2D Grid Mapping and Navigation with ORB SLAM

CMPUT 631 Project Report

Abhineet Kumar Singh
Ali Jahani Amiri

1 Introduction

Simultaneous Localization and Mapping (SLAM) is an important field in robotics and autonomous navigation. It is the process by which a robot moves around an unknown environment and uses its sensor and odometry information to build a map of the environment while simultaneously estimating its location within this map. SLAM is indispensable for the autonomous operation of robotic systems for both simple indoor applications like automatic vacuum cleaners as well as far more complex outdoor ones like self driving cars. Using sophisticated 3D sensors like LiDAR and Kinect, SLAM is now by and large considered a solved problem at least in most non-challenging indoor environments [1]. However, these sensors have limitations like the high cost of LiDAR and the limited range of Kinect which render the former unusable for low cost systems and the latter for outdoor scenarios. As a result, visual SLAM - where only one or two 2D cameras are used as sensors - is still a popular area of research. It is also a very challenging one since the absence of direct 3D information means that the 3D structure of the scene has to be deduced by matching features in multiple images of the scene taken from different viewpoints. However, since the rich visual information from cameras is needed anyway for reliable loop closure detection even in the presence of 3D sensors, it can be highly advantageous if the same information can also be used for map generation and localization. Hence, the potential benefits of getting such a system to work are also significant.

ORB-SLAM [2, 3] is a state of the art visual SLAM system that is capable of creating a point cloud based map of even challenging outdoor environments using only a monocular camera. Though this point cloud can be useful for obtaining the 3D structure of the environment, it is not as useful for path planning and navigation using algorithms that need a 2D occupancy grid map [4] as input. The point cloud produced by ORB SLAM is somewhat sparse which makes it difficult to generate an occupancy map that contains most of the obstacles while also providing sufficient contiguity in the known free spaces for path planning to work reliably. This is the problem that this project aims to solve by finding a way to build an occupancy grid map in real time using the 3D points produced by ORB SLAM. The grid map should be good enough for the standard navigation stack of ROS [5] to be able to use it to generate navigation commands that can allow a robot (actual or simulated) to follow the camera trajectory also produced by ORB SLAM.

2 Literature Review

Occupancy grid map generation and obstacle detection from a point cloud is a fairly well researched topic in the literature. A recent work by Goeddel et. al [6] presented a method for extracting a 2D map from 3D LiDAR data for performing localization. It works by imposing a verticality constraint on each point by thresholding on the slope of the plane that the point is estimated to lie in. In order to discriminate between obstacles at a finer level, it uses two different thresholds - a smaller one (15 degrees) for detecting navigation hazards and a larger one (80 degrees) for detecting slammable structures. Two additional constraints are imposed on obstacles to further reduce false detections. Firstly, only those points are considered whose z-height from the ground is within a specified range. Secondly, the number of occupied voxels present in the vertical line containing the point is required to exceed a threshold. The core idea of using slope thresholding to determine obstacles has also been used by Huesman [7] to convert a point cloud to a 2D occupancy map. Our own approach too uses this as one of the ways to detect false obstacles.

A somewhat different approach based on natural neighbor interpolation was used by Beutel et. al [8] to generate both 2D and 3D grid maps where the latter contain surface elevation information in addition to occupancy probability. The ROS grid mapping library [9] also provides functions to generate several types of 2D grid maps from a variety of input sources though we could not find a way to use it to generate occupancy maps from 3D point clouds. The learning based approach proposed by Thrun [10] that uses expectation maximization to directly generate occupancy maps from sensor data is likewise not applicable to our case.

Unlike the above methods that only use 3D data, Santana et. al [11] also take advantage of the available image data by performing color based visual segmentation of the scene to divide it into floor and non floor regions. This is followed by using the homograhy matrix generated by SLAM to map the floor parts of the image to free cells in the occupancy map. We considered using this method to refine our map but did bot have enough time to implement it. Also, performing image segmentation along with running ORB SLAM and our base grid map generation algorithm in real time would be very challenging on mobile computing hardware. Another possible approach to solve our problem is to convert the point cloud into simulated LiDAR scans using, for instance, the `pointcloud_to_laserscan`

ROS package [12] and then using the resulting data as input to a mapping algorithm like Gmapping [13] that can produce occupancy maps from LiDAR scans. However, this method might introduce additional uncertainty into the map since the mapping algorithm always assumes the LiDAR data to be noisy while in our case the point cloud produced by ORB SLAM is known to contain only reliable points. This approach was therefore also not considered.

3 Methodology

3.1 Common Components

This includes the project components common for all students and consists of the following tasks:

3.1.1 Review of literature on using vision to build a grid map

This part has been detailed in Sec. 2.

3.1.2 Installation of ORB-SLAM

This part consisted of installing ORB-SLAM2 using the instructions provided on the project Github page [14]. The current version of ORB-SLAM2 has some conflicts with Eigen 3.3.3 and OpenCV 3 on ROS-Indigo. As a result, we used Eigen 3.2.10 and did not install OpenCV 3, instead using the OpenCV 2.4 that comes with ROS-Indigo.

3.1.3 Calibration of camera

For this part, we followed the instructions provided on the ROS camera calibration tutorial [15] to calibrate the Logitech C615 1080p web cam that we used for generating our testing sequence (Sec. 3.1.7). The camera parameters thus obtained are given in Table 1.

