Improved Ground Robot Indoor Autonomous Navigation with MAV

Qianle Li
Email: qli59@stevens.edu
Advisor: Yi Guo

***Abstract*—Robot technology is moving from factories to everyday applications in households, offices and public areas. Autonomous navigation is the premise for robots to perform their tasks without human involvement. Several well-developed algorithms have been deployed on ground robots and they work fine in experimental environments and have been applied in many real robots. However, onboard sensors have limitations on environmental perception as farther obstacles hide from closer ones, which becomes a huge problem when the environment is crowded with both static and dynamic obstacles. By contrast, though aerial vehicles do not share that issue, they have far more errors when doing mapping and localization due to lack of odometry information. Therefore, this project tries to design a system which combines the perspective advantage of aerial robots and the accuracy of navigation and mapping algorithm based on ground robots. A popular used navigation stack and gmapping algorithm are deployed on the ground robot guide the robot and build an accurate map, at the same time, a Micro Aerial Vehicle (MAV) cooperates with the ground robot by feeding it with supplementary information which enables the navigation stack to plan a more efficient path without exploring the whole space in unknown environments.**

The system is built based on Robot Operating System (ROS) and popular tools including Gazebo and Rviz are used to visualize and analyze experiments. Though the system is not perfect, experimental results show that this is a promising research area. For convenience of further researches, I make the source code and experiments videos public on Github, readers may find the link in section VI.

Keywords—MAV, path planning, ROS, Gazebo, stereo vision.

I. INTRODUCTION

Nowadays, more and more robots are deployed to serve people in public areas. In most robot indoor applications, all jobs are done by ground robot itself. Usually, the ground robot is equipped with laser sensors, sonars or physical sensors to gather information from the surrounding environment and applying path-planning algorithms to find a path to the goal. In this project, the ground robot is equipped with a laser sensor which can detect obstacles in a 4-meter radius circle centered in the robot. The navigation stack and gmapping package are applied on the ground robot. Gmapping is a popular Simultaneous localization and mapping (SLAM) package and the navigation stack accounts for path-planning, maintaining the cost map and driving the robot. The combination of two packages is enough to guide the robot in most indoor environments. However, experiments in Gazebo environment shows that the navigation stack has relatively poor performance in a narrow, crowded or dynamic environment.

The reason is apparent. In populated areas such as shopping malls, museums and libraries, people move as a group, or they are forced to move in a limited space due to other static obstacles like shelfs and showcases. In those mentioned areas, it is almost impossible for onboard sensors like laser sensors and RGB-D cameras to provide full knowledge of the environment. The situation would be better if the robot has a prior knowledge of the environment with static obstacles, but problems still exist when dynamic obstacles like people, pets and other robots involved, which makes the known map unreliable.

In order to solve that problem, on one hand, the robot must be fed with more information. On the other hand, there has to be a way to fuse information from different sensors so that the ground robot can make use of it. In Ahmed's study, he installed optical encoders, a binocular camera, a laser rangefinder a compass and GPS modules on an electric golf carts to detect outdoor obstacles [1]. However, as so many modules installed, complexity and the cost of the robot raised. Besides, a robot with that size is not suitable to work in an indoor crowded area. Not to mention GPS signals are poor in indoor spaces.

In this project, a MAV is deployed as a new eye of the ground robot to provide an extra view for the ground robot. Though the complexity of the system is increased, the robot has a better knowledge of the surrounding environment, which results in better path-planning and saves much time from exploring unknown areas.

In order to detect obstacles, the MAV is equipped with a stereo camera and an image processing algorithm named Oriented FAST and Rotated BRIEF (ORB) detector [2]. The ORB detector picks out map points of input images and a layered 2-dimentional cost map is used to fuse observations from the laser sensor and the stereo camera [3]. During that process, a mechanism introduced by the work of Mur-Artal is applied to filter those useless map points [4][5].

As the MAV and the ground robot are two separate entities, there has to be a reliable communication system between them for messages exchange.Fortunately, ROS has its own communication system consists of nodes and topics which provides convenience to simulations like this project. The application in real world is not studied and tested in this project.

My main contribution is testing the possibility that the MAV is able to cooperate with the ground robot and provide the navigation stack with information that the laser sensor cannot offer. Several Gazebo environments are deigned to test the system. They show that MAV equipped with a stereo camera provides useful information that help the robot to generate more efficient path and saving plenty of times as well. However, pros and cons of having a MAV in the system should also be considered as it brings complexity to the whole system.

II. FORMULATION OF THE PROBLEM

This problem consists of two parts:

a) How does the MAV detect obstacles?

There are several options for the MAV to observe environment including cameras, IMUs, and sonars. In recent years, with the rise of Augmented Reality (AR), visual SLAM (vSLAM) becomes a popular research field. My thought appeared when I was reading the paper of ORB-SLAM. The experimental results including a 3D point cloud map impressed me. As shown in Fig 1, the map is quite accurate compare to the environment in Fig. 2. That's why I believe my project can make use of the ORB detector to detect obstacles.

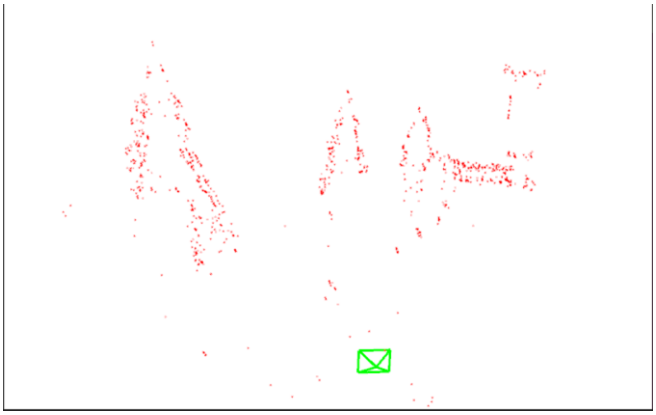

Fig. 1. Point Cloud Map Generated by ORB-SLAM

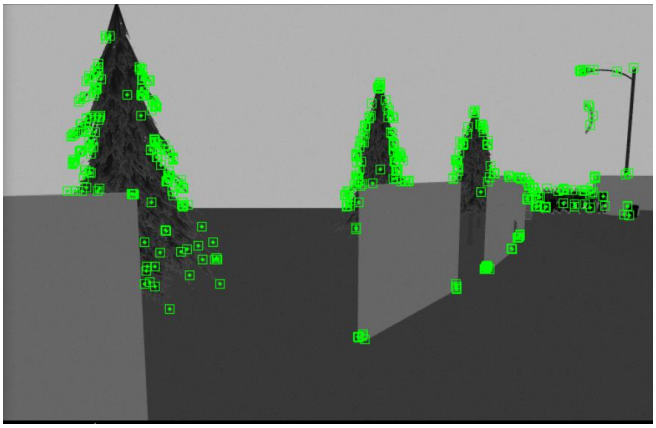

Fig. 2. Input Image related to Fig 1

b) How does the MAV inform the ground robot about those obstacles it detected?

