16-662 Robot Autonomy: Final Report
Symbiotic Localization

I-Chen Jwo (ijwo), Pratibha Tripathi (pratibht), Pulkit Goyal (pulkitg),
Daniel Arnett (darnett) and Akshat Agarwal (akshata)

Project Mentors: Eugene Fang, Red Whittaker

May 2018

1 Introduction

Goal: The goal for our project was to develop a symbiotic localization system and deploy it on a pair of planetary rovers, enabling them to be localized with much greater accuracy than if they were operating alone.

Motivation: Rovers deployed in pairs can enable vastly superior technologies and scientific experimentation than a single rover, at the low cost of a small increment in rocket payload and development cost.

Planetary explorations to date have been solo acts. Rover pairs can cover more terrain, take more risks, accomplish more planned goals, localize more accurately, image one another and provide third-person views never before possible with solitary robots. The two rovers can be equipped with different scientific equipment for conducting a variety of tests and can enable mission control to take greater risk, explore more areas and potentially serve as backups for the other.

Rovers are traditionally localized using data from wheel encoders and inertial measurement units. The data is put through a filter (like the Extended Kalman Filter, or EKF), and used to derive an estimate of the location of the robot with respect to its starting location. This process of dead-reckoning is highly prone to accumulation of drift over time, leading to incorrect localization which is obviously undesirable. This problem is severely compounded in a long-term autonomous deployment situation, like that of planetary rovers - since they basically do not get any ground truth estimate of their position at any point during their mission. Localization accuracy for multiple rovers is improved over dead-reckoning by using each rover as a landmark for the other rovers. Since the errors due to drift accumulation in both rovers' localization are random, knowing the other rover's relative bearing acts as a sort of loop closure.

Problem Definition: Formally, the problem of multi-rover co-localization using relative observations can be defined as follows. Given a team of multiple rovers, each with the capability of estimating its own state (position and orientation) using proprioceptive sensors (encoders and inertial measurement units) and the ability to observe other rovers using exteroceptive sensors (VIVE tracker, camera, range sensor, etc.), determine the set of states for all rovers over time that maximize the probability of obtaining the set of exteroceptive observations.

Hardware Platform We worked on two planetary rovers (Auto Krawler 1 and 2, respectively) developed apriori in the Planetary Robotics Lab. Both are equipped with wheel encoders and an IMU. We retrofit them with the HTC Vive VR Tracker and 2 Base Stations, each. For onboard processing and communication, they have an onboard Odroid computer. The ROS architecture for controlling a single rover by giving it waypoints had also been provided to us.

Figure 1: Our Robots - Autokrawler 1 and 2 - Planetary Rovers

Approach Outline: We planned to improve localization of the 2 planetary rovers using their relative bearing measurements with an HTC Vive Tracker and a camera. This work was roughly divided into the following 3 stages:

- **HTC Vive Tracker Pipeline**: Using the HTC Vive Trackers and Base Stations to get the relative pose for the 2^{nd} rover
- **Vision Pipeline**: Tracking one rover with a pan-tilt camera on-board another, and using that to get a relative bearing for localization
- **System Integration**: Filtering these measurements with wheel odometry for robust collaborative localization.

2 Key Challenges

- Getting both Vive Trackers working robustly with fluid base station interchange
  - Each rover has been retrofit with 2 Base Stations, one facing forward and the other backward. Since the field of view of each base station is approx. 110°, there exists a blind zone in between, when the two rovers are roughly side-by-side. In this interval, when the Vive tracker is not able to track the rover, localization and odometry is done using wheel encoder and IMU data. When the rover enters the view of one base station of one base station after having exited the view of the other station a while back, it is important that there be a robust snap back of the localization to correct drift accumulated in the blind zone. This is quite complex to implement robustly.
- Integrating Intel RealSense with NVIDIA Jetson TK1
  - The kernel of Ubuntu 14.04 is not compatible with NVIDIA Jetson TK1, so we need to upgrade the kernel to at least 4.4.1. However, upgrading the kernel has to use a package called u-boot which is not available on the Jetson TK1. We have to flash the package on TK1 but the process didn't go well because the costume dtb file will be overwritten by the default shell execute the file.
- Combining two teb_local_planners
  - Each rover has its own teb_local_planner. In order to achieve leap-frogging, two teb_local_planners needed to be combined together and receive the start or stop command whenever the distance between two rovers is bigger than a certain threshold.