Table 1: Camera Calibration Parameters

Camera Matrix	**fx**	**fy**	**cx**	**cy**	
	642.994934	647.678101	315.938509	235.397152	
Distortion Coefficients	**k1**	**k2**	**p1**	**p2**	**k3**
	-0.084841	0.041748	-0.007377	0.004092	0.000000

3.1.4 Reproduction of results on KITTI and TUM

For this part, we first downloaded sequences 00 and 05 of the KITTI dataset [16] and sequence `fr3_walking_halfsphere` from the TUM dataset [17]. We then ran ORB-SLAM on all 3 sequences following the instructions on the Github page [14] to reproduce the results. Screen shots of the point clouds thus generated are shown in Fig. 1. Following commands were used for running ORB SLAM on the three sequences:

```
./Examples/Monocular/mono_kitti Vocabulary/ORBvoc.txt Examples/Monocular/KITTI00-02.yaml
     ./KITTI/00
./Examples/Monocular/mono_kitti Vocabulary/ORBvoc.txt Examples/Monocular/KITTI04-12.yaml
     ./KITTI/05
./Examples/Monocular/mono_tum Vocabulary/ORBvoc.txt Examples/Monocular/TUM3.yaml
     ./TUM-RGBD/rgbd_dataset_freiburg3_walking_halfsphere
```

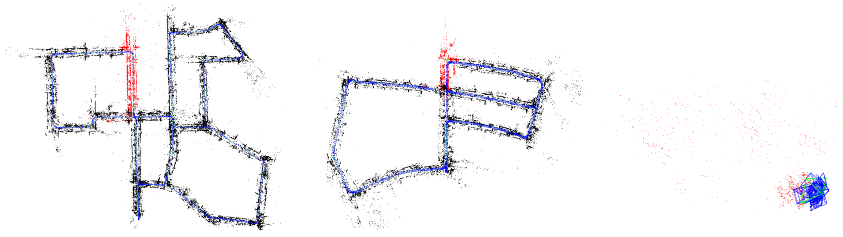
Figure 1: Screen shots of the point clouds produced by running ORB-SLAM on KITTI and TUM datasets: from left to right: KITTI 00, KITTI 05 and TUM `fr3_walking_halfsphere`

3.1.5 Production of 2D grid map using ORB-SLAM map points

For this step, we created a Python script to generate a 2D occupancy grid map off line by processing all the keyframes and map points produced by ORB SLAM after it has completed processing all the frames in the sequence. This script is included with this report as `pointCloudToGridMap2D.py`.

In order to generate the input data needed by this script, we modified the ORB SLAM monocular application `Examples/Monocular/mono_tum` to output the 3D poses of all keyframes along with all map points visible in each keyframe to a text file. The script parses this file and stores all keyframe/-camera poses and the associated map points into a dictionary structure after projecting them to the XZ plane. This projection is done by simply removing the y coordinate. Since the ORB SLAM coordinate locations are in units of meters, a finer grid resolution is obtained by multiplying all positions by a scaling factor which is chosen as the inverse of the desired resolution in m/cell. For instance, if a resolution of 10 cm/cell or 0.1 m/cell is needed, the scaling factor becomes 10. The keyframes are then processed one at a time and following steps are applied to all map points visible in each keyframe:

1. Cast a ray from the camera position to all the visible points using the Bressenham's line drawing algorithm [18].

2. Increment a visit counter for each point along the ray and an occupied counter for the end point that corresponds to the location of the map point.

The visit and occupied counters are stored as integral arrays of the same size as the range of x and z locations of the camera and map points after scaling. Note that ORB SLAM considers the XZ plane as the horizontal plane so that the y coordinate of a point is regarded as its height. Once all keyframes have been processed, the occupancy probability for each cell of the grid map is computed as:

$$p_{free}(i,j) = 1 - \frac{occupied(i,j)}{visit(i,j)} \tag{1}$$

where $occupied(i,j)$ and $visit(i,j)$ are the corresponding entries in the occupied and visit counters respectively and p_{free} is the probability that this cell is not occupied. This probability map is converted into a ternary cost map by using two thresholds `free_thresh` and `occupied_thresh` such that a particular cell is considered as free if its p_{free} is greater than `free_thresh`, occupied if it is less than `occupied_thresh` and unknown otherwise.

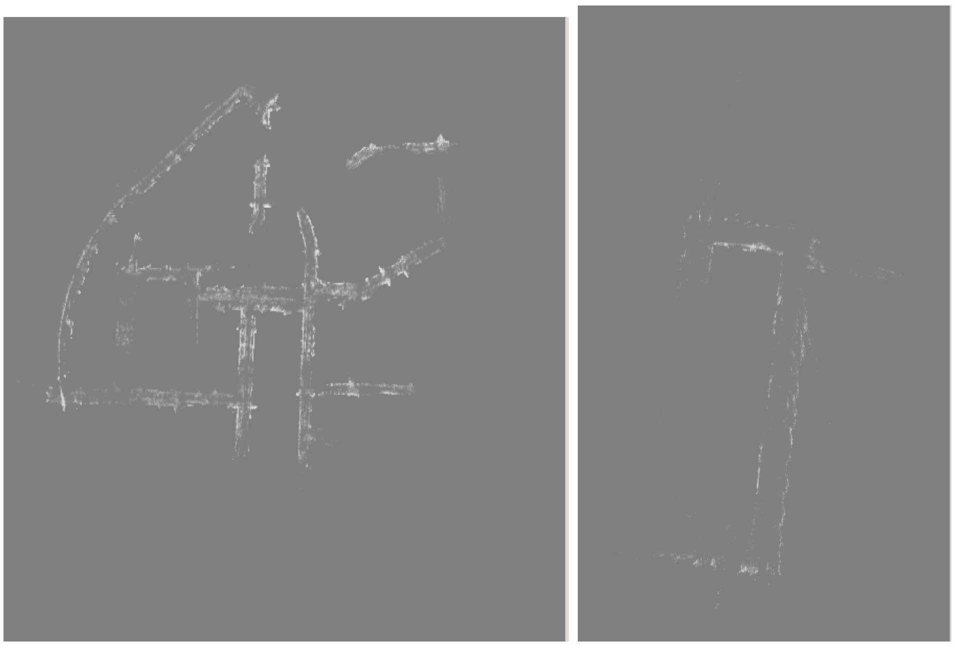

Figure 2: Probability maps before thresholding for to KITTI 00 sequence (left) and CSC first floor sequence (right) (Sec. 3.1.7)