The ground robot uses the navigation stack to navigate. In the navigation stack, it includes a 2d cost map to represent the environment [3]. Therefore, the question becomes to find a way to mark the detection from the MAV on the cost map. Fortunately, the design of the cost map has considered similar situations. The cost map is a layered map and obstacles observed by the MAV is expected to be marked by creating a new layer of the map.

III. My Design

My design has the following parts:

A. MAV with a stereo camera

The MAV simulation package named "RotorS Simulator" developed by Furrer is used in this project [6]. This package includes many real MAV models like Firefly, Humming Bird and Pelican, it includes a model of a stereo camera, VI-sensor, as well [7]. For different applications, the orientation of the VI-sensor can be different. For example, if the camera looks down from the MAV, it can see the whole area around the ground robot. Some researches suggest that it helps to recognize boundaries of obstacles and update the map. However, in my project, the robot is assumed to work in an indoor environment, where the flying height of the MAV is limited, which means camera can only observe a narrowed area surrounding the area. In that case, the MAV is useless. Therefore, I make the camera looking forward, so that it can detect the obstacles in a distance that the robot is heading to, and inform the robot to bypass and avoid it.

The package runs well in ROS and simulates a flying MAV in a Gazebo environment. With help of ROS topics and nodes, the pose of the MAV can be controlled easily.

B. ORB detector

The ORB detector algorithm is run on the MAV, which can extract feature points from input images. Since the Gazebo world is quite simple and lack of features, positions of feature points can be also treated as positions of obstacles. Apparently, this is not acceptable in a real-world application, and this is discussed in section V.

I modified and applied the existing ORB-SLAM package as it includes an ORB detector and I can make use of some parts of it so that I don't have to build the detector from scripts. The system overview of ORB-SLAM is shown in Fig. 3.

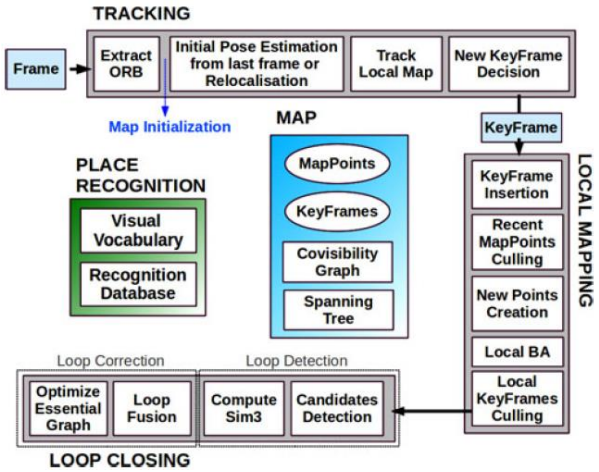

Fig 3. ORB-SLAM System Overview

There are three main parts that I modified and used in my project. First part is the ORB detector, which extract points from input images.

Besides, I found the mechanism that ORB-SLAM used to create its key frames is useful to my project. In ORB-SLAM, the map is consisted of key frames. Each key frame is created when the current frame satisfies certain requirements. A new key frame is created only when certain time passed, the current frame contains enough points and those points are very different from last key frame.

The ORB-SLAM package also calculates poses of all map points and frames. Each frame consists of map points. When a frame is created, its pose in the world coordinate is predicted by using a constant velocity model and it will be optimized later. When a map point is extracted from the input image, its pose in the world coordinate is calculated based on the frame pose. Obviously, the pose of a certain map point in a certain frame coordinate can be calculated based on poses of the map point and the frame in the world coordinate. Poses

of map points in a key frame are published so that they can be marked on the cost map.

In order to improve the performance of the ORB detector, three rules are made based on multiple tests.

a) Only publishes the new set of map points of the newest key frame when six new key frames are created.

This rule reduces the frequency of publishments and creates delays on purpose, so that the last publishment would stay on the cost map for a while, which saves the robot from dilemmas in which the robot cannot determine which path to take. In the environment shown in Fig 11, the MAV can successfully detect people behind bookshelves, as shown in Fig 12, then it can plan a path to bypass both crowd of people and bookshelves. However, if the robot starts moving and rotating, the MAV loses its view of the crowd, since map points are updated at every frame, obstacles represented last set of map points disappears without the rule, as shown in Fig 4.