  - Two teb_local_planners have to have their own stage separately in order to make the planners receive different commands.
  - The distance could be calculated by the Vive Trackers and the base station, however, the orientation and the position of the Trackers couldn't be obtained directly, they needed to be calculated through the transformation and the rotation from the base stations mounted on the rovers.
- Integrating DJI Camera
  - The plan was to integrate the DJI Camera on the rover in order to track the other rover and send back the image if needed.
  - The DJI camera needs a certain matrix to connect to the Manifold, which is a costume NVIDIA Jetson TK1 processor.
- Detecting the other rover in camera feed and controlling the pan-tilt camera to keep it in sight
  - To further increase the robustness of localization and provide redundancy in localization methods (as backups in case of malfunction/unexpected contingencies), the rovers will be equipped with a pan-tilt camera. Each rover will learn to identify the position of the other rover in its camera feed, and use optical flow along with PID control to control the pitch, roll, and yaw of the camera gimbal such that the other rover is kept firmly in sight.
  - Using monocular visual localization methods, we can estimate the relative bearing of the other rover via the camera, and use that as another source of data for the localization filter to obtain an even more accurate estimate of the other rovers' position.
  - Integrating both relative pose measurements with wheel odometry
  - Constraining paths to maintain rover line-of-sight within 6m

3 System Architecture

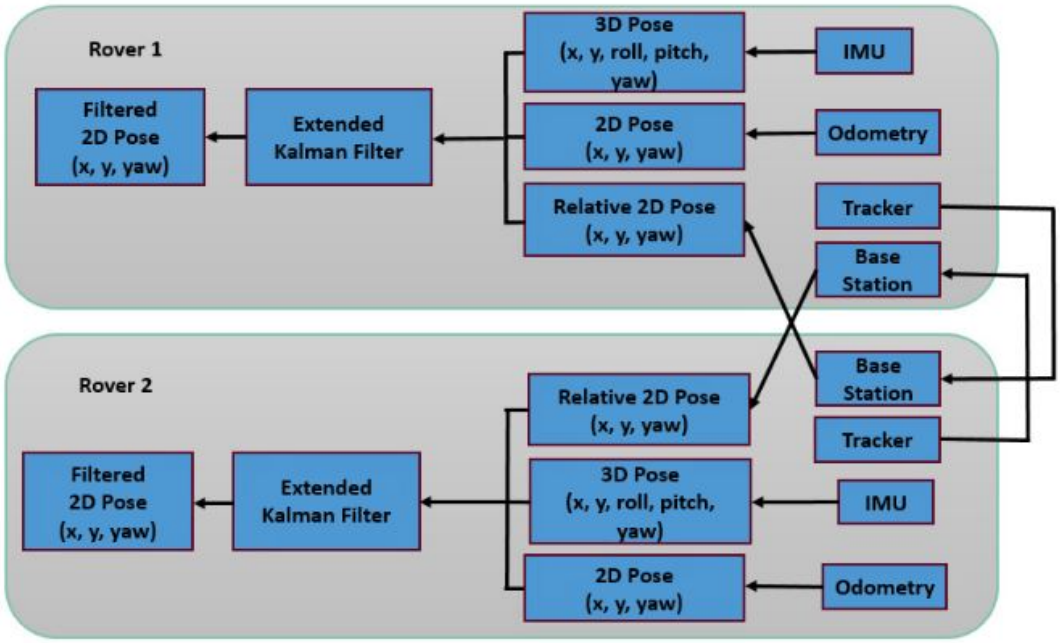

Figure 2: System architecture

A brief description of Components in the system follows.

- **Rovers**: These are serving as the robotic platform for our project.
  - Input: From all sensors.
  - Output: Execution of the path.

