Fig. 8. Husky Network Diagram

4) Network: A router was installed on the Husky. The router would interface the Jetson TX1s and the raspberry pi as shown in the figure. The Raspberry Pi was connected to the NCSU Wifi which provided internet usage. The WAN connection from the Pi to the Router provided internet access to the Jetsons too. The Jetsons as well as the Pi were assigned static IP addresses for coherant communication. The Base station was connected to the router using the WIfi of the router.

5) Processors: The Husky relies on powerful computation to autonomously navigate. Hence, it consists of three Jetson Tx1s and one Raspberry Pi 3. The Jetsons perform various tasks such as segmentation, mapping, localization, blimp tracking and navigation in a given area. Each Jetson publishes data over the network using ROS. The Raspberry Pi controls the actuators and receives commands over TCP/IP from the Jetson.

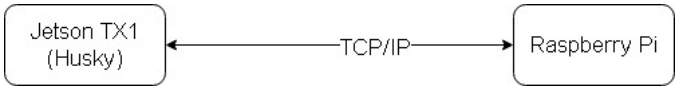

Fig. 9. Husky Processors Diagram

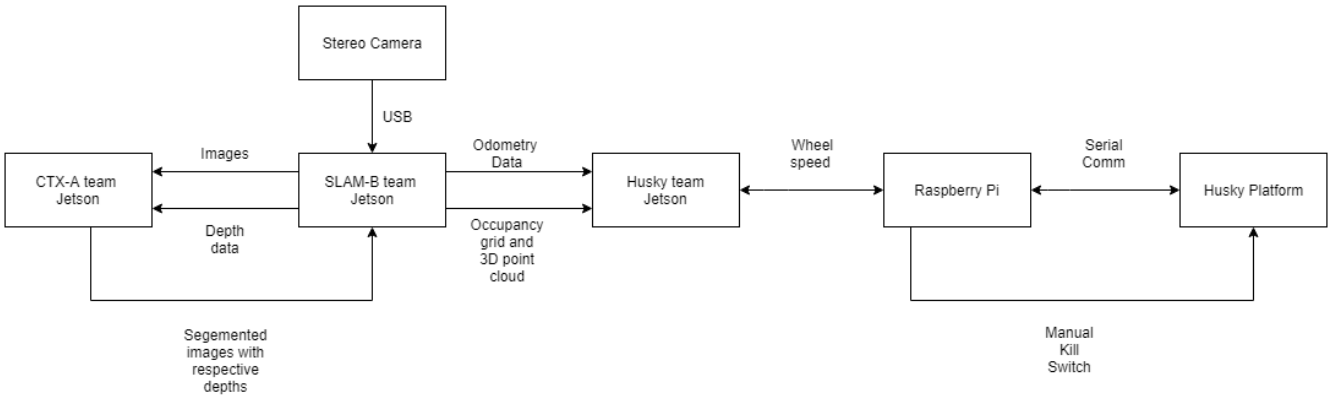

Fig. 10. Stereo vision-based general pipeline

III. Context A

A. System Overview

At the system level our scene segmentation framework has been part of Husky navigation demo (See Figure 10). The main objective of CTX-A is to help husky to navigate autonomously. This is achieved by semantic segmentation and depth map info. As husky was supposed to be navigated indoor as well as outdoor, we have trained our system with both types of images. We have also considered sunny and cloudy weather while training our network. SLAM-B team provides the occupancy map using the data published by the CTX-A team. Occupancy grid is used by husky for the localization and navigation purpose. For semantic segmentation ENet with Caffe deep learning network is used. All the inputs and outputs provided by CTX-A team are subscribed and published by ROS topics. In the following we will provide more detail about different stages of the pipeline namely, input, processing, and output.

In the input stage, we subscribed to the image feed (left camera of ZED) and depth images from the SLAM-B through ROS topic. For training of the ENet model we annotated 1000 images in three classes namely, ground, obstacles, and sky. Input for the left camera feed and corresponding annotated images are shown in Figure 11 and depth camera feed is shown in Figure 12

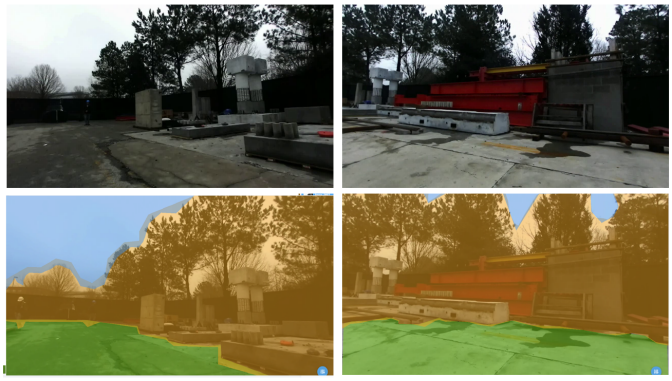

Fig. 11. top:left camera feed images, bottom:labeled images

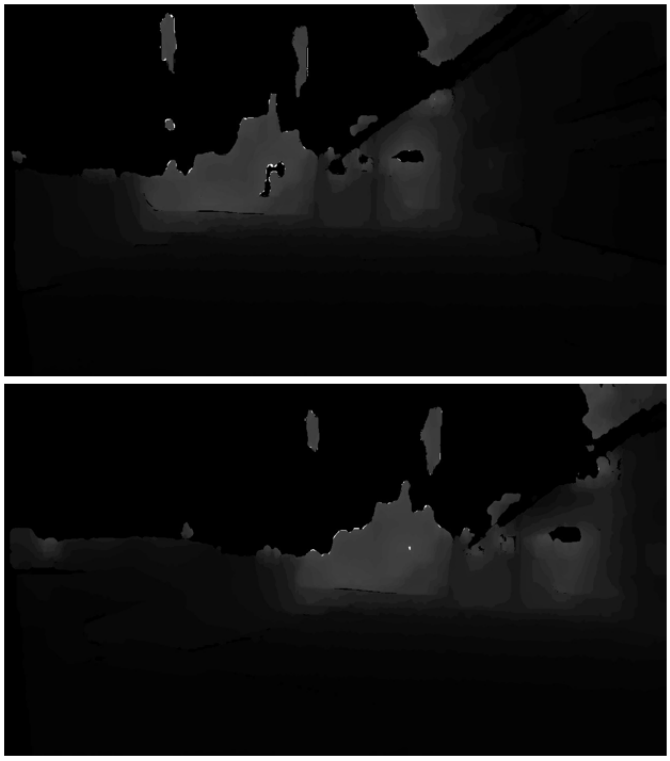

Fig. 12. Depth images

During the processing stage, we segmented the images into two classes for the final demo. Ground was indicated by the white pixels and rest by black pixels. Images were also published as image vector where a line separated ground from the obstacles. we synchronized the depth data with segmented images and published them. This was done because delay caused by latency of ROS and the network.

As the output we published three ROS topics consisting of segmented images, respective depth map images, and obstacle vectors. Figure 13 illustrates the left camera image on the top and the respective segmented images and the obstacles boundaries in the middle and bottom respectively.