3.1.6 Map visualization and robot navigation in Rviz

Since we do not have access to a real Turtlebot robot, we use simulation for this part instead. First, we create a virtual Turtlebot in an empty world using the Gazebo [19] simulator and then use adaptive Monte Carlo localization (AMCL) [20, 21] for navigation and Rviz [22] for visualization. Following commands are used for running these nodes:

```
roslaunch turtlebot_gazebo turtlebot_world.launch
     world_file:=/opt/ros/indigo/share/turtlebot_gazebo/worlds/empty.world
roslaunch turtlebot_gazebo amcl_demo.launch map_file:=./grid_map.yaml
roslaunch turtlebot_rviz_launchers view_navigation.launch
```

Fig. 3 shows screen shots of the map generated by the Python script and the navigation path produced by AMCL as visualized in Rviz.

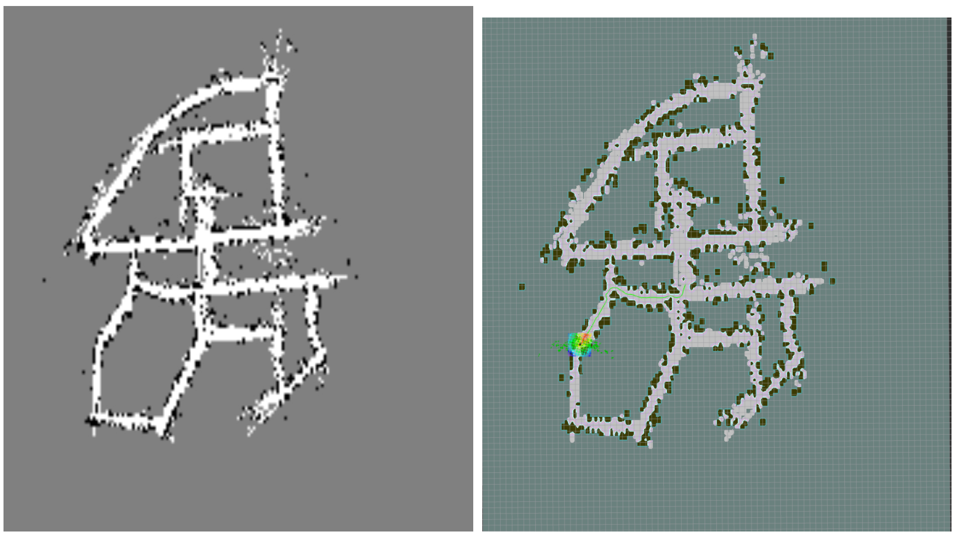

Figure 3: Map visualization and robot navigation

3.1.7 Evaluation and comparison of the result with that found in the literature

We could not find any existing work in literature where a 2D grid map was created using only 3D points generated by a visual SLAM method like ORB SLAM. Therefore, we needed LiDAR data for our datasets to compare with existing results which we also did not have. As a result, in order to evaluate our work, we recorded our own sequence using the calibrated camera on the first floor of the CSC building. Fig. 4 shows a couple of frames from this sequence along with the corresponding point cloud generated by ORB SLAM.

Next, we created the ground truth map for this sequence using the floor plan for CSC provided on the University website and used two measures for evaluation - an accuracy measure and a completeness score. The accuracy measure shows well our map agrees with the ground truth map while the completeness score shows how much of the true map our algorithm was able to generate regardless of it being correct or not. Before the two maps can be compared, they need to be aligned which can be done by estimating the homography between them. Then we use the following formulations to obtain the evaluation metrics:

$$\text{Completeness} = \frac{\text{number of known cells in the map}}{\text{total number of cells in the map}} \tag{2}$$

$$\text{Accuracy} = \frac{\text{number of correct known cells in the map}}{\text{total number of known cells in the map}} \tag{3}$$

3.2 Novel Components

In this part, we built on the results of the previous section to generate a grid map in real time and use it for navigation to follow the camera trajectory. This included the following steps:

3.2.1 Online variant

The Python code was converted into C++ and adapted to work in an incremental manner since it is not possible to process in real time all keyframes and map points obtained up to any given point in the sequence. This involved the creation of two ROS nodes by modifying the monocular ROS node `Examples/ROS/ORB_SLAM2/Mono`. The first node is called `Monopub` and publishes the pose of each keyframe whenever it is added to the map along with all map points visible in that keyframe. In addition, it detects whenever a loop closure is performed and then publishes all keyframes along with all map points added thus far. `Monopub` was also modified to accept input images from live cameras and ROS topics in addition to reading them from images on disk. Following four commands can be used to run this node on the KITTI 00 sequence, TUM `frg3_walking_halfsphere` sequence, live camera, and a ROS node publishing images to `/usb_cam/image_raw` topic:

```
rosrun ORB_SLAM2 Monopub Vocabulary/ORBvoc.txt Examples/Monocular/KITTI00-02.yaml ./KITTI/00 0
rosrun ORB_SLAM2 Monopub Vocabulary/ORBvoc.txt Examples/Monocular/TUM3.yaml
     ./TUM-RGBD/rgbd_dataset_freiburg3_walking_halfsphere
rosrun ORB_SLAM2 Monopub Vocabulary/ORBvoc.txt Examples/Monocular/mono.yaml 0
rosrun ORB_SLAM2 Monopub Vocabulary/ORBvoc.txt Examples/Monocular/demo_cam.yaml -1
     /usb_cam/image_raw
```