  For path planning in these rovers we are using the teb_local_planner package of the ROS.
- **HTC Vive Tracker** : We are using these to improve the localization of rovers with respect to each other.
  - Input: IR of the other rovers vive base
  - Output: Localizing/Pose correction w.r.t other robot's pose
- **Intel RealSense**: It is used to get the dense point cloud of the environment. Ultimately they will be used to segment the obstacle and place that in our local cost map. An occupancy grid obtained from this data is used for obstacle avoidance during path planning.
  - Input: Environment visual details
  - Output: Point cloud of environment finally converted to obstacles in map. For converting the point cloud to obstacles in occupancy grid, we are going to use Elevation mapping package from ROS. We had to make considerable amount of changes to read the data from appropriate topic and publish the data in right format on right topic. We have just completed the basic integration of this package. It'll require considerable amount of modification in the base package to make it useful for our use case.
- **Pan-tilt camera**: We were unable to complete this stretch goal, due to the fact that DJI likes to make all its products obsolete as soon as they release a newer version of the product, and most flight controllers, cameras and communication bridges are incompatible with each other.

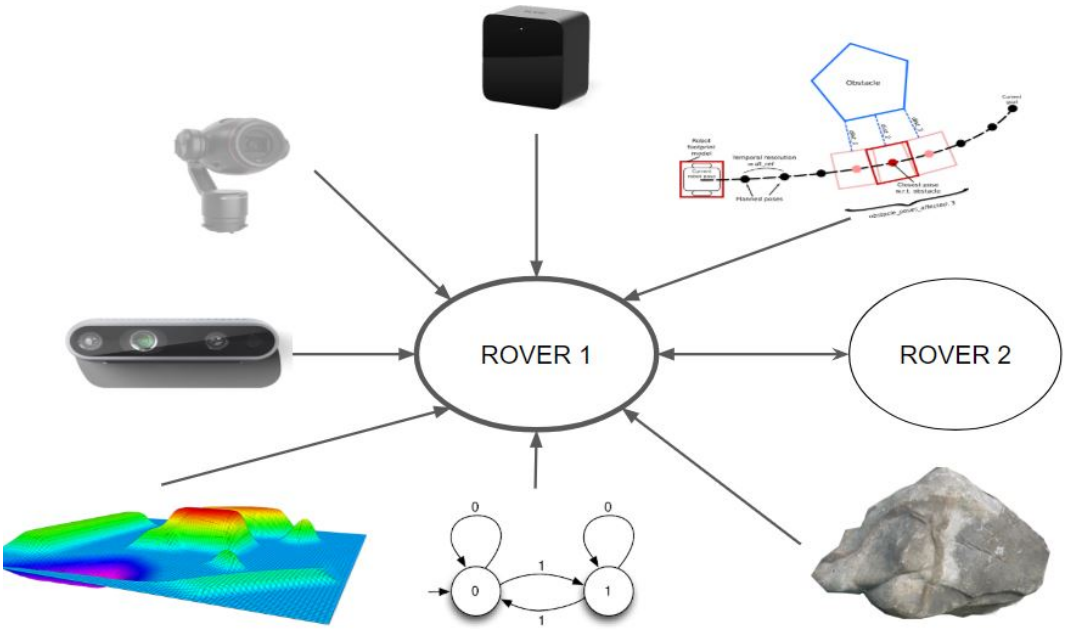

Figure 3: Overall Components in our system

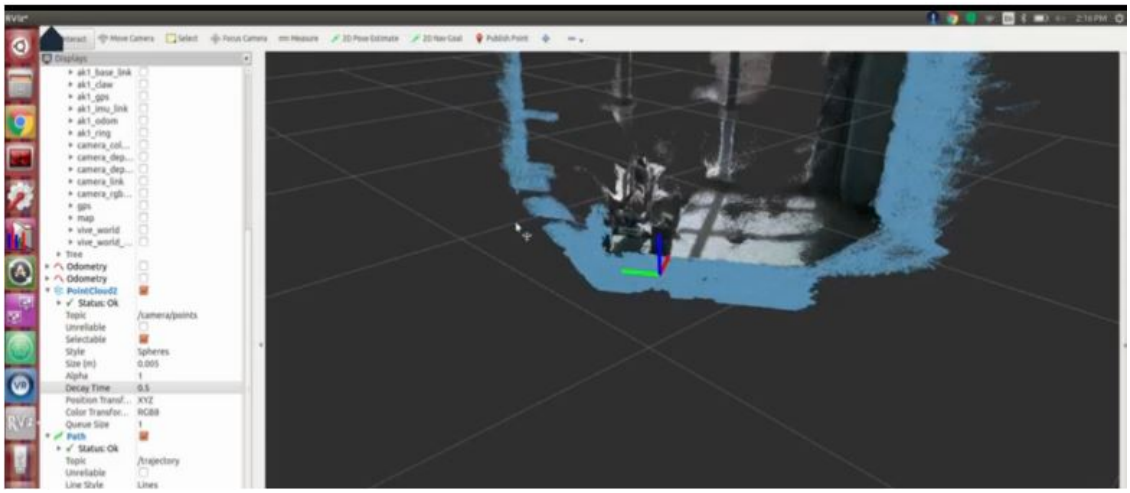

Figure 4: Point cloud from Intel RealSense

4 Subsystems and their Evaluations

4.1 Intel Realsense

We are able to get the point cloud from Intel RealSense on ROS Node. We have used the RTAB-Map ros package to convert this raw point cloud into a useful information like 3D-map of the environment and then generate the corresponding 2D occupancy grid of the environment.

We faced some issues because of kernel and ubuntu version in integration of the Realsense. So, we first used the Realsense viewer to directly view the 2D and 3D output from the Intel Realsense on windows and ubuntu. Then we added a the ros wrapper layer over it. we had to solve a lot of dependency issues to finally use it with the ros wrapper. After the independent testing of Intel Realsense, we integrated obstacle avoidance package with it, we've explained it in the next section.