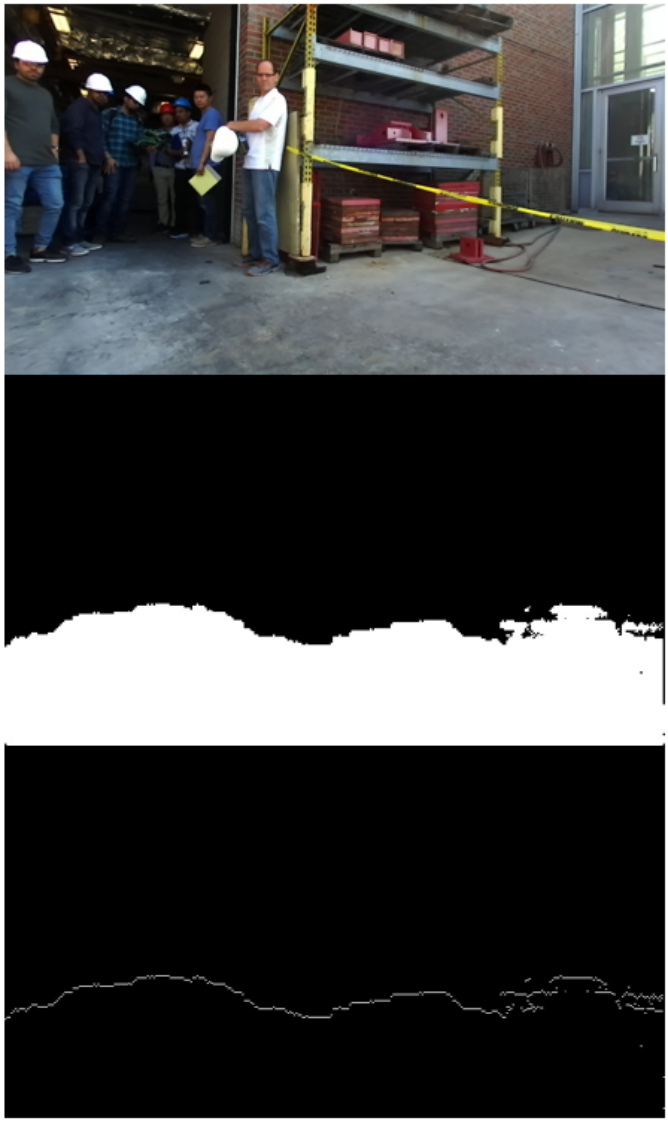

Fig. 13. top: input RGB image, middle: segmented result, bottom: obstacle line boundary

B. ROS topics

We subscribed to ROS image topics which were published by SLAM-B team. The SLAM-B team uses a ZED camera to generate all image topics. We subscribe to left camera image topic. The topic consists of a stream of rectified color images. We subscribe to this topic and use these images for semantic segmentation algorithm. We also subscribe to depth image topic published from the ZED camera. We synchronize left camera image stream and the depth images so that we can provide a two streams of time synchronized images using which the SLAM-B team can generate occupancy map.

We publish three image streams as three different ROS topics. The first topic that we publish consists of a stream segmented images. The images in this stream consist of two classes namely, ground and obstacle. Our segmentation output consists of three classes namely, sky, ground, and obstacle. But for the purpose of generating occupancy maps the SLAM-B team requires only two classes which are equivalent to two zones namely safe and non safe zone. So we post process our segmented images to merge the obstacle and sky class into one.

The second ROS topic that we publish consists of an image stream which consists images depicting an obstacle vector. We post process the semantic segmentation output images to depict a line vector as a boundary between the Two classes. This stream along with the depth stream can be used by the SLAM-B team to find the exact distance at which an obstacle is present.

The Third topic that we publish consists of a stream of depth images. The stream Consists of time synchronized depth images which match with each image given as output from the segmentation algorithm. Time synchronization is a necessary step so that the SLAM-B team receives corresponding segmented and depth images simultaneously.

IV. Context B

A. System Overview

The contextually aware autonomous robotic system using a blimp for the monitoring of construction site activity for the current project study is described below. A blimp or an inflated balloon was tied to the moveable hardware robot Husky and had a monocular camera attached to it that would stream a live camera feed at a constant frequency of 30 Hz. There were odometry sensors mounted on the blimp which would also stream odometry data to the SLAM module to perform real time localization. The SLAM module would also generate point cloud information from the visual data and send it to the context awareness module for projection of the segmented image onto it. The segmented image would then be used by the motion planning team to issue commands to the robotic system to help it navigate autonomously. Robotic Operating System was used to simplify the data exchange and synchronize communication between different modules in this project work. Figure 14 shows the System Design of the CTX module along with its interactions with other components in the system.

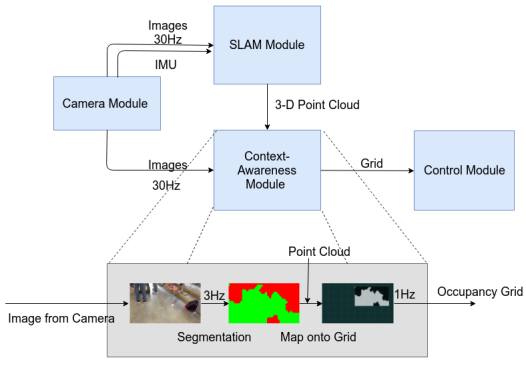

Fig. 14. An illustration of activity happening within a live construction site

To ensure that the context awareness module is able to identify the navigable and non-navigable areas clearly, we made sure that the data collected for the autonomous robot to work properly mimicked an actual construction site environment. This was made possible by the access to the Constructed Facilities Laboratory(CFL) at NCSU which provided a very good simulation environment for the whole system to be tested within. Figure 15 below illustrates a likely everyday scene at a construction site.

Fig. 15. An illustration of activity happening within a live construction site

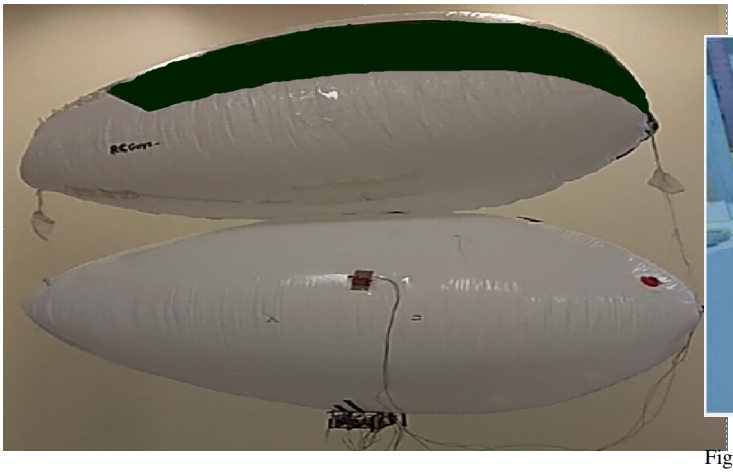

Fig. 16. Indoor Blimp setup with mounted camera for monitoring activity

B. Data Collection and Annotation

The image dataset was extracted from the videos recorded at a frame rate of 30 frames per seconds. The video was taken inside the CFL building at the construction site in the NCSU Centennial Campus. Since the blimp camera was tilted toward about 60 degrees along the horizon, the videos were shot and simulated at the same angle and at a height of about 2.5 meters from the ground level. The images were resized to 1280*720 before training the model as this was found to be the maximum resolution at which the blimp (shown in Figure 16) would be able to stream at a frequency of 30Hz. Since the blimp movement was not smooth, some motion-blurred images were also selected into the dataset so that the algorithmic model was able to learn this behavior. Two (sharp and blurred) sample images are shown in Figure 17, respectively.

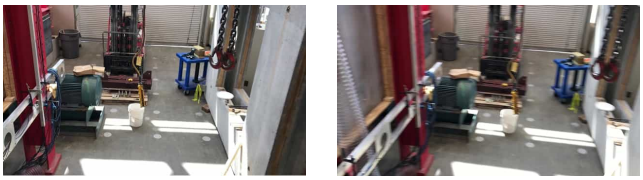

Fig. 17. Sample set of images collected for training the model

A number of tools were evaluated to annotate the images extracted from the video frames. Labelme developed by the CSAIL team at MIT was one of the first tool that we used to annotate but it lacked a simple and proper project interface for maintaining the labels and images. There were some others like Sloth, Annotorious and Fast Image Date Annotation Tool but we decided to supervise.ly which provides a rich web based ecosystem to cover not only all aspects of training data development but also to manage, annotate, validate and experiment with your data without coding.. There are two classes in the image segmentation task, ground (the navigable area for the autonomous robot), and the obstacles. A set of sample training images are shown in Figure IV-B. The green patch in the Figure 18 illustrates the navigable area in the construction site for the given view.