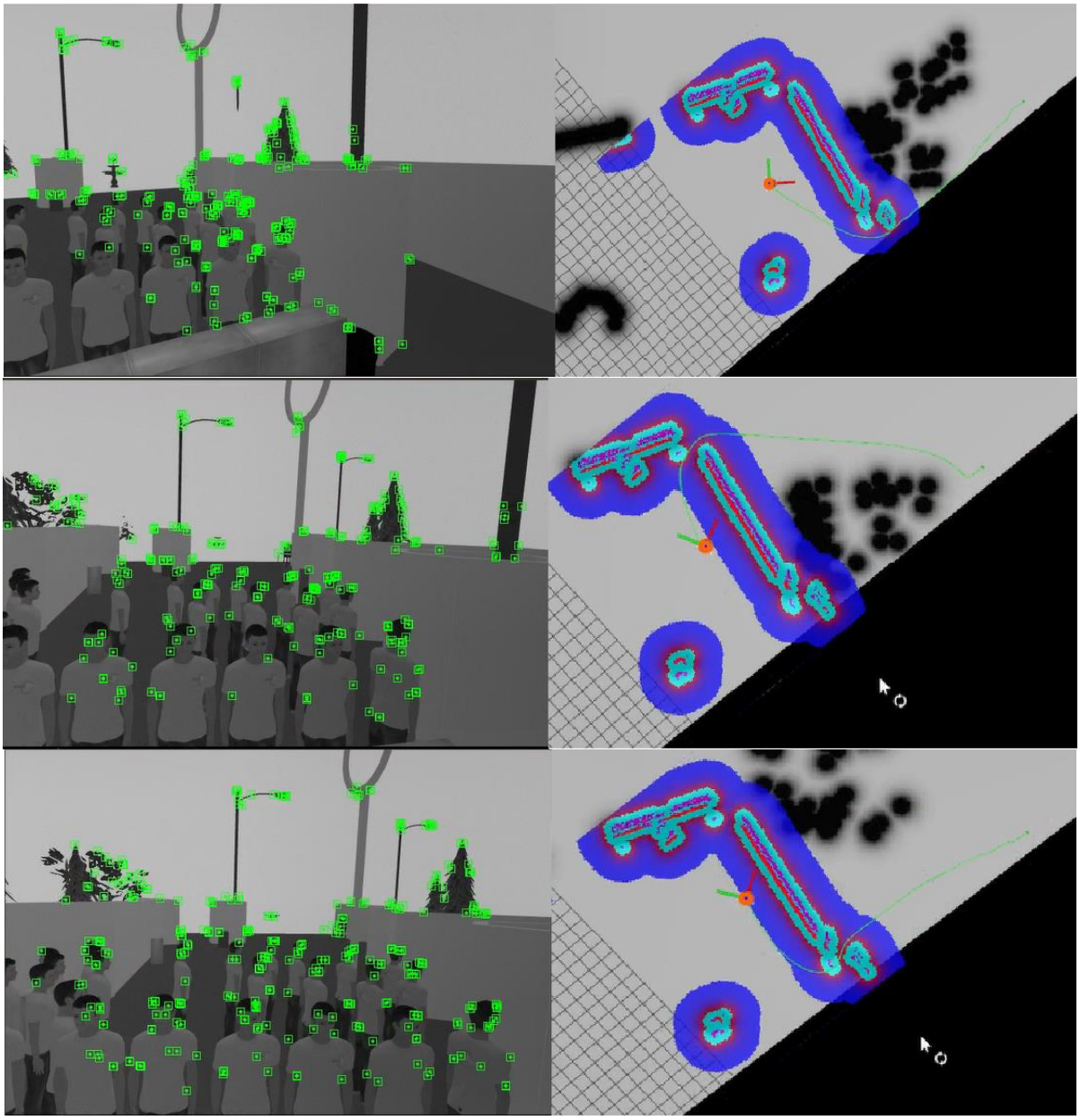

Fig. 4. Top: The MAV detected people behind bookshelves and the ground robot planed a path to avoid both bookshelves and people; Middle: As the ground robot rotated, the MAV lost vision on part of the crowd, which means part of obstacles on the cost map are lost, and the robot planed a path again without considering the missing part of the crowd; Bottom: As the robot rotated again, the missing part of crowd was detected by the MAV again, the robot went back to the path planned before.

In the example shown in Fig 4, the robot kept doing path-planning and can never get out of that dilemma by itself. In that situation, though lost the latest observation of the MAV, the delay gives the robot enough time to navigate on the most efficient path. Apparently, this method could cause new problem, which is discussed in section V.

Another big problem of the ORB detector is that it cannot correctly calculate 3D positions of map points in a distance farther than 10 meters. For example, the point clouds in Fig 1 represent the environment in Fig 2 very well. However, if we rotate the point clouds, which is shown in Fig 5, we can see that only points of the nearest tree， which is in the green circle, are correctly positioned in the map. Besides, though

ORB detector has good performance in detecting features, errors occur and it extracts points from areas with no feature at all. Points in the orange circle are those error points. Those positions would be treated as obstacles by the cost map, which leads to chaos on the map. Therefore, it is necessary to filter out all useless points to make sure that all points represent obstacles.

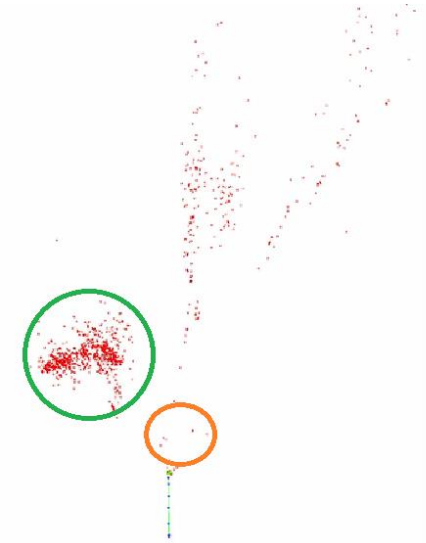

Fig. 5 Rotated 3D point cloud map

Two rules are made to retain useful points, but to illustrate them, I have to mention the image pyramid strategy first. In image processing of ORB-SLAM, the input image is processed in 8 levels. The image is amplified from level 1 to level 8 in order to extract all features in the image, which means the level of pyramid is positive related with the depth of the point to some extent. Based on all I discussed above and multiple tests, the ORB detector works well when it:

b) only retain points with depths less than 10.

c) only retain points which are detected at level higher than 4.

With the implementation of the above three rules, the quality of map points is improved which allows the MAV to send obstacle information with acceptable accuracy and limited error map points.

C. Ground robot with a laser sensor

A ground robot model which is created on my own is used in this project. The robot is equipped with a laser sensor and an RGB-D kinetic camera. However, only the laser sensor is used as it provides a wider range detection. The navigation stack and gmapping algorithm are run on the ground robot, which means that it can navigate and explore the environment without the MAV.

D. Pose Syncronization

In my project, the MAV keeps hovering above the ground robot at the height of 2.5 meters, which is shown in Fig 6. They synchronize their location through ROS nodes and topics so that the MAV is able to keep searching feature points in front of the robot and publish sets of points positions to the ground robot.

To make the simulation more realistic, the hovering height is set as 2.5 meters because floor heights normally range from 2.8 meters to 3 meters normal people do not have that height.

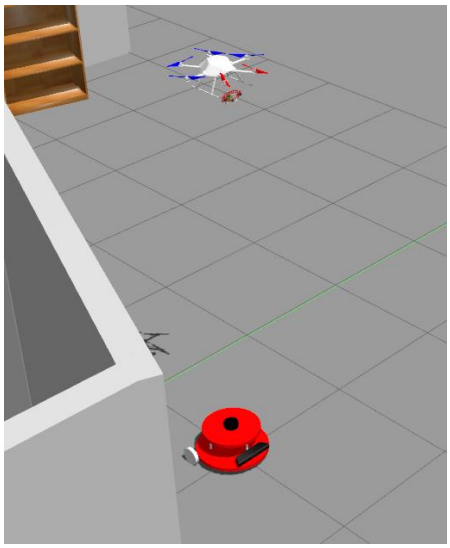

Fig. 6. Relative Poses of the Ground Robot and the MAV

E. Environment

Environments are designed in Gazebo to simulate librarian scenario, one example is shown in Fig 7.