4.1.1 Obstacle Avoidance

For obstacle avoidance, we've used RTAB-Map ros package. As shown in figure 5 for sensor input we're using Intel Realsense. It is RGB-D Graph based SLAM approach. It the first step, it extracts the 2D feature like SIFT, SURF etc from the images. These features are used for the global loop closure. The pose of the robot is calculated by the wheel odometry. The depth information is used to find the 3D position of these visual features. For efficient memory management, they've introduced the concept of working memory(WM) and long term memory(LTM). After a defined time T, the nodes are transferred from there WM to the LTM, to keep the WM nearly constant, while loop closures are detected for refining them, nodes are retrieved from WM to LTM.

The complete pipeline is like, first we get the 2D features from the RGB image, we get the 3D location of them using depth information. We find the corresponding 3D points in the next incoming frames and find the correspondences between them and we re-project them back in the 2D map. When we revisit these points, using the wheel odometry and the bag-of-words/ features we've detected we can do the loop closure. One additional very useful feature of this package is that it can be used for multisession mapping, so if it has already mapped some area, and later on if we only show a part of that area to it, it'll rebuild the whole map on its own. So we can map multiple different areas separately and can easily reconstruct a combined map.

For evaluation of the RTAB-Map package, we mapped whole Planetary Robotics Laboratory (shown in figure

6). So, we found out that, if we move the Realsense very quickly it messes up with the odometry and all the windows will turn red, to correct that error, you just need to go back to a previous location till where it had mapped properly with good odometry, you need not to initiate the whole process all over again. It is really good feature, as in most of the SLAM algorithms implementations, if you don't initialize properly or or loose the tracking in middle you've of to start the mapping process all over again. In addition to this as soon as you visit the same area again, using the features, it'll do the loop closure and you'll get a perfect 3D map of the area. Then, it'll convert this 3D map into 2D map, which was of our interest for this project.