Fig. 18. A sample annotated image with the green patch depicting the navigable area for the autonomous robot

V. Context C

A. System Overview

The context-c team processes the data given by the outdoor blimp. This contains mainly two modules - object detection, semantic segmentation. The goal of ctx-c is to help outdoor blimp navigate by segmenting out the obstacles that are seen by blimp. Context-C should also enhance the understanding of surrounding environment for Husky to navigate.

B. Pipeline

The outdoor blimp publishes the images at the rate with which they can stream. In our case, it was 1 FPS. A subscriber node subscribes to the published images and then used it to segment out ground plane and obstacles. The segmentation is done into three classes - sky, ground, obstacle. More details

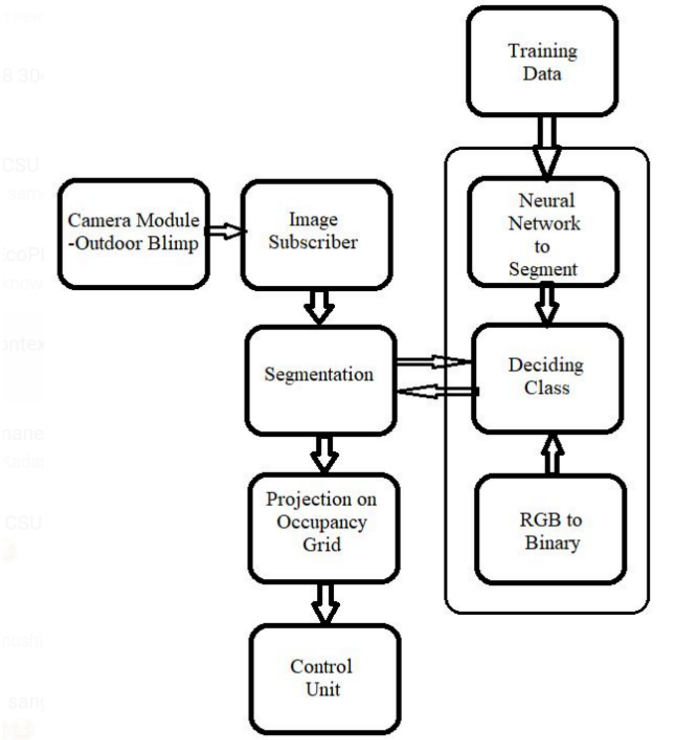

Fig. 19. Context C block diagram

about the classes are given in Part 2 of the report. Figure 19 shows the block diagram of the whole system. The segmented output is converted into binary by making only ground as free area to move. This output is visualized on rviz. This data is also published back for the control unit of husky/outdoor blimp to make navigation decisions. Sample data of normal activity work day is shown in figure 20 and 21. This is the outdoor environment of CFL site. We used this data to train our neural network to segment out the obstacles in the image. We used FCN Alexnet on Nvidia digits for segmentation, more details of which is in part 2, again.

Fig. 20. Sample data1

Fig. 21. Sample data2

Data annotations were done with the help of supervise.ly tool. We evaluated multiple other tools and finally decided on using this tool as the tool was very convenient to annotate. A screenshot of annotation tool is shown in figure 22. We annotated the images to classify into three classes, used the same three classes as outputs from the inference image to segment out obstacles. Further, obstacles were separated out from other two classes thus converting image to binary. This was projected on occupancy grid to help husky make navigation decisions.

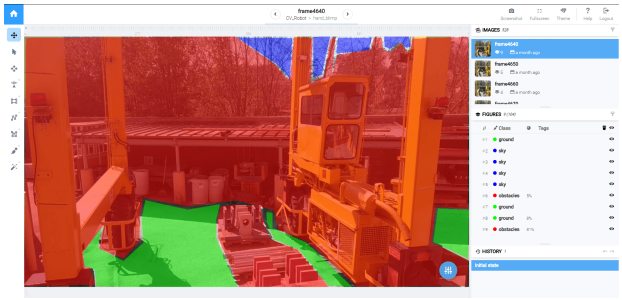

Fig. 22. Annotation tool

Hardware used -

- Host - Laptop with GTX - 1060 GPU
- Nvidia Jetson TX1 placed on husky robot

Ros topics subscribed to

- Outdoor-blimp/compressed

Ros topics published

- Topic Name: /ctx-c-segment/output/image-raw
- Message Type: Sensor Msgs - Image
- Topic Name: /map-ctxc
- Message Type: Occupancy map

VI. SLAM A

A. System Overview

VINS - Mono is a robust real-time algorithm for performing simultaneous localization and mapping. Making use of the visual and inertial information, it is able to make robust estimate of the current position of our agent, as well as provide a sparse representation of the environment in terms of a point

cloud. It uses an optimization-based sliding window formulation for providing high-accuracy visual-inertial odometry as visible in Figure 23. It features efficient IMU pre-integration with bias correction, automatic estimator initialization, online extrinsic calibration, failure detection and recovery, loop detection,global pose graph optimization, map merge, pose graph reuse, online temporal calibration, and rolling shutter support.

All the advantages come with a price. For monocular VINS, acceleration excitation is needed to make metric scale observable. This implies that monocular VINS estimators cannot start from a stationary condition, but rather launch from an unknown moving state. Hence during the demonstrations, the blimp had to be kept moving in a particular pattern to get robust initialization.

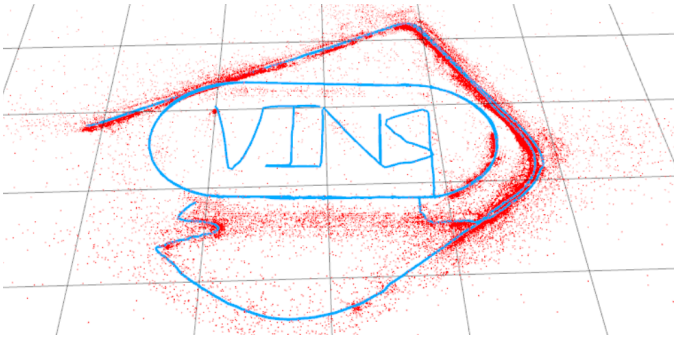

Fig. 23. VINS Estimator

1) Sensors Used: The bi-sensory input of visual and inertial features consist of a 30 Hz compressed input from a monocular Raspberry Pi Camera and 9 Degrees of Freedom(DoF) Inertial Measurement Unit (IMU - BNO005). This entire setup is present on the Blimp. Images are streamed at 410x308 resolution along with a quaternion of gyroscope measurements and linear acceleration along the three Cartesian Axes in real time.

At any given point of the time, SLAM localizes using the input from either Indoor Blimp or Outdoor Blimp. While both the blimps stream the odometry data at 120 Hz, the images streamed by both the blimps differed in frame rate. The frame rate provided by indoor blimp is 30 frames per sec and the frame rate provided by outdoor blimp is 5 frames per sec.

The system is setup such that the incoming sensory signals are time synchronized to a tolerance of below 30 ms. This is important as the time skew between the camera and the IMU can have a drastic impact on the pose prediction of the camera. As discussed above, the synchronization step was essential for indoor blimp. We are able to obtain results without synchronization for outdoor blimp as only 5 frames were sent per second. The algorithm processes the image to find key points which are extracted using a Harris corner detector to detect new key-points in every new image. Previously seen features are tracked using a KLT optical flow algorithm which tracks the intensity changes of a pixel and its neighborhood between frames. These points are used to create the history point-cloud which is saved relative to the camera coordinate system.

The entire system works on the ROS pipeline which provides a robust and efficient means of transporting sensor data from multiple sources. Figure 25 shows the entire data pipeline.