In the example environment, four columns of book shelves are set as static obstacles, which creates three passages. In a library, some people stand in passages reading or looking for books, others walking through passages looking for other book sections. Therefore, standing people and walking people act as static obstacles and dynamic obstacles are added into the environment later to test the performance of the system.

For every experiment, the initial position and the goal position of the robot is the same. By setting different locations of human obstacles and comparing the performance of the robot navigation with or without the MAV, we can see if the involvement of the MAV can improve the performance. More details of environments will be discussed in section IV.

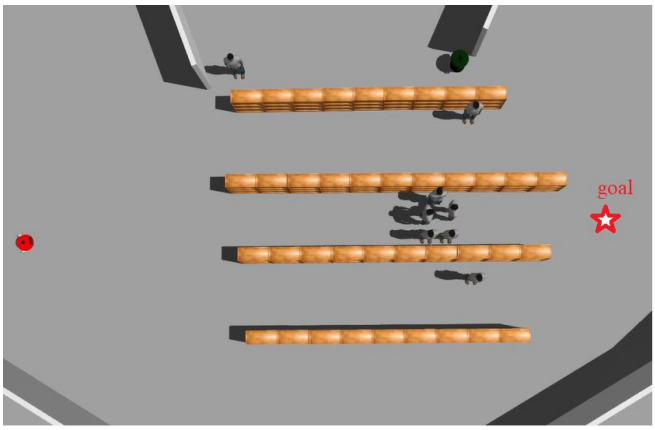

Fig. 7. Example of Text Environment

F. 2D Cost Map

2D Cost Map is an important component of the navigation stack and the key to fuse observations from the laser sensor and the stereo camera.

As shown in Fig 8, the cost map is usually made of three layers, which is static layer, obstacle layer and inflation layer. Relatively, these layers represent static obstacles from a known map, discovered obstacles and inflated areas. Each layer is an interface to update the map master, which is a grid

map. The default configuration of the cost map is shown in Fig.3. The cost map can work efficiently and reliably under this mechanism. As Lu claims in his paper, users can build their own layers as plugins to modify the cost map. Therefore, I created a new layer named “SimpleLayer” to mark those map points on the cost map.

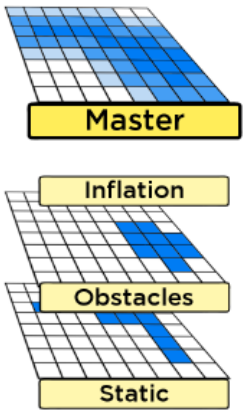

Fig. 8. Layers of a Typical Cost Map

In order to mark map points published by the ORB detector, the new layer runs the updateBounds method to update the size of a bounding box to determine how much of the cost map it needs to update, then the updateValues method updates the values of the cost map cells within the bounding box.

G. Coordinate Transformation

As commonly used applications, Gazebo, Rviz, and the cost map share the same coordinate. However, those map points are in the camera coordinate of the ORB-SLAM package, which is totally different from the Gazebo coordinate. Therefore, coordinate transformation is necessary before updating the cost map.

As shown in Fig 9, z axis in the camera coordinate is x axis in the Gazebo coordinate, x axis in the camera coordinate is the opposite of the y axis in the Gazebo coordinate.

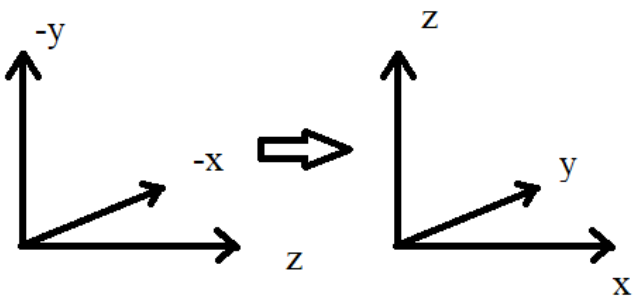

Fig. 9. Left: Camera Coordinate; Right: Gazebo World Coordinate

In addition to the coordinate transformation, as the robot moves and rotates during the navigation, map points poses have to be rotated and translated in order to represent the correct positions of obstacles. Fortunately, the cost map provides the current pose of the robot including x, y and yaw.

True poses of map points can be calculated by using rotation and translation matrixes.

For clockwise rotation, the rotation matrix is:

$$R = \begin{bmatrix} \cos(yaw) & -\sin(yaw) \\ \sin(yaw) & \cos(yaw) \end{bmatrix}$$

For counter-clockwise rotation, the rotation matrix is:

$$R = \begin{bmatrix} \cos(-yaw) & \sin(-yaw) \\ -\sin(-yaw) & \cos(-yaw) \end{bmatrix}$$

The transition matrix is:

$$T = \begin{bmatrix} robot\_x \\ robot\_y \end{bmatrix}$$

Transform the pose in camera coordinate to the pose in Gazebo coordinate:

$$P_g = RP_c + T$$

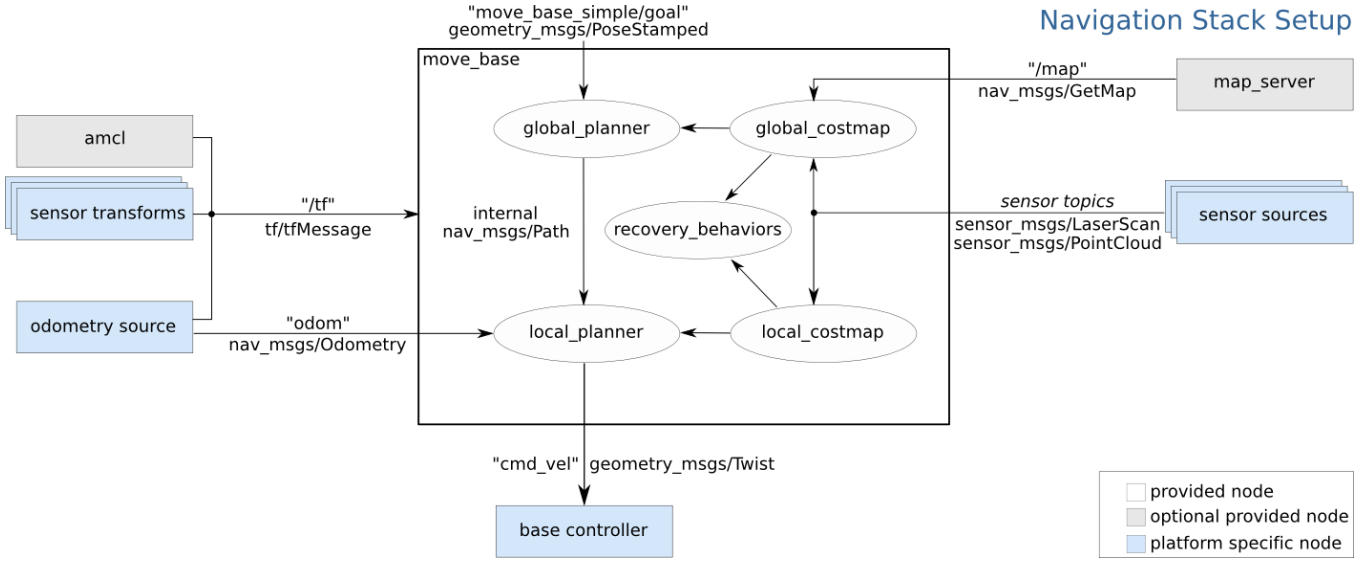

Fig. 10. Navigation Stack Setup

H. Navigation Stack and Gmapping Package

The navigation stack is a commonly used package for ground robot to do path-planning and navigate. It takes in information from odometry and sensor streams and outputs velocity commands to send to a mobile base.

The move base package is the major component of the navigation stack, as shown in Fig 10. In this project, optional nodes including map server and AMCL are not applied, which means the robot has no prior knowledge about the environment. Therefore, gmapping package are used to map the environment.

Gmapping is a popular SLAM algorithm in ground robot application. It takes in odometry and sensor information and build an occupancy grid map, which is compatible with the 2D cost map in the navigation stack. Since the gmapping requires odometry, its application is limited to ground robots.

Custom parameters including robot radius, max velocity and max acceleration are modified to fit my own ground robot. Besides, A* algorithm and Dijkstra algorithm are available for path-planning, and the former one is applied since it is much faster than the later one.

The ground robot is able to navigate with those two packages, more details will be discussed in section IV.

IV. System Demonstration

A. Experiments Goals

My experiments are conducted to prove the involvement of MAV can improve the path-planning and navigation performance of the navigation stack. Therefore, I set three goals for my experiments:

a) Test if the stereo camera on the MAV is able to detect obstacles.

b) Test if the MAV can improve the performance of navigation stack.

c) Test if the whole system can be applied to a dynamic environment.

B. Experiment 1

This experiment can prove the stereo camera is able to detect obstacles. The test environment is shown in Fig 11.

In this environment, I placed a column of bookshelves before the crowd of people, so that I could make sure that the laser sensor on the ground robot cannot detect those people behind those bookshelves. The output image of ORB detector and the simulation in Rviz is shown in Fig 12.

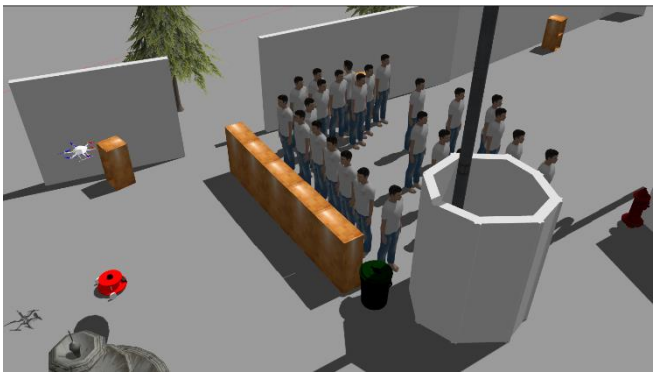

Fig. 11. Environment for experiment 1

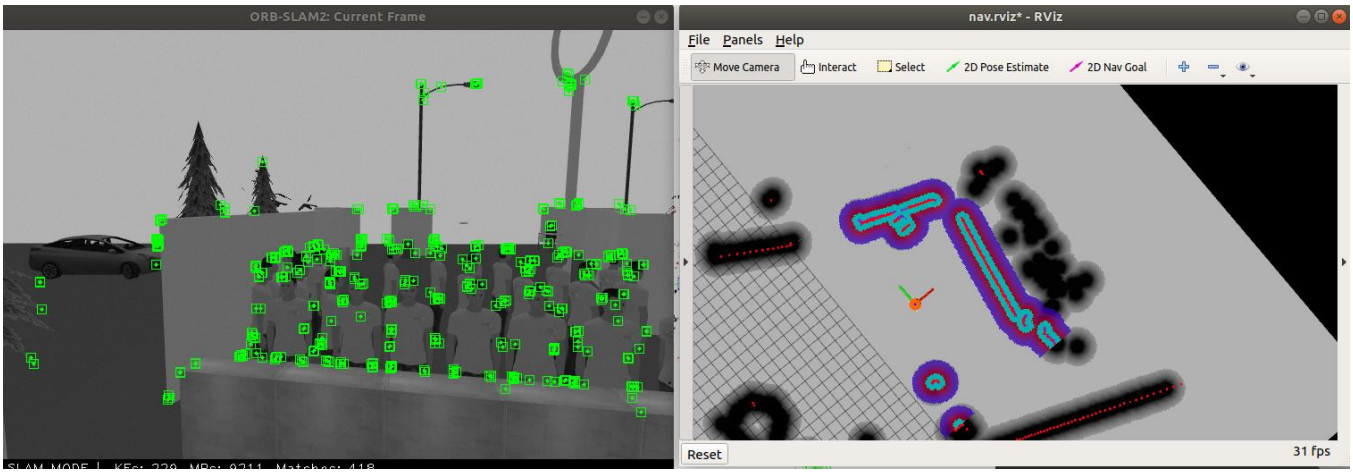

Fig. 12. Left: Output Image from the ORB Detector; Right: Rviz Simulation

The experimental result proves that all parts of the system work well and the MAV is able to detect obstacles that cannot be detected by the laser sensor.

C. Experiment 2

The environment of experiment 2 an is shown in Fig 8. I want to prove the MAV is able to improve the performance of navigation stack. As I mentioned before, the environment is simulating a library scenario, where people standing in the middle of the middle passage while reading books.

In order to compare the navigation performance with and without the MAV, both two situations were tested.