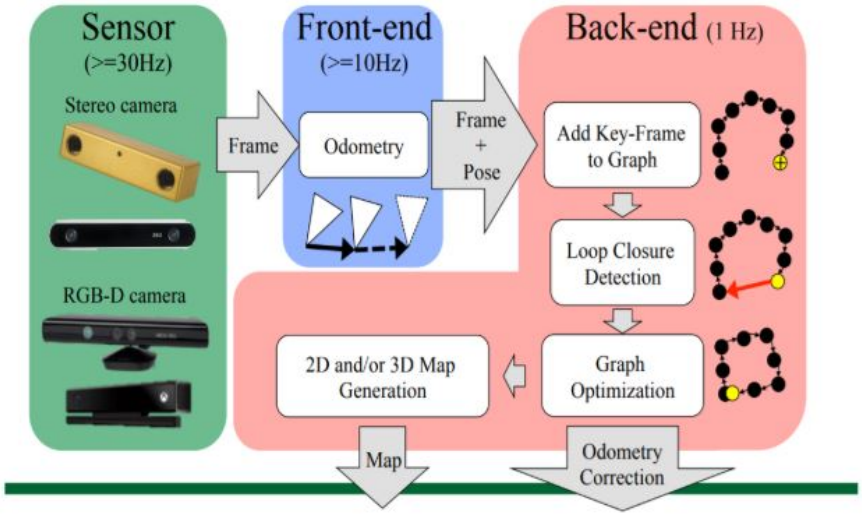

Figure 5: RTAB-Map (Real-Time Appearance Based Mapping)

Figure 6 represents the 3D output generated by this package and then in the figure 7, the corresponding 2D occupancy grid is generated. So the rovers are using just this 2D occupancy grid at present, but later on the 3D information might also be a helpful, as it gives the richer information about the environment.

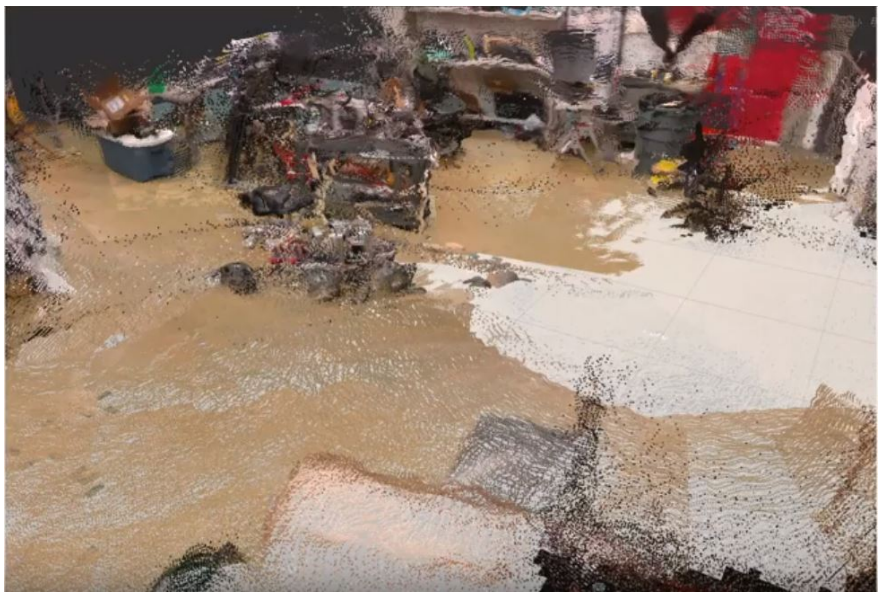

Figure 6: 3D-Map output of RTAB-Map (Real-Time Appearance Based Mapping)