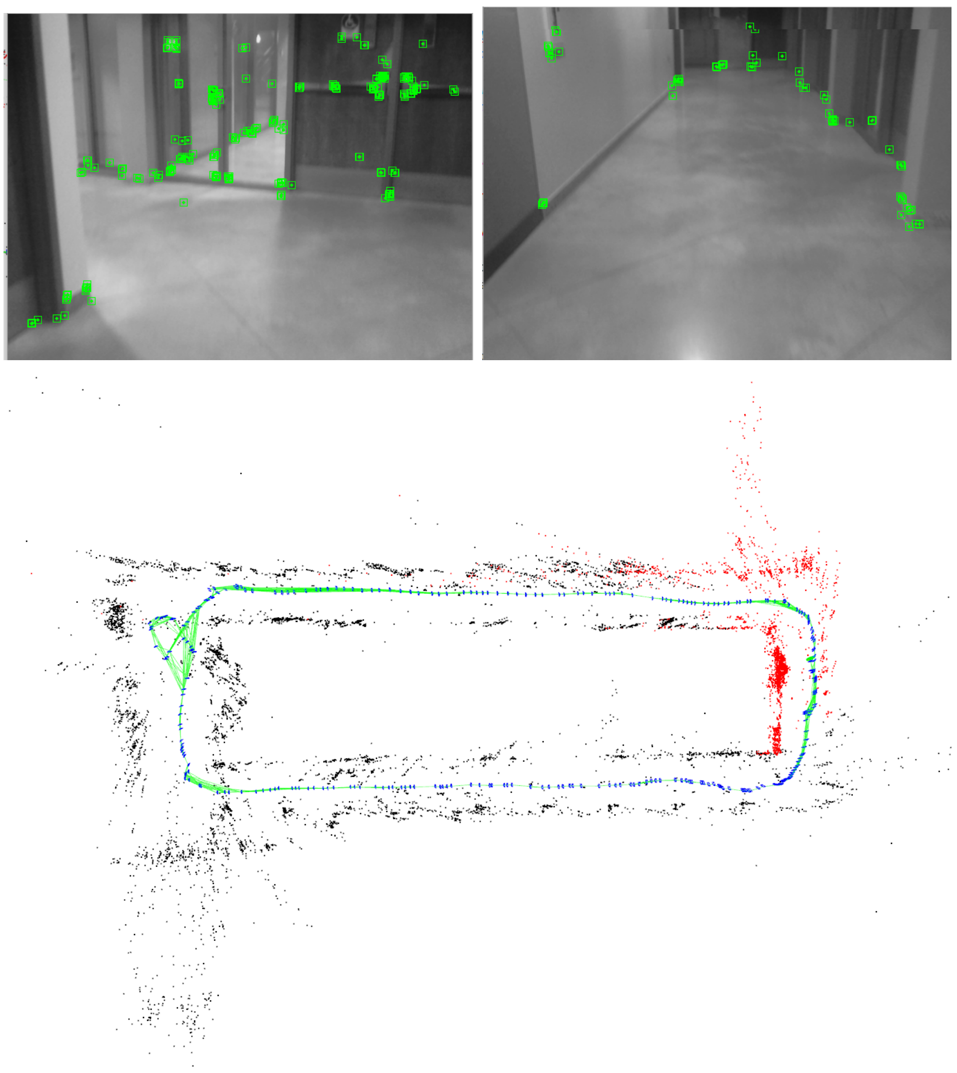

Figure 4: Two frames from the CSC sequence showing the ORB feature points (top row) and the 3D point cloud produced by running ORB SLAM on this sequence (bottom row)

The second node is called `Monosub` and subscribes to the pose data published by `Monopub`. When it receives a single keyframe, it processes it using the same method as in Sec. 3.1.5 though with several additional tricks that are described in the subsequent sections. The visit and occupied counters corresponding to each such keyframe are added together to build the map incrementally. Since the counters are updated independently for each keyframe and co-visibility between different keyframes are not taken into account, the map thus produced is only an approximation to the true map. In practice, however, it turned out to be quite accurate and more than sufficient for navigation.

When all keyframes are received after a loop closure, the counters are completely rest and recomputed using all the keyframes and map points received from the publisher simultaneously. This allows us to deal with the slight inaccuracies that may have accumulated over time. `Monopub` also provides the option to periodically publish all keyframes and points even if no loop closure is detected for scenarios where loop closures are too too few and far between. Map resetting is a time consuming process but needs to be performed rarely enough for it to not matter.

`Monosub` publishes the generated map and its meta data along with the camera trajectories to be used as navigation goals in AMCL and Rviz. Due to the many parameters that can be adjusted in `Monosub`, it accepts a large number of command line arguments that are listed below:

```
rosrun ORB_SLAM2 Monosub <scale_factor> <resize_factor> <cloud_max_x> <cloud_min_x>
    <cloud_max_z> <cloud_min_z> <free_thresh> <occupied_thresh> <use_local_counters>
    <visit_thresh> <use_gaussian_counters> <use_boundary_detection>
    <use_height_thresholding> <normal_thresh_deg> <canny_thresh> <enable_goal_publishing>
    <show_camera_location> <gauss_kernel_size>
```

For example, following commands can be used to run the two nodes on the KITTI 00 and the CSC first floor sequences respectively with different parameters fine tuned for each:

```
rosrun ORB_SLAM2 Monosub 10 1 29 -25 48 -12 0.55 0.50 1 5 1 0 1 75 350
rosrun ORB_SLAM2 Monosub 30 1 10 -10 22 -12 0.45 0.40 1 10 1 1 1 30 400
```

Note that the resolution for the first command is 1/10 m/cell and for second one is 1/30 m/cell. This node also allows the parameters to be adjusted at runtime through key presses whose details can be found in the `showGridMap` function of the `Monopub` source file. The code for both nodes in included with this report along with the modified version of ORB SLAM needed to run them. The source files for the publisher and subscriber are called
`Examples/ROS/ORB_SLAM2/src/ros_mono_pub.cc`
and
`Examples/ROS/ORB_SLAM2/src/ros_mono_sub.cc`
respectively.