B. VINS Algorithm

The pipeline of monocular visual-inertial state estimator (VINS-Mono) is shown in Figure 24. Once the system initialized, the algorithm starts with measurement preprocessing, where features are extracted and tracked, and IMU measurements between two consecutive frames are pre-integrated. The algorithm gets all necessary values, including pose, velocity, gravity, vector, gyroscope bias, and 3D feature location from the initialization procedure. Visual Inertial Odometery(VIO) requires nonlinear optimization. Pre-integrated IMU measurements, feature observations, and re-detected features from loop closure are fused in the Local Visual-Inertial Odometry with Relocalization module. Finally, the pose graph optimization module takes in geometrically verified relocalization results, and perform global optimization to eliminate drift. The VIO, relocalization, and pose graph optimization modules run concurrently in a multi-thread setting. Each module has different running rates and real-time guarantees to ensure reliable operation at all times [1].

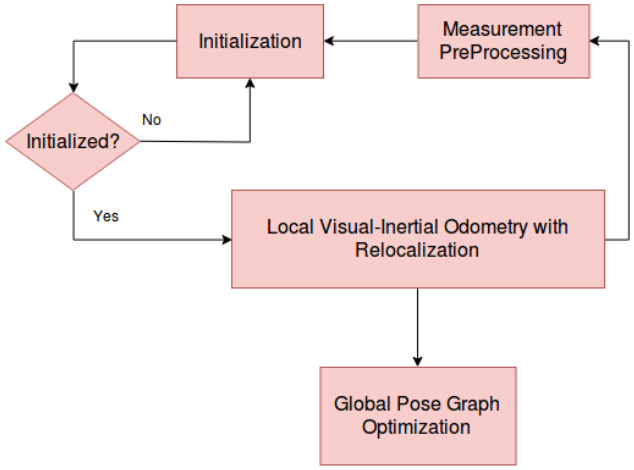

Fig. 24. VINS Estimator Block Diagram

C. Input

1) Visual Sensor: Raspberry Pi Camera Module:

Shutter type: Rolling Shutter
Image size: 410x308
Frame rate: 30 fps

2) Inertial Sensor: The BNO055 is a 9 DOF sensor consisting of an accelerometer, gyroscope and magnetometer. Powered with a high speed ARM Cortex-M0 based processor, the sensor is capable of giving out quaternions and Euler angles of its current pose.

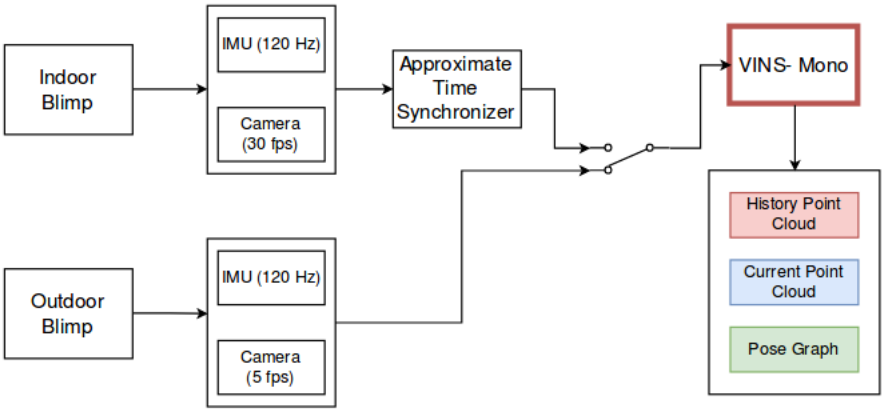

Fig. 25. System Overview of Monocular SLAM

D. Preprocessing

The data from the IMU sensor and the camera feed are usually not synchronized owing to the traffic on the network on the ROS pipeline. So we used the ApproximateTime package from the message _filter library to match the messages coming from the two sensors. This policy can match messages even if they have different time stamps. The output only depends on the time stamps, and not on the arrival time of messages. It assumes that the messages arrive in order on each topic and thus, Approximate Time can be used on messages that have suffered arbitrary networking or processing delays.

E. ROS Pipeline

Figure 26 explains the ROS pipeline adopted. VINS- Mono algorithm subscribed to image and IMU topics of Indoor and Outdoor Blimp. The trajectory and the sparse point cloud was then published to channel for context awareness teams. Exact topic names are depicted in Figure 26.

F. Output

The output provided by the Monocular SLAM consists of a sparse pointcloud, an estimate of the trajectory in the form of a pose graph and the realtime localization in term of 3-D coordinates.

To have the camera and the IMU sensor on the same coordinate axes, it is imperative to have a transformation between the axes of these two sensors. Hence, to achieve this, a Rotational and Translational transformation is published under the /tf topic.

VII. SLAM B

The SLAM B team (Simultaneous Localization And Mapping team B) was given the task of creating a map of the environment while concurrently localizing a robot within that map. This generated map is a 2D or 3D representation of the environment which is captured using a variety of sensors. Our team developed the SLAM algorithms for use with the HUSKY and as such left the development of the SLAM algorithms for the blimps to our colleagues in SLAM A.

A. System Overview

Our system uses both internal and external sensors to our team's JETSON TX1. The external sensors are transmitted to our JETSON as ROS topics by the other teams via a wireless router on the Husky. More specifically, the ZED Stereo Camera is connected to our board, while the LIDAR, the IMU and the Husky's odometer are broadcast as ROS topics from Team HUSKY's JETSON. Once we obtain the necessary ROS topics that the Husky team is publishing, we combine them with the ZED Camera data to build the maps and localize the Husky within the environment. Once we have created the map, we publish all the data from the sensors and all the maps as ROS topics, and we send them through the network to make them available for other teams to use. An example use case of this transmitted data would be the raw images sent to the context team for segmentation. The general structure of our module is shown in Figure 27.

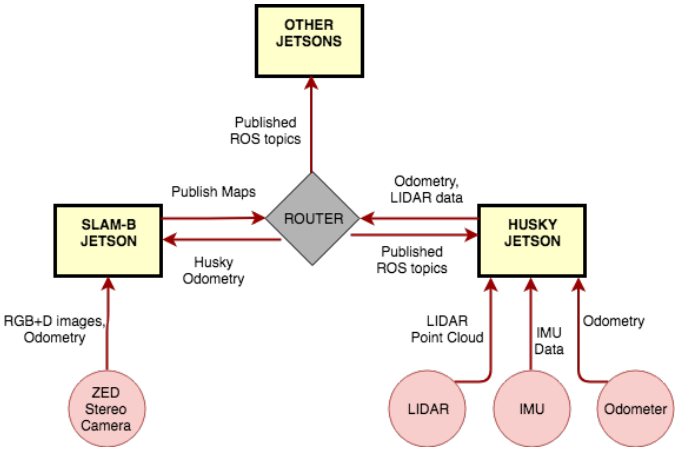

Fig. 27. Overview of the Architecture of SLAM B

B. Sensors

Our team used the ZED stereo camera as well as odometry and IMU information for our Stereo SLAM implementation. Future plans include the addition of LIDAR information so