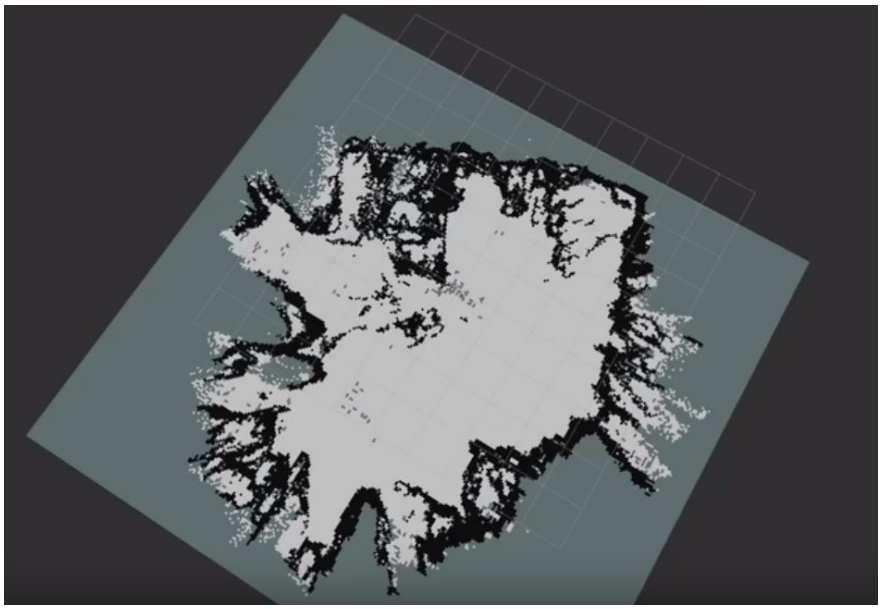

Figure 7: 2D-Map output of RTAB-Map

4.2 TEB (Timed Elastic Band) Local Planner

We have integrated the path planning algorithm - Timed-Elastic-band path planner (teb) for both the rovers.The initial trajectory generated by a global planner is optimized during runtime w.r.t. minimizing the trajectory execution time. We are able to do this on the basis of just the wheel odometry.

The teb_local_planner is a ROS package. It receives a destination as input and use both global and local planners for planning. The global planner could plan a path from starting point to the destination and the local planner could adjust the rover to follow the path.

We independently tested the teb planner without other subsystems. So, there is an option in rviz to provide a 2D Nav goal to the robot. The robot will try to move to that location in real world. And on rviz you can easily visualize the path the robot is planning. We can see in figure 8 the trajectory the robot is planning and according to the changes in environment or goal, it'll continuously update. This is a really good local planning package and the rovers were able to plan and execute the path pretty accurately using this planner.

4.2.1 Leap Frogging Behavior

The process for leap frogging is:

1. Give a destination for both rovers
2. Start one of the rovers planning toward the destination while the other remains still
3. If the distance between two rovers are bigger than a certain threshold, stop the moving rover and start the other one instead
4. Loop step 2 and 3 until one of the rover has reached the destination

Two rovers have their own teb_local_planner therefore they could plan to destination separately while receiving information from the other rover. There is a ROS node responsible for coordinating between two rovers. It will publish stop and start command to the rovers whenever the distance between two rovers is bigger than a certain threshold. The node subscribes to the Vive Trackers which publish the positions and

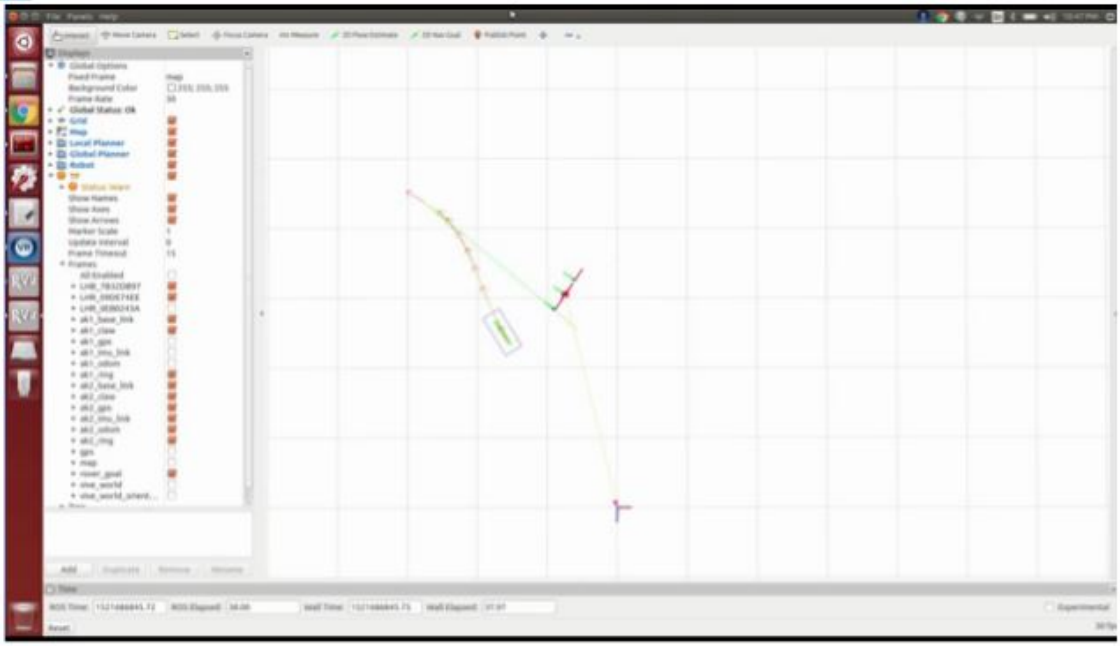

Figure 8: TEB local path planner

orientation for two rovers. Hence we could get the precise relative orientation and position of the rovers instead of the odometry published by the IMU.