3.2.2 Local and global counters

The idea of having local and global counters arises from an issue with the simple 2D projection of 3D points into the XZ plane which is used as the first step in ray casting. Let us consider a scenario in a single keyframe in which the projections of multiple points are co-linear in 2D (e.g. Fig. 5]). In this case, if we only use one global counter for generating the 2D map, we are misrepresenting the available information. For instance, the middle point in Fig. 5 is actually occupied but we are decreasing the occupied ratio for all the points that lie along this line by incrementing their visit counter. This means that, by only using global counters, we will end up replacing some occupied cells with free ones. Fig. 6 shows the comparison between using and not using local counters with our CSC sequence. As we can see, a lot of actually occupied points get replaced with free space if local counters are not used.

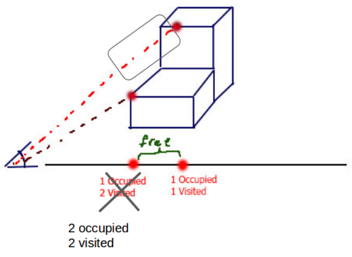

Figure 5: Issue of corrupting counters when points are co-linear in 2D

3.2.3 Visit thresholding

We have used the visit count of a cell as a measure of confidence of that cell such that we only make a grid cell occupied or empty if its total visit count is higher than a threshold. This allows us to remove outliers generated by wrong triangulations in ORB SLAM specially those that are a long distance away from the camera. Fig. 7 presents a comparison of maps generated using different visit thresholds. We have used a default visit threshold of 5 for our tests.

3.2.4 Height thresholding

Similar to the approach used by Goeddel et. al [6] (Sec. 2), only points that lie within a specific range of y-height above the XZ plane are accepted as obstacles. Therefore, we converted all points back to the camera coordinate frame and imposed a threshold on their height. Points whose y coordinates are below this threshold are assumed to lie on the floor/road and instead of

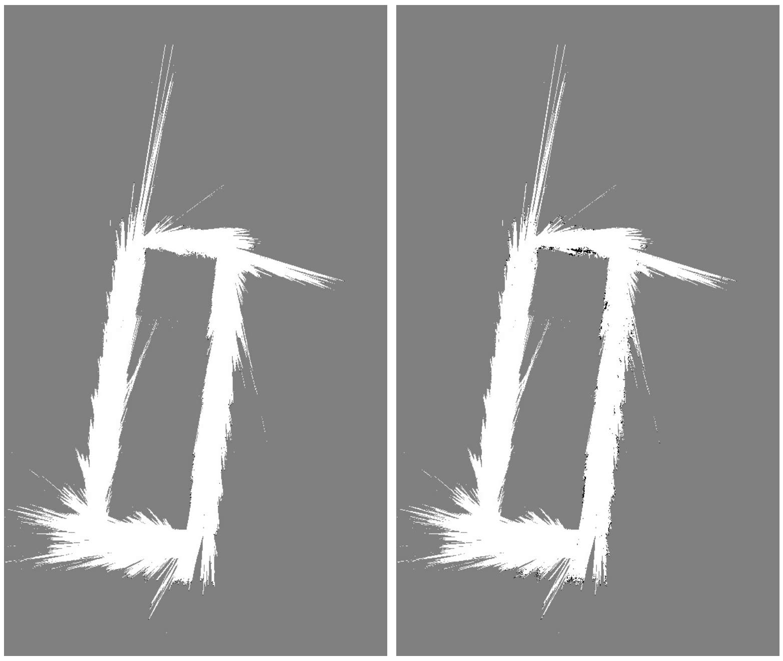
Figure 6: Effect of local counters on map quality. The map on the left was generated without local counters while the one on the right was generated with local counters.

removing them or counting the corresponding cells as occupied, we consider them as a free space by increasing only their visit counter. Fig. 8 shows a comparison of the maps obtained with and without height thresholding. As we can see, many of the occupied cells on the floor have been changed to free space after using height thresholding.

3.2.5 Gaussian smoothing of counters

Since space is not discrete, it is very likely that when two cells separated by a small distance are occupied, the cells between them are also occupied. These, however, might get marked as unknown leading to "holes" between otherwise contiguous regions of free or occupied cells. Inspired by the approach employed in the likelihood field model, we apply Gaussian smoothing to the local counters (both visit and occupied counters) to handle this issue. This causes the value in any given cell to influence its neighboring cells too so that any cells that were empty before would take on values similar to their nearby non-empty cells in both the counters. This in turn causes transitions within the probability map to become smoother such that the occupied and free cells affect their neighborhood. Fig. 9 illustrates the effect of different sizes of the Gaussian kernel used for smoothing. Based on these results, we have used a kernel size of 3 as the default for our tests.