Fig. 13 shows the test in which the MAV is not deployed, the robot detected those people obstacle after moving into the middle passage. The DWA local planner kept trying to plan a path which was similar to the path generated by the A* algorithm. After lots of failures, it started the recover process in which the robot rotated and tried to do global path planning again. However, as the laser sensor could not detect the space inside bookshelves, it planned a wrong path again, which is shown in Fig. 14. After the robot fully explored the space, it planned an available path to the goal. This process took it almost 2 minutes.

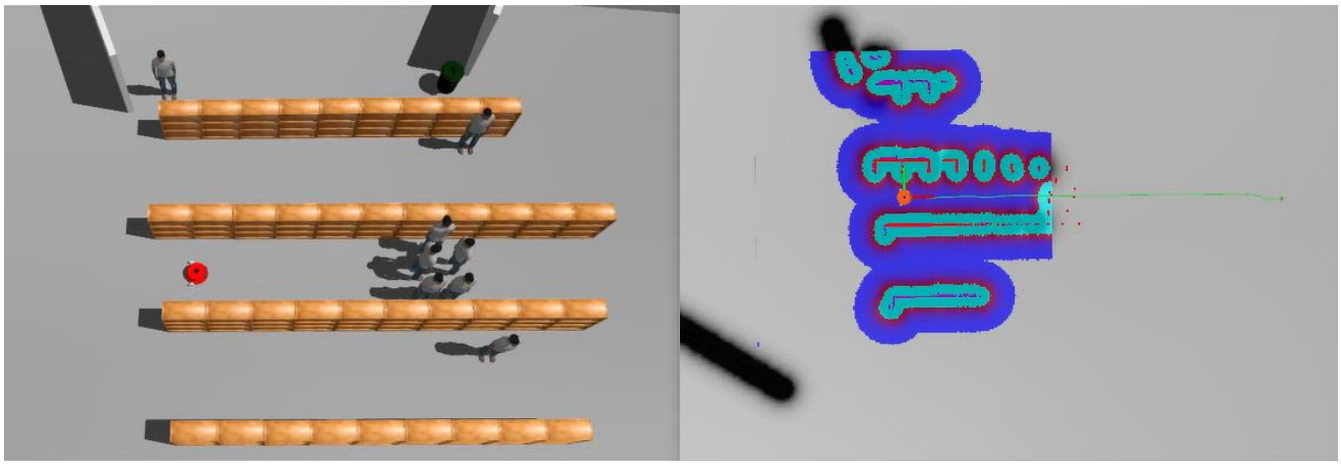

Fig. 13. Left: Gazebo Environment; Right: Rviz Simulation

Fig. 16 shows the test in which the MAV is involved, which makes the robot capable of detecting the people obstacles in a distance. With the MAV, only one minute is required to navigate to the same goal, which strongly proved that the MAV can greatly enhance the performance of the navigation stack.

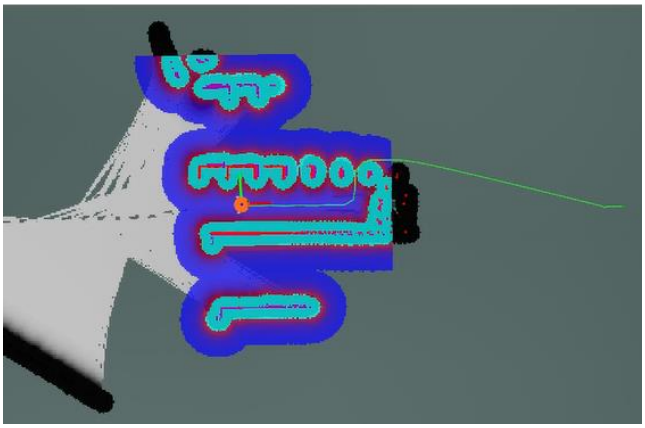

Fig. 14. The robot planned a wrong path as obstacles hidden behind others.

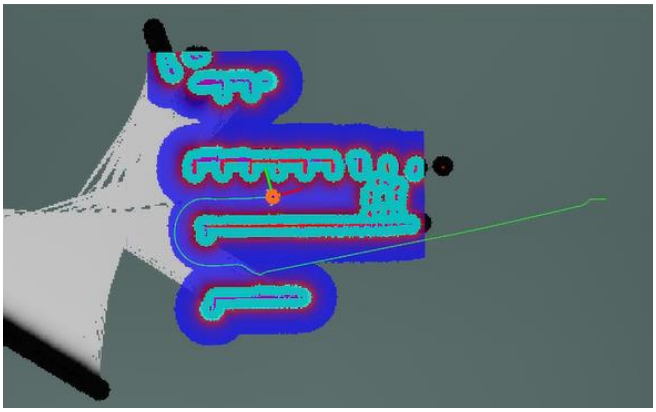

Fig. 15. The robot finally planned an available path after it wasted much time on exploring space.

However, such an environment is too simple to show the capability of the MAV, so I modified the environment and put more people obstacles either in the left or the right passage and test if the MAV can provide enough information so that the robot can always choose the right passage to navigate. The modified environment is shown in Fig. 17.

Note that the positions of new people obstacles is carefully chosen to make sure that the onboard laser sensor cannot detect them if the robot follows the path to go through the middle passage. The result in Fig. 18 shows that the MAV is able to detect people in adjacent passages which cannot be detected by the laser sensor and update the cost map so that the navigation stack guides the robot to a more promising path.

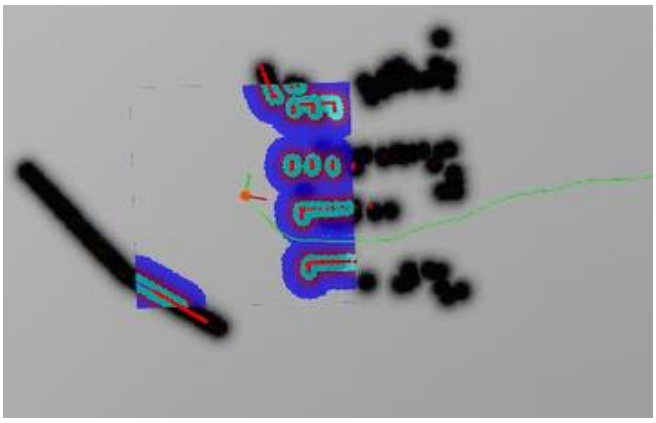

Fig. 16. The robot detected those people in advance and bypass them before entering the middle passage.