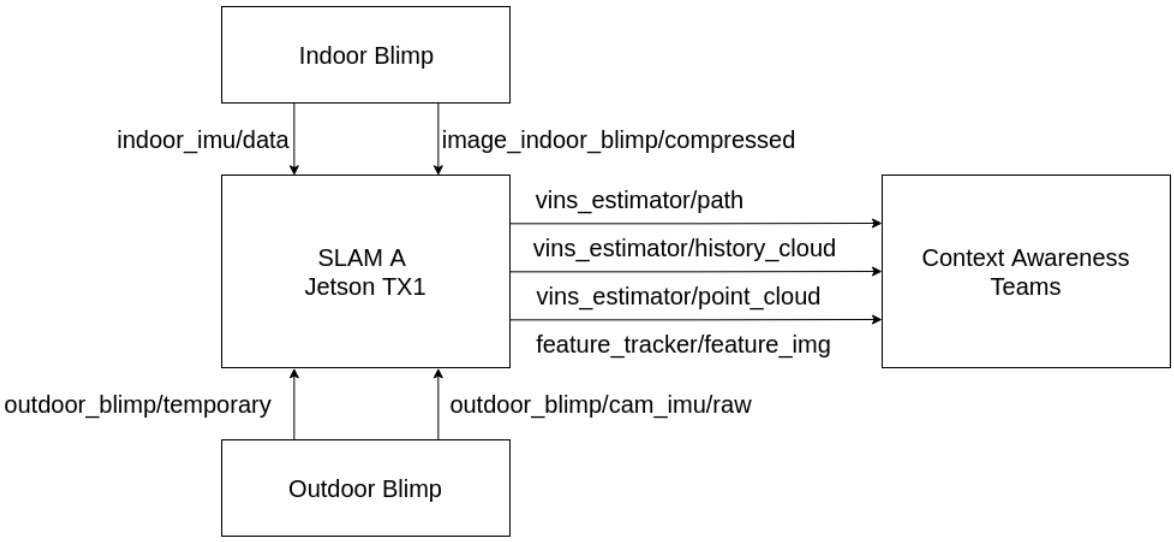

Fig. 26. ROS Pipeline of Monocular SLAM

information about this sensor is also included.

1) ZED Stereo Camera: The Stereo Camera is the primary sensor used to build the environment map. An image of the camera can be seen in Figure 28. It provides two RGB images in both raw and compressed formats for both the left and the right cameras. It also provides a depth image, which consists of an image where the pixels go from white to black, where white indicates the pixel is close to the camera and black indicates the pixel is far from the camera. Depth values in the intermediate range are represented as various shades of gray where lighter shades indicate a depth closer to the camera than darker shades. The depth image corresponds to the RGB image provided by the left camera of the ZED. This means that if the depth image and the RGB image need to be used at the same time, the RGB images of the left camera are the ones which perfectly overlap with the grayscale depth images.

The ZED camera is also capable of providing odometry data. It uses visual features and how they change over time in order to localize the position of the camera with respect to its environment in a technique called six degree of freedom pose estimation (6-DoF). For this reason, it is recommended not to decrease the frame rate of the camera to less than 15 fps or else the 6-DoF algorithm begins to fail. The odometry consists of the position of the camera, in x, y, z coordinates, as well as a value ω which represents the angle in which the camera is pointing. The values are not very precise, and the quality of the resulting localization and map can vary, if some objects are too close or if the image lacks visual features (for example, when it sees just a wall that is painted in a single color). However, it is possible to get a moderately accurate map using this odometry data if the environment contains visual features and there are not many moving objects passing in front of the camera.

Fig. 28. ZED Stereo Camera

2) Inertial Measurement Unit (IMU): The Husky team processes the IMU data sent by the sensor integrated in the Husky, and combines the data with the odometer data. The result is precise odometry consisting again of a position x, y, z, as well as a value for the orientation, ω. More specifically, the IMU uses the forces that are applied to the Husky as a result of the movement in order to calculate the orientation of the Husky. Our team did not directly take part in the calculations or processing of the sensor information. Instead, that is done by the Husky team, however we utilize the result of those calculations.

3) Odometer: The Husky provides odometry data that we can use as an alternative to the ZED camera odometry. It does so by fusing sensor data from an encoder on its wheels tracking wheel movements and the IMU to generate odometry data. The odometer is used to find the position (x, y, z coordinates) of the Husky, while the IMU is used to find the orientation (ω, or the angle at which it is pointing). This odometry works the same way as the one provided by the camera, but we found the Husky odometry to be more precise and less prone to errors. So we use this odometry instead of the ZED camera odometry when building the map. As a result, we can build a more accurate map and better localize the Husky within the environment.

4) LIDAR: The LIDAR sensor uses lasers to provide a 2D occupancy grid indicating where it believes obstacles are located and where the environment is clear. Figure 29 shows the LIDAR sensor employed on the Husky. We tried connecting the LIDAR to our Jetson directly, but it was not recognized as a connected USB device so we could not use it when connected to our Jetson. Thus, the LIDAR was connected to the Husky team's Jetson and the sensor data published as a ROS topic. Although we were able to visualize instantaneous LIDAR sensor data in rviz we were not able to fully integrate the LIDAR into the current version of the SLAM algorithm. This was due to synchronization issues between the Jetsons which currently remain unresolved and because of these issues the LIDAR information was not employed in the final demo.

Fig. 29. LIDAR Sensor

C. Software Overview

We used several software packages to make the SLAM algorithm work inside the NVIDIA Jetson TX1 board and to transmit that data to other teams as ROS topics. We first flashed JetPack 3.1 into the Jetson board, which installs a version of Ubuntu, as well as CUDA and OpenCV. Then, we installed the ZED ROS Wrapper, together with all its dependencies. Finally, we installed ROS and RTAB-Map.

With this software installed, we could use RTAB-Map to create a map using the published topics provided by the sensors mentioned in the previous section. First, we created a script to start publishing the ZED camera data as ROS topics. Secondly, we modified the rtabmap.launch file and created a script to start mapping the environment using the camera data as well as the Husky odometry data. This mapping results in the creation of a 2D occupancy grid map, as well as a 3D point cloud. Any team could then subscribe to those topics and use them for anything they needed, such as navigation. Finally, to visualize the results of the SLAM algorithm we had a laptop with ROS and rviz installed which could subscribe to all the available topics (RGB images, occupancy grid map, point cloud, two different odometries,...).

The final functionality that we provide with our software is the ability to store and retrieve mapped environments. When we decide to stop mapping the environment, the resulting map will be stored in a database (creating a .db format file). This database will contain all the images and the history of localizations that were used during the mapping phase. If we can find where the robot is in the context of previous mapped environments we can avoid the need to remap the environment. This process relies on enabling rtab-map in localization mode rather than mapping mode. This mode attempts to find a loop closure by comparing the images the robot currently sees with the stored images in the database. If the current visual features closely match previous features then loop closure occurs and the robot's current mapping and position is superimposed within the stored environmental mapping. After testing this mode, we can say that even though it takes some time for the loop closure to occur, once it happens the new localization is very accurate, so it works very well. It should also be noted that other teams receive the superimposed mapping once the loop closure occurs so it will not affect their ability to navigate the Husky well.

VIII. Indoor Blimp

The indoor blimp was a new addition to the existing system consisting of Husky robot and provides an aerial perspective for the autonomous surveillance system. As the task involved the development of a completely new system, a multistage approach was undertaken. The first step in the development of an aerial blimp was to design a flying mechanism consisting of the blimp and a gondola to carry the required hardware. The payload capacity, deployment environment and flight dynamics were considered while designing the same. Secondly, the development of sensors and actuator system was undertaken, taking into consideration the electrical and software aspects of each. Final stage involved the development of a navigation system with the ability to navigate the blimp in the indoor construction site as desired. The tasks targeted by the indoor blimp team can be summarized as follows:

1. Estimating the weight of the payload, that includes the hardware components, to estimate the size of balloon.
2. Designing the CAD model and 3D printing the gondola of the blimp.
3. Designing the electrical circuitry for interfacing various sensors, actuators and power supply regulation system.
4. Pilot testing the soundness of the circuits for sensors and actuators.
5. Developing the control philosophy pipeline.
6. Software development that includes the software driver development and control philosophy coding.