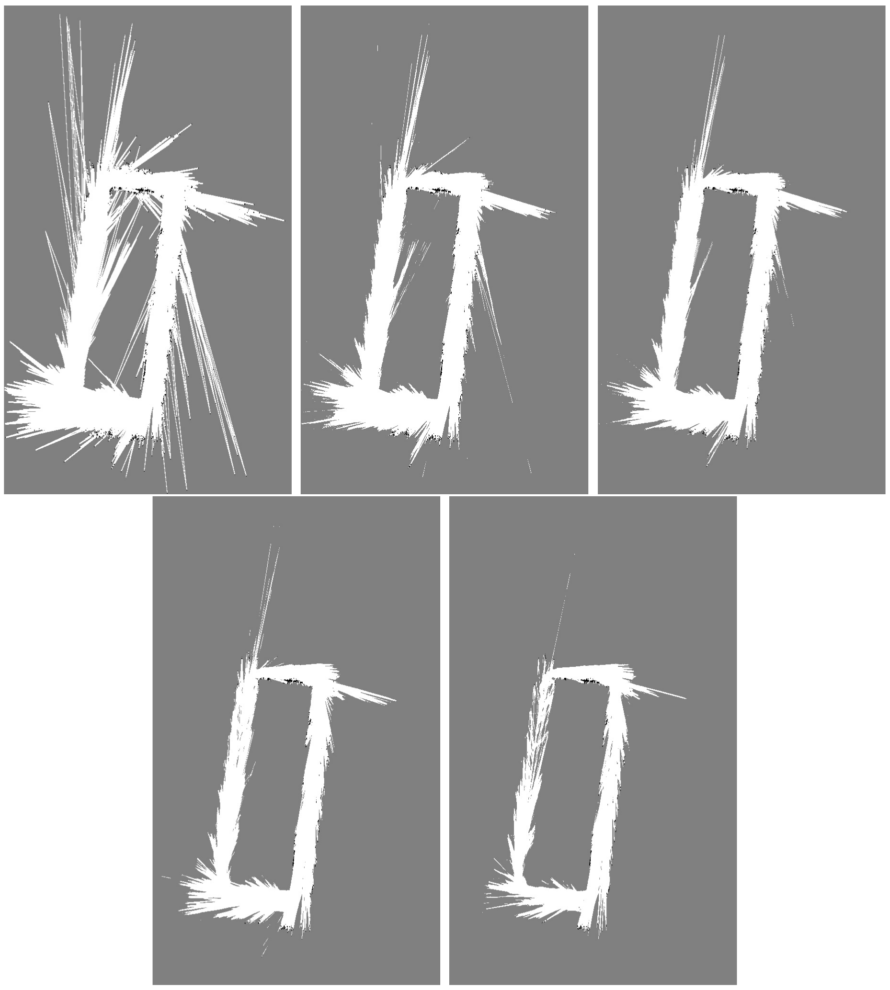

Figure 7: Comparison of different visit thresholds: from left to right and top to bottom - 0, 3, 5, 10, 20, 40

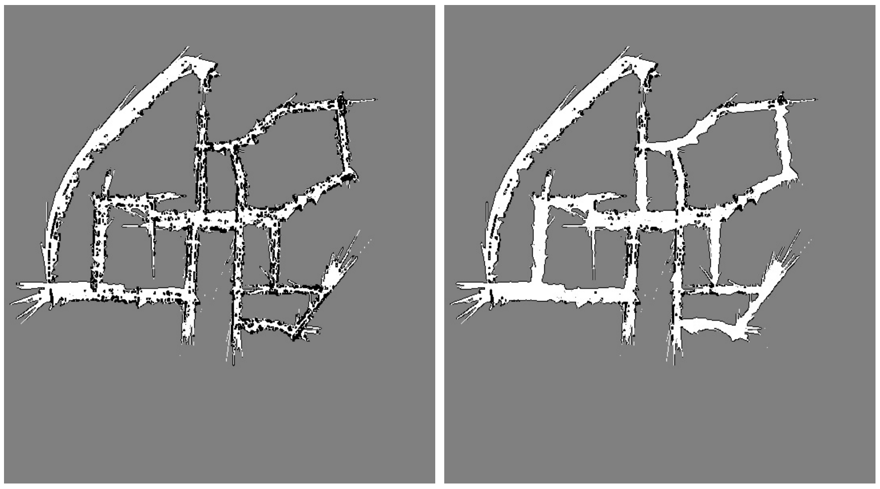

Figure 8: Effect of height thresholding on KITTI 00 sequence. In the left image, height thresholding has been disabled while the right one shows the result in the presence of the height thresholding

3.2.6 Canny boundary detection

Since most of the tricks we have employed are designed to *remove* false obstacles, an issue that is encountered is that too many of the *true* obstacles also get removed. We noted that most of these removed obstacles lay at the boundary between the free and unknown areas. Therefore, a simple way to bring them back is to mark the boundary pixels between the two areas as obstacles. We used Canny edge detection to find these boundary pixels and then set all pixels corresponding to the detected edges as 0 in the probability map to mark them as occupied. Fig. 10 shows the raw output of canny contour and the comparison between the maps obtained using different thresholds. It can be seen that this relatively simple approach not only produces a better map in terms of accuracy but also leads to better navigation paths that do not get too close to the unknown areas.

Fig. 11 shows the effect of the lower Canny threshold on the quality of the generated map with the upper threshold fixed to twice the lower one. We have used a default Canny threshold of 350 for our tests as it was found to eliminate most false edges in the interior of the free areas without removing any of the outer boundaries that we need.