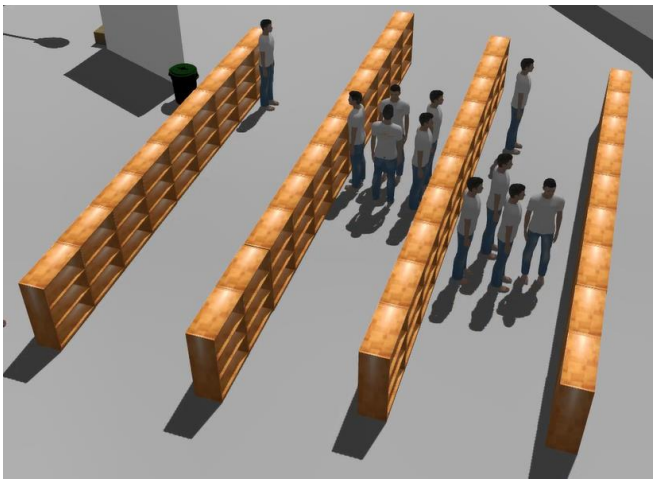

Fig. 17. People block the middle and the right passage.

Besides, similar environment with blocked left passage and free right passage was tested as well, and the robot always navigated through the less crowded passage. Therefore, it is safe to say that the involvement of MAV improves the performance of the navigation stack.

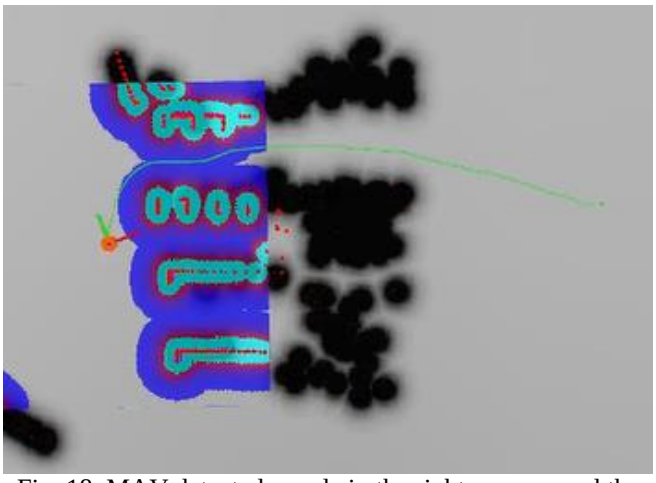

Fig. 18. MAV detected people in the right passage and the navigation stack planned a path through the left passage.

D. *Experiment 3*

In real world, robots usually work in public areas where people move around. Therefore, testing the system performance with moving people is necessary.

Gazebo provides animated models (actors) to simulate walking people. I replaced standing people obstacles in the middle passage with two walking actors, the new environment is shown in Fig. 19. The result is similar to the result of experiment 2, which proves that the ORB detector is able to extract points from those actors and the system can be applied to dynamic environments. The result is shown in Fig. 20.

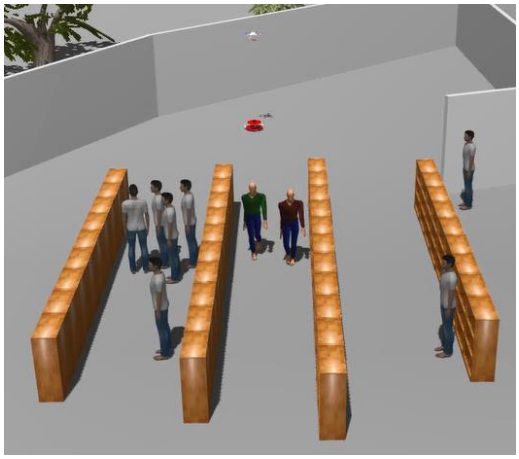

Fig. 19. Environment with two walking actors.

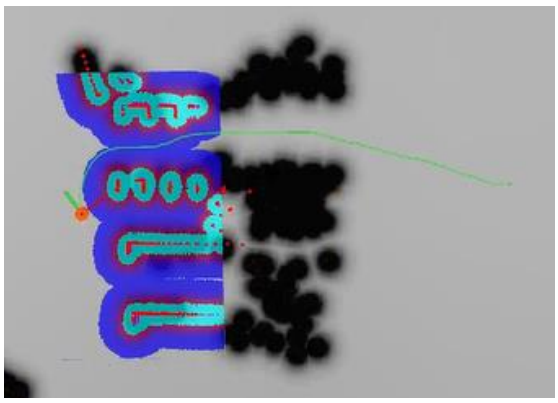

Fig. 20. MAV detects two walking actors and standing people in the right passage.

V. PERFORMANCE DISCUSSIONS

The experiments prove the possibility of my thought that the MAV equipped with a stereo camera is able to improve the performance of the navigation stack which is commonly used in ground robots. However, as a result of limited time and my capability, this system required further research and development. There are several problems from my point of view:

a) Inaccurate calculation of poses.

The original ORB-SLAM package is a SLAM package, which means the accuracy of poses of map points is not as important as it is in path-planning and collision avoidance. Therefore, it only use trilateration to calculate those poses, which make the results inaccurate and unreliable, especially when map points are in a distance. In my opinion, this can be solved by either applying a more advanced algorithm to calculate poses, or replace the stereo camera with a RGB-D camera so that the input images comes with depths. Besides, the original ORB-SLAM package supports RGB-D cameras, which decreases the difficulty of the replacement.

b) Application in real world

The ORB detector extracts feature points from input images. The Gazebo environment is lack of features, which forced me to put trees and other obstacles to the environment to keep the package working. Because of the same reason, features extracted from the image can be treated as obstacles. However, real world is full of features. For instance, textures, advertising board, rubbishes. All those elements make the environment even more complicated. Some of them should be marked as obstacles, others may not. Hense, this is a tough problem must be solved before real-world application.

One possible solution is to use depth cameras or calculate accurate poses of features so that most features can be filtered by heights, depths or other possible policies. Another possibility is to introduce artificial intelligence, but AI is really not my field. Needless to say, this need a lot of work and research.

c) Pose syncronization between the robot and the MAV.

In this project, the pose of the robot and the MAV are syncronized, but even in the simulation, communication beteween two nodes takes time. As a result, a difference between those two poses always exists. Just like what is shown in Fig. 21. If the robot rotates from orientation A to orientation B, when it finishs its rotation, the MAV is still in the middle of the rotation and its current orientation is C. If the ORB detector extracts map points from the current input images, then those map points cannot be marked in the right position because the SimpleLayer uses robot's pose to tranform coordinates. Sometimes, such an error may lead to a misunderstanding of the environment and make the situation worse.