A. System Overview

The system essentially was realized using a single board computer, raspberry pi. The sensors and the actuators were interfaced to this controller. The sensors facilitated the blimp to understand the surrounding environment and assist the operation of different modules essential for autonomous navigation. For instance, the inertial measurement system (IMU) along with the monocular vision camera provided the data essential for localization and mapping of the Blimp. The data from such sensors was integrated with SLAM system to facilitate localization and mapping. Alongside, obstacle detection system in blimp assisted in performing local control for avoiding collision with any surrounding object. The designed system also incorporates a module that regulates the voltage to ensure that the controller and the drivers are powered from a constant supply voltage. Finally, being an aerial vehicle, the system also has a motor-propeller module that forms the basis of navigation system of the blimp. All these modules together form the blimp system which is shown in Fig. 30. The payload carrier with all hardware components is shown below Fig. 31.

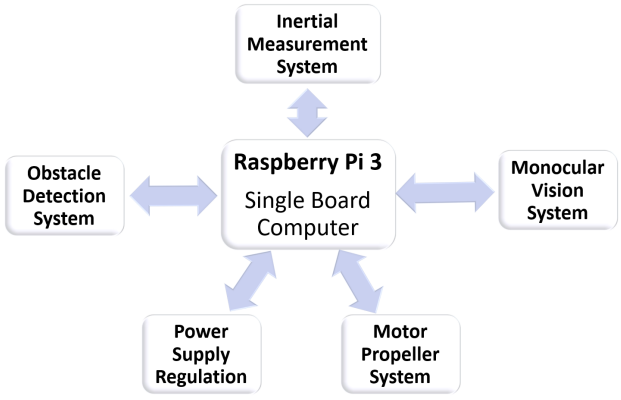

Fig. 30. Indoor blimp Sys Overview

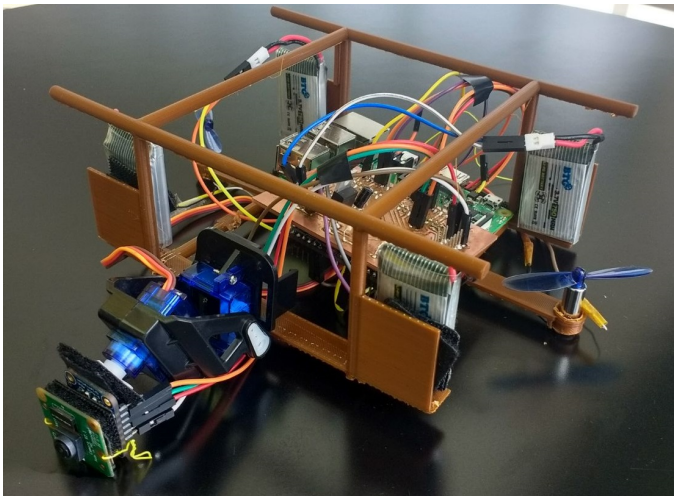

Fig. 31. Payload Carrier with all hardware components

B. System Modules

1) Aerial Blimp Structure: The aerial blimp structure basically consists of a gondola and an envelope. The envelope (Fig. 32) contains Helium, which being lighter than air gives upward thrust to the system . Geometrically, the envelope is ellipsoid in shape, with a diameter of 0.8 meter and length of 1.7 meter. The overall volume of the envelope is 0.5 cubic meter. The envelope, when completely filled with Helium, was designed to carry total weight of 280 grams (approximately). Over time, the envelope was prone to leakage and deflation resulting in reduced payload carrying capability(180 grams). So the blimp teams decided to stack up two balloons. The gondola was attached at the base of the ellipsoid envelope and was designed to carry the hardware components of the flying blimp and the motor propeller systems.

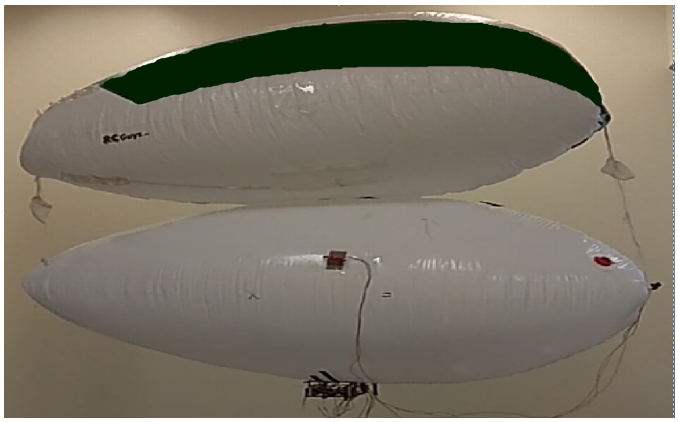

Fig. 32. Indoor blimp Envelope

2) Controller: The main criteria for selection of controller was that it should be able to run an operating system which could further facilitate the implementation of ROS environment. Additionally, it should also have the desired functionality for interfacing sensors and actuators and provide the required driving power. Finally, the controller had to be as light weight as possible. Given, such constraints we selected Raspberry pi Zero W which also has Wi-Fi that provides wire free networking capability. However, due to some constraints posed by the video processing task for live camera feed using the decided ROS package, the indoor blimp team eventually decided to use Raspberry pi 3.

3) Supply regulation system: The system was powered by two 750 milliampere batteries and it was observed that the voltage varied over time as the batteries were getting drained. But,it was essential that the system be supplied with a constant voltage for proper functioning of the controllers and the hardware components . This was achieved by incorporating a linear voltage regulator that received supply from the batteries. The output of the linear voltage regulator supplied a fixed voltage to the components within the system. LM1085 voltage regulator IC was used to get regulated voltage from the batteries.

4) PCB Design: A PCB was introduced into the blimp system to make it more robust, reliable and also to reduce its complexity. The PCB was fabricated in the lab and we tried different versions to get out the final model mentioned below. The PCB has voltage regulator on it which supplies constant voltage to Raspberry Pi and other peripheral devices as shown in the Fig.33. Other devices which incorporated on the PCB include motor drivers, ultrasonic sensors, connections for servo gimbal, and IMU. We also added few extra pins for power (both 5V and 3.3V) and ground in case the available pins on Raspberry Pi were not enough. EAGLE software was used to design the PCB.

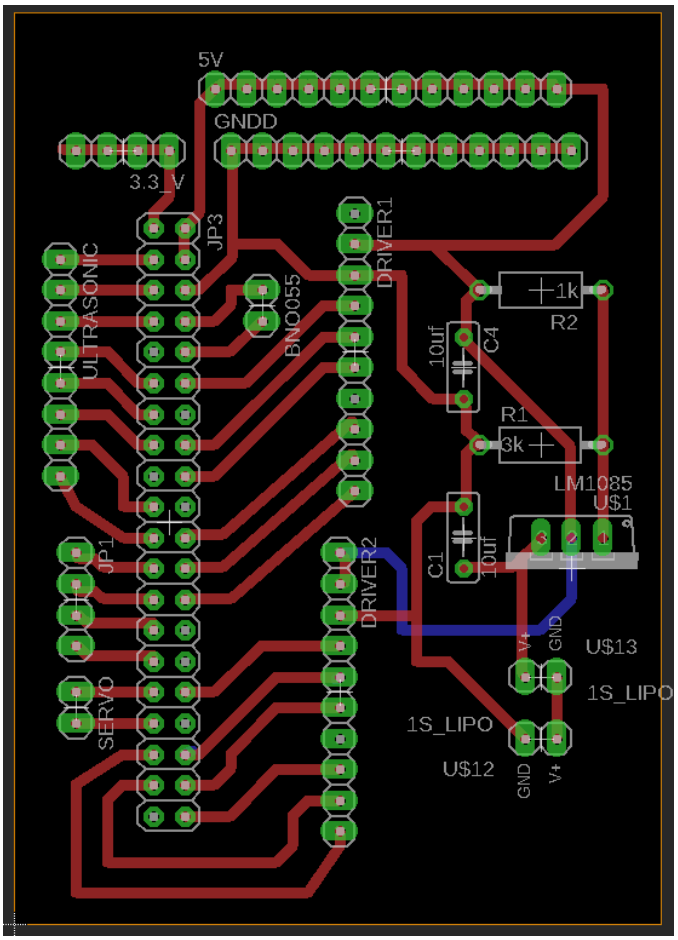

Fig. 33. Board design for final PCB

5) Motor Propeller System: The motor propeller system consisted of two motors for vertical movement and two motors for lateral motion. The lateral motion control system in some manner replicated the differential drive system of a ground robot. The vertical motion system provides upward and downward movement capability, while the lateral motion system facilitated forward, backward and left/right(yaw) turn motion. Being an aerial navigation system and considering the complex dynamics of the blimp system, pitch and roll movements were not incorporated.