3.2.7 Slope thresholding

This method is similar to the approach used by Goeddel et. al [6] (Sec. 2) where points that are estimated to lie on relatively horizontal planes are not regarded as obstacles. The plane that a point lies in is estimated by finding the two points nearest to it and computing the best fit plane containing these

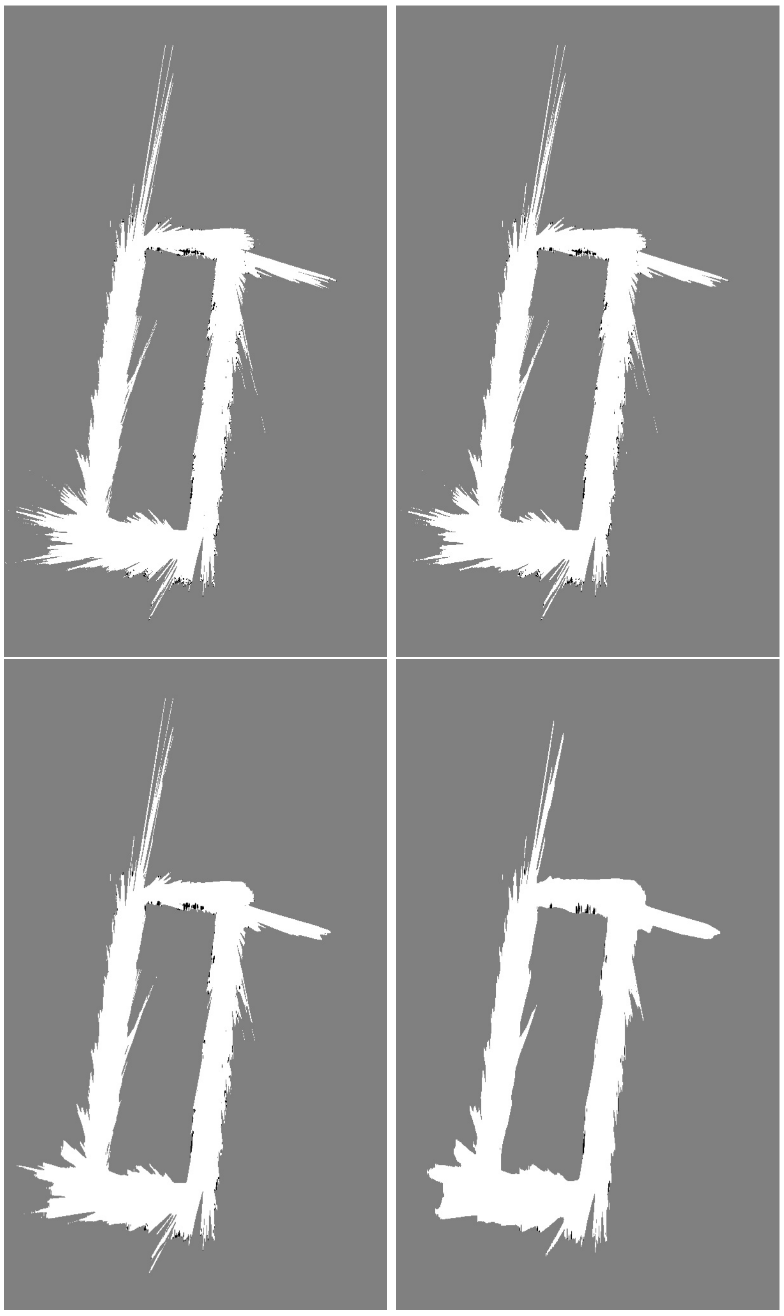

Figure 9: Maps generated using Gaussian smoothing of counters with different kernel sizes: left to right and top to bottom - 3, 5, 10 and 30

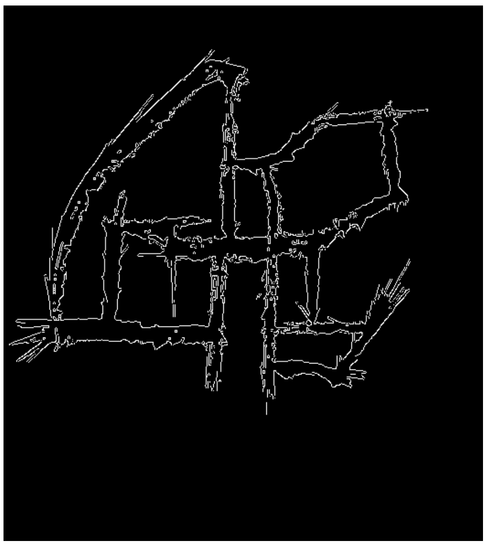

Figure 10: An example of finding contours

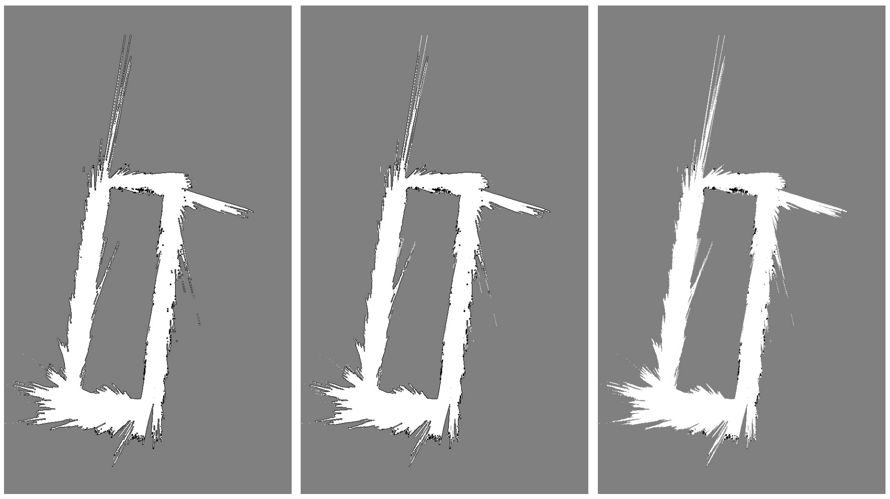

Figure 11: Comparison of maps generated by using different thresholds for Canny boundary detection: from left to right - 200, 350, 700

three points in the least squares sense. If the angle made by the normal to this this plane with the XZ plane exceeds a threshold, this plane is regarded as horizontal and the point is considered as corresponding to a free cell. A threshold of 75 degrees was found to give good results in our tests. The FLANN library [23] has been used to find the nearest points quickly and SVD was used to estimate the best fit plane.