An example is shown in Fig. 22, The obstacles in the red circle represent the column of bookshelves. However, they are marked in wrong positions because of the angular difference between the ground robot and the MAV. Fortunately, in this experiment, this error did not make a bad influence to the result. However, it may cause failure in a more complicated real-world environment.

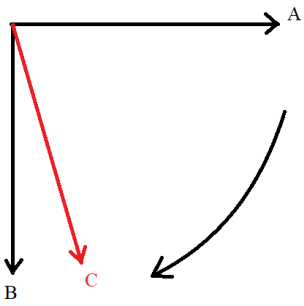

Fig. 21. Angular difference between the robot and the MAV

One way to limit the error is to slow down the angular velocity of the ground robot, so that angle between the robot and the MAV is narrowed. Nevertheless, this method cannot solve the problem fundamentally, it also limits the flexibility of the robot.

Another way is to using the pose of the MAV to transform the coordinate, which means the SimpleLayer plugin has to subscribe from one more topic. The programming may be tricky, but this is a simple and straightforward solution after all.

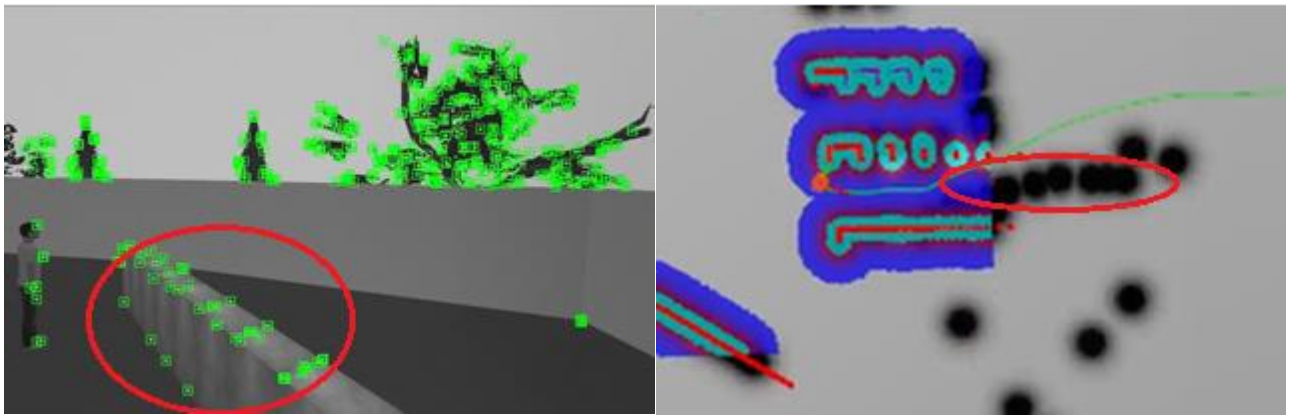

Fig. 22. Left: Input image; Right: Cost map

d) The delay created on purpose.

As I mentioned before, in section III, I create delays on purpose to make the obstacles on the SimpleLayer stay for a few seconds so that the robot has enough time to navigate and stick to the path it planned before. This is a good policy but has a flaw. If error occurs because of problems I discussed above, the error can only be corrected when several key frames passed. However, if the error stops the robot, which means the key frames can hardly be created, it may result in a totally different path, which may cost the robot more time.

This problem is caused by the inaccurate calculation of poses and angle difference between the robot and the MAV. Therefore, possible solutions for this specific problem are illustrated before.

VI. Conclusion

Navigation is a necessary capability for an autonomous mobile robot. As more and more service robots are applied in public areas, it is a big challenge to do path-planning considering unpredictable factors like human beings. In this project, a MAV is deployed to provide additional information for navigation stack to save time for space exploring and make plan a more efficient path.

The experimental results show that the MAV provides information that is complementary to the detection from the onboard laser sensor, enables the navigation stack to do path-planning with a better knowledge of the environment without fully mapping the whole environment. Besides, the system is capable of working in a dynamic environment, which makes it promising to be deployed in a real environment.

However, problems exist and further development is required for application in real world. The main problem of this system is that it is not very accurate in poses calculation so that it can only feed the navigation stack with very ambiguous information, otherwise it can build the map together with the gmapping algorithm or even map the environment alone.

I want to thank professor Yi Guo for her patience and her instructions which guide and support me through this project. I also appreciate necessary helps from her phD student Guang Yang. I could not finish this project without their helps.

For convenience of further researches, the source code and videos of experiments are all shared on https://github.com/QianleLi/EE800.git.

References

[1] Hussein, Ahmed, Pablo Marín-Plaza, David Martín, Arturo de la Escalera, and José María Armingol. "Autonomous off-road navigation using stereo-vision and laser-rangefinder fusion for outdoor obstacles detection." In 2016 IEEE Intelligent Vehicles Symposium (IV), pp. 104-109. IEEE, 2016.

[2] Rublee, Ethan, Vincent Rabaud, Kurt Konolige, and Gary Bradski. "ORB: An efficient alternative to SIFT or SURF." In 2011 International conference on computer vision, pp. 2564-2571. Ieee, 2011.

[3] Lu, David V., Dave Hershberger, and William D. Smart. "Layered costmaps for context-sensitive navigation." In 2014 IEEE/RSJ International Conference on Intelligent Robots and Systems, pp. 709-715. IEEE, 2014.

[4] 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, no. 5 (2015): 1147-1163.

[5] Mur-Artal, Raul, and Juan D. Tardós. "Orb-slam2: An open-source slam system for monocular, stereo, and rgb-d cameras." IEEE Transactions on Robotics 33, no. 5 (2017): 1255-1262.

[6] Furrer, Fadri, Michael Burri, Markus Achtelik, and Roland Siegwart. "RotorS—A modular gazebo MAV simulator framework." In Robot Operating System (ROS), pp. 595-625. Springer, Cham, 2016.

[7] Nikolic, Janosch, Joern Rehder, Michael Burri, Pascal Gohl, Stefan Leutenegger, Paul T. Furgale, and Roland

Siegwart. "A synchronized visual-inertial sensor system with FPGA pre-processing for accurate real-time SLAM." In 2014 IEEE international conference on robotics and automation (ICRA), pp. 431-437. IEEE, 2014.