6) Monocular Vision System: The monocular vision system was realized using a raspberry pi camera module, which comes along with the Raspberry pi. ROS packages were used to interface the camera and publish the desired images at a desired frame rate that assisted the SLAM and context awareness teams.

7) Inertial Measurement System: The Inertial Measurement System measures and reports the specific force, angular rate, and the magnetic field surrounding the unit using a combination of accelerometer,gyroscope and magnetometer. This helps in localization of the unit by providing the linear acceleration and orientation. This is used by the SLAM algorithm along with the monocular images for localization and mapping.

8) Obstacle Detection System: The obstacle detection system was realized using an ultrasonic sensor that measures distance by sending out a sound wave at a specific frequency and listening for that sound wave to bounce back. By recording the elapsed time between the sound wave being generated and the sound wave bouncing back, it calculates the distance between the sonar sensor and the object. Thus, whenever there was an obstacle close to the unit, the ultrasonic reading will be lower than the pre-defined threshold and this helps in stopping the unit from moving towards the obstacle and to redirect it to some other direction away from the obstacle.

C. Software Development

1) Robotic Operating system: A communication system is often one of the primary components in the implementation of a new robot application. ROS's built-in and well-tested messaging system saves the time by managing the details of communication between distributed nodes via the publish/subscribe mechanism. The interfacing between all the individual components are done using the libraries and tools available in ROS. This is very helpful in communicating with other modules that forms the autonomous navigation system. There are message definitions for geometric concepts like poses, transforms, and vectors for sensors like cameras and IMUs. By using these standard messages in the application, the code will inter-operate seamlessly with the rest of the ROS ecosystem, from development tools to libraries. Another useful feature of ROS is the availability of a visualization tool 'Rviz' which helps in visualizing all the published/subscribed data using a graphical user interface.Visualizing all of the data in the same application not only looks impressive, but also allows to quickly see what the robot sees, and identify problems such as sensor misalignments and robot model inaccuracies.

Different packages were created for publishing/subscribing data which helps in communicating with other modules. Figure 34 shows the interface between other modules.

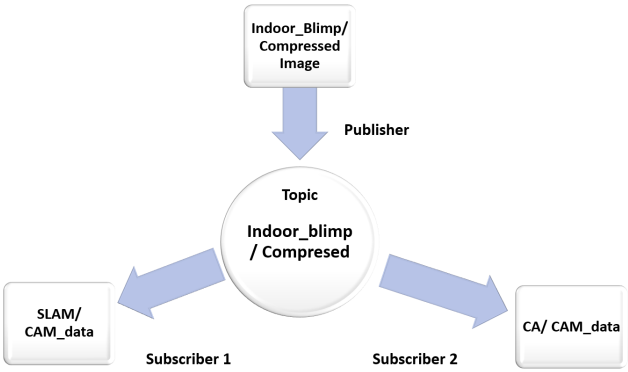

Fig. 34. Indoor blimp ROS Publisher/Subscriber

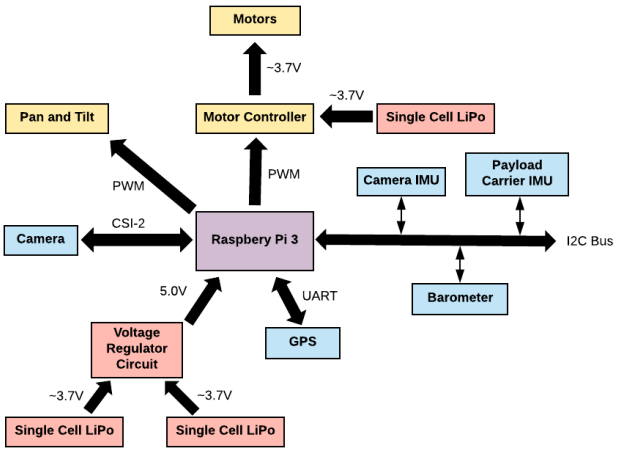

Fig. 35. Outdoor blimp system block diagram.

2) Nodes: The package '$raspicam\_node$' publishes the video data using the node '$CompressedImage$' with the topic name '$image\_indoor\_blimp\_compressed$'. The package '$bosch\_imu\_driver$' publishes the IMU data using the node 'IMU' with the topic names '$/indoor\_imu/data/$', '$/indoor\_imu/raw/$' and '$/imu/yaw$'. The package ultrasonic publishes the ultrasonic measurements using the node' $talker$' with the topic name '$ultrasonic$'.

3) Control System: An open-loop control is implemented for the forward-backward, up-down and left-right motion. In addition to this, a PID based control is implemented for the yaw motion which helps in accurate and responsive correction to a control function. The orientation data regarding the unit is obtained from the IMU is used for this PID control.

IX. Outdoor Blimp

A. Responsibilities

The outdoor blimp system was designed to aid the ground husky in outdoor operations and traversing areas that the husky cannot reach. The system allows an aerial view of the outdoor areas when the husky is navigating the construction area. The outdoor blimp uses a hardware sensor system in order to collect data on the area, motors and actuators in order to navigate the construction site, and a software system in order to coordinate with the husky, SLAM, and context awareness systems.

B. Hardware design

The hardware block diagram of the system is outlined in figure 35.

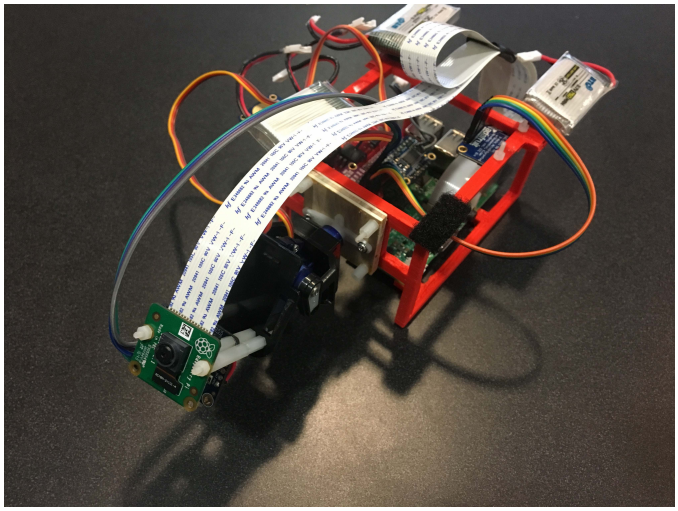

Fig. 36. Image of the outdoor blimp payload carrier with attached hardware components.

1) Processors: The on-board processor on the blimp uses a Raspberry Pi 3 in order to coordinate the sensors and control systems on the blimp. In addition to the Raspberry Pi 3, a Jetson TX1 is mounted on the Husky which has better performance and used for more complicated blimp processes. For example, since the Raspberry Pi 3 was not powerful enough to publish videos to all of the nodes on the ROS network, the Raspberry Pi 3 publishes the videos to the Jetson, and then, the Jetson republishes the videos to all of the nodes of the ROS network. Using this approach, the processing load reduces from the Raspberry Pi 3 and Jetson can handle the necessary tasks.