4 Results

The main results with real time navigation have already been shown in the demo during the project presentation. Fig. 12 presents a couple of screen shots from that demo showing the grid map for KITTI 00 sequence and its navigation result in Rviz. Fig. 13 shows these results for the CSC sequence.

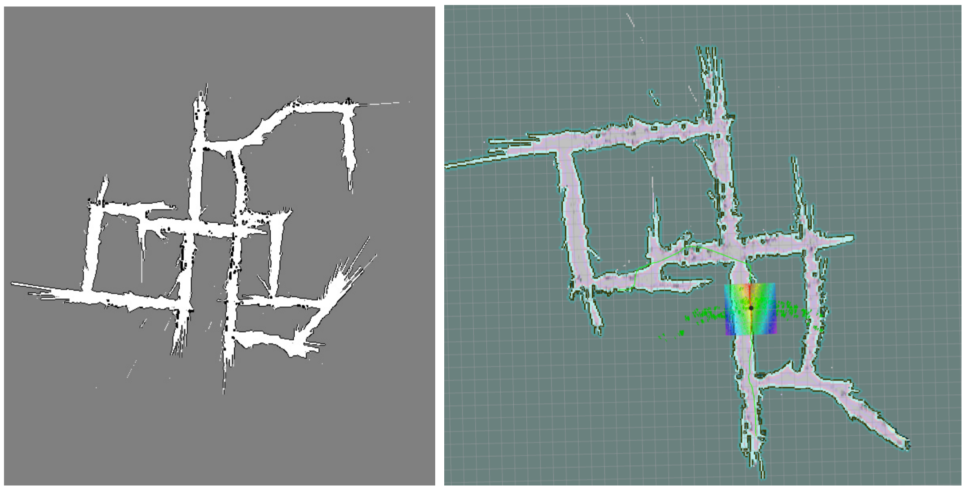

Figure 12: Online occupancy grid map and navigation on KITTI OO

We had to modify the launch files used in Sec. 3.1.6 to get them working with the online map and goals being published by `Monosub`. The modified launch files are included in the ORB SLAM root as `amcl_demo.launch` and `view_navigation.launch`. Following commands can be used to get the automatic navigation running:

```
roslaunch turtlebot_gazebo turtlebot_world.launch
    world_file:=/opt/ros/indigo/share/turtlebot_gazebo/worlds/empty.world
roslaunch amcl_demo.launch
roslaunch view_navigation.launch
<run Monopub>
<run Monosub with enable_goal_publishing set to 1 for automatic goal setting and 0 for manual
    goal selection in Rviz>
```

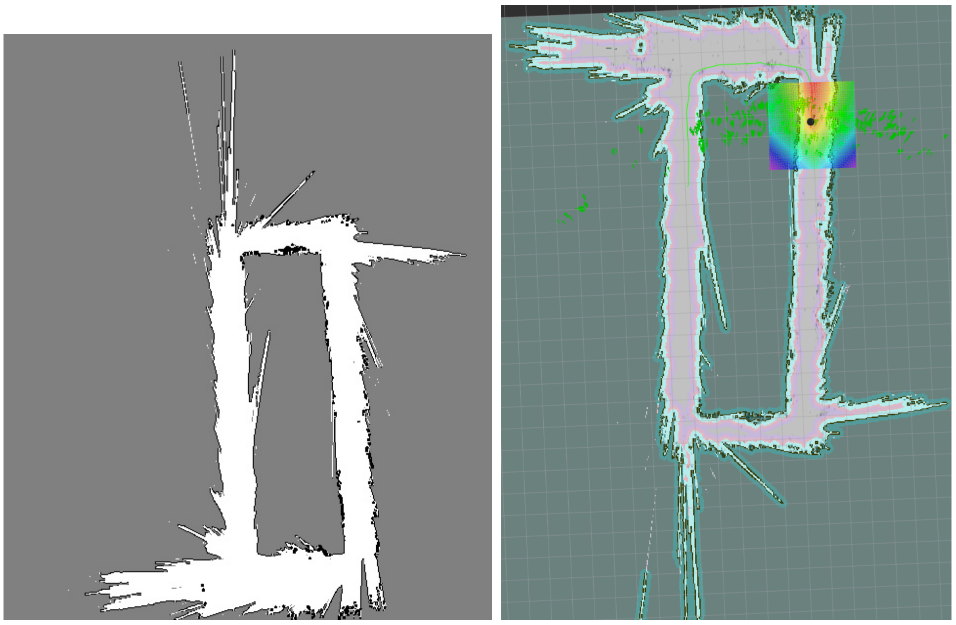
Figure 13: Online occupancy grid map and navigation on the CSC sequence

5 Conclusion

In this project, we investigated different ways of generating a 2D occupancy grid map from the sparse 3D map produced by ORB-SLAM. We experimented with several tricks like local counters, visit thresholding, Gaussian smoothing of counters, Canny boundary detection, height thresholding and slope thresholding to find the combination that produced the best map. We also used the resultant map for navigation of a simulated Turtlebot and found it to work quite well.

References

[1] C. Cadena, L. Carlone, H. Carrillo, Y. Latif, D. Scaramuzza, J