4.3 Vive Tracking

We have integrated the Vive trackers with both the robots on one side. We've used the leapfrog approach and have placed Vive trackers on both side for both rovers . But when both the rovers are parallel to each other the Vive trackers lose its signal and the rover move only on the basis of the wheel odometry.

We had to extensively test the odometry of the robot when the vive trackers are able to see each other, and after passing the blind spot how the odometry correct itself. We were initially having issues in integrating these two as the odometry and vive trackers movement was very synchronous when one base station is able to see the vive tracker as it is constantly doing the pose correction, but as soon as the rover was coming in the blind spot the robots only move on the basis of the wheel odometry. But our wheel odometry wsa initially not that accurate and there was huge accumulation of error in that blid period and then the rovers were loosing the sync. But we were able to rectify these errors to get a good wheel odometry and no sudden jumps while the vive trackers are not functional.

(a) Vive Tracker

(b) Vive Base Station

5 Conclusion

We've drawn few conclusions from our testing:

- There are a lot of good inbuilt packages in ros, and we should explore them and build our work over them instead of reinventing the wheel. Like the TEB Planner or the RTAB-Map, they were able to make our localization, planning and obstacle avoidance task very easy.
- Sometimes the software compatibility issues can deter us from getting the desired result or debugging them can be very time consuming. For instance, the kernel or ubuntu issues in integrating the realsense with the system or the still unresolved issue of the DJI Camera.
- We've been able to integrate all these different sensors and planners, with some customization as per requirement. The system is able to achieve the desired functionality.

6 Future Work

We still believe that this system has more potential than what we've been able to accomplish in this small scope of our project. There can be a lot of things that can be built above this base system of ours.

- We are still figuring out a way to integrate the DJI camera, that will help in better localization at the same time getting good images from rovers.
- We've only used the 2D map of the RTAB-map, we can explore the option of using3D-map for better localization and mapping of the environment.

7 Organization and Reflection

7.1 Work Assignment

Name	Responsible Work
I-Chen Jwo (ijwo)	Install Firmware, Teb_Local_Planner, Reports
Pratibha Tripathi (pratibht)	Obstacle Detection, Reports
Pulkit Goyal (pulkitg)	Obstacle Detection, Reports
Daniel Arnett (darnett)	Obstacle Detection, Teb_Local_Planner, Vive Tracker, Integration
Akshat Agarwal (akshata)	Teb_Local_Planner, Vive Tracker, Reports

7.2 Components that take significantly longer than expected

- Installing firmware on the Jetson TK1 and DJI manifold

  In order to install RealSense on the Rover, we need to upgrade the kernel of Ubuntu 14.04. However, we didn't expect the upgrading the kernel would be such an lengthy task.

- Getting the relative position and orientation of Vive Tracker and the Base Station

  Each rover has its own Vive Tracker and the Base Stations are installed on the other rover. In order to get the position and the orientation of the Vive tracker, we have to calculate the transformation and quaternion between the Vive Tracker and the Base Station. Calculating these are not trivial task since we have to find the right transformation, inverse it and apply it to the right tf.

- Getting the DJI Cameara working

  Since every camera from DJI has its own matrix for connecting to the Manifold, we have to have the right matrix in order to control the camera or get an image from it. We had the wrong matrix at the first time and it took us forever to realize we have to reorder a new one. After ordering the right matrix, the Manifold(DJI processor) couldn't work properly. In the end, we change to RealSense for obstacle detection.

7.3 Link to Project videos!

- Symbiotic Localization: Project description
- Initial Modeling using RealSense and ViveTracker
- Creating Occupancy Grid using Real Sense and Vive Tracker
- Obstacle avoidance by the Rover: screen
- Obstacles avoidance by the Rover: FPV
- Planning stack of the Rover
- Leap frogging with single rover
- Final project integration: Leap frogging on both the rovers
- Combined Project Video