2) Sensors: The following sensors were used

a) Inertial Measurement Unit: Two BNO055 9DOF inertial measurement units were used on the system. These were chosen because of their light weight and their combination of an accurate, 10 Hz accelerometer, gyroscope, and magnetometer along with an onboard ARM Cortex processor for

sensor fusion, reducing computational load on the Raspberry Pi. One IMU was placed on the payload carrier to control the yaw of the blimp and pitch and roll of the pan-and-tilt mechanism. The second IMU was in axis with the camera to provide accurate information about camera heading to SLAM and CTX Teams.

b) GPS: An Adafruit GPS for absolute positioning was used on the outdoor blimp. Once the GPS unit obtained a lock, it provided latitude, longitude, and altitude data to the Raspberry Pi at a frequency of 10 Hz.

3) Motors and Actuators: Two 3.7 volt DC motors were utilized for the yaw movement of the system, and a pan-and-tilt servo system was implemented to control the roll and pitch of the camera system. A H-Bridge breakout board controlled the DC motors while the pan-and-tilt servos were controlled by the Raspberry Pi PWM signals.

4) Envelope: The envelope of the blimp has an ellipsoid shape. The diameter of the envelope is 0.8 meter and the length is 1.7 meters. The envelope's volume is about 0.5 cubic meter and it is able to lift about 280 grams as the vendor said, but because of deflation and leakage one single balloon can only lift around 180 grams. So, the outdoor blimp team decided to use two stacked balloon in order to reach enough capacity to lift payload carrier along with motors and other parts.

5) Power: The power of the system is given by three single-cell, Lithium Ion, 750-milliamp hour batteries which is regulated by a circuit on the printed circuit board. Two batteries power the Raspberry Pi, sensors, and pan-and-tilt servo motors. The batteries are connected in series and then regulated to 5V using a LM1085 voltage regulator. The LM1085 has a maximum current rating of 3 Amperes, and through a current calculation, the outdoor blimp team able to determine that our system would not draw more than the maximum current of the voltage regulator. The third battery is used solely for the motor system to propel and move the blimp. The block diagram for the power system is given in figure 37.

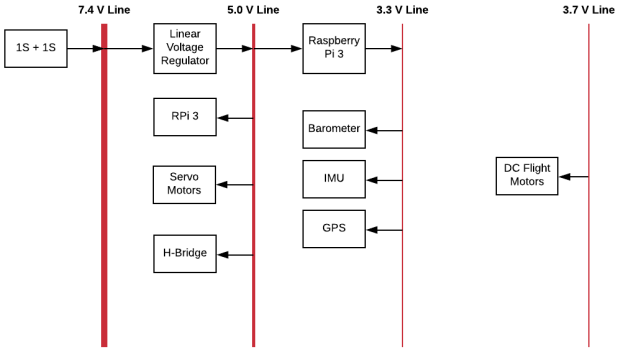

Fig. 37. Block Diagram for the Power System.

6) Printed Circuit Board: A custom printed circuit board (PCB) was fabricated in order to implement the voltage regulation system and to reduce the complexity of the system. The PCB was designed and prototyped in the lab, but in order to create a robust and reliable system, a PCB was manufactured to integrate the entire system cleanly. The PCB allows the sensor systems to integrate into the GPIO pins of the Raspberry Pi simply without having to use large and bulky jumper wires. The board design of the PCB is given in figure 38.

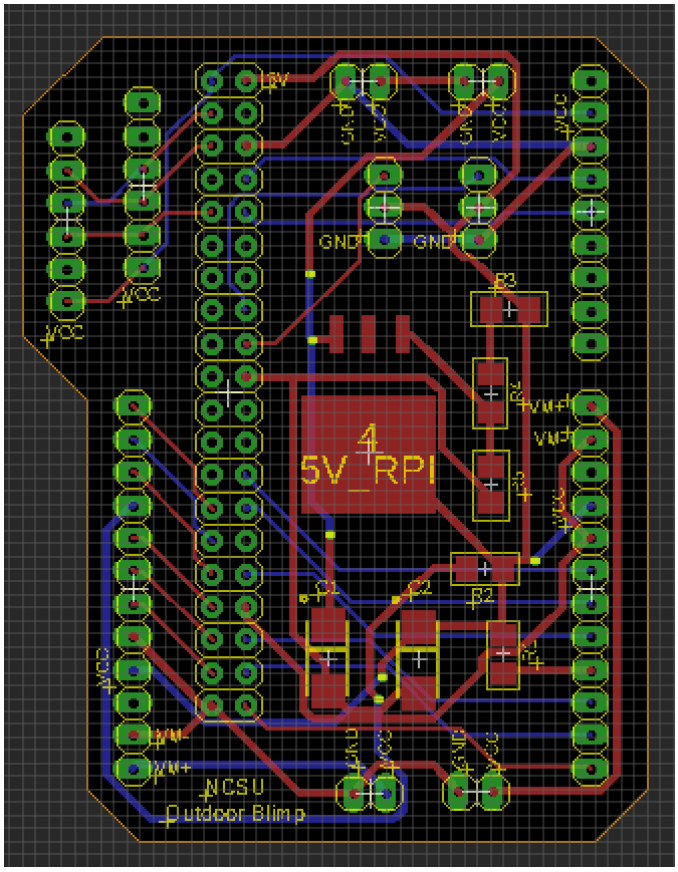

Fig. 38. Schematic for the Printed Circuit Board.

7) Network: The main communication modules are the built-in Wi-Fi module on the Raspberry Pi 3 and the built-in Wi-Fi module on the Jetson TX1. In addition to those units, a central router on the Husky is used in order to transmit the necessary data from the Raspberry Pi 3 to the Jetson TX on the Husky and other nodes on the network such as the slam and context awareness teams.

C. Software

1) Robot Operating System: The Robotic Operating System (ROS) was used in order to integrate all of the separate systems cleanly. ROS was installed onto the outdoor blimp's Raspberry Pi 3 and on the Jetson TX1. The Jetson runs the Roscore while the Raspberry Pi publishes topics to which other systems and teams can subscribe. This allows systems to easily communicate to each other using pre-determined messages.

2) Nodes: The following nodes were published from the Raspberry Pi at a frequency of 10 Hertz:

- Raspberry Pi Camera - raw_image.
- Raspberry Pi Camera - compressed_image
- Camera IMU - pose
- Camera IMU - orientation
- Camera IMU - twist
- Payload Carrier IMU - pose
- Payload Carrier IMU - orientation
- Payload Carrier IMU - twist
- GPS - gps_info
- GPS - gps_velocity

The image nodes were republished by the Jetson TX1 to offload the processing power to the more powerful Jetson TX1 processor. The Raspberry Pi subscribed to a setpoint in order to determine heading for the system.

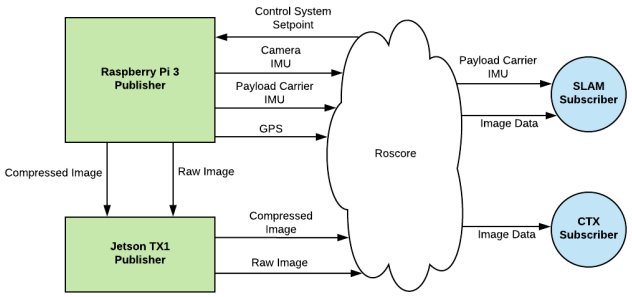

Fig. 39. Overview for the ROS publishers and subscribers to and from the outdoor blimp system.

3) Control System: A PID control system was implemented for the yaw control of the blimp system. The roll and pitch of the blimp were controlled by the pan-and-tilt actuator system. These motors and actuators were controlled by PWM signals from the Raspberry Pi 3.

X. Conclusion

Thus a brief system overview has been presented in this part of the report while the module details and the performance analysis is presented in part II and part III respectively.

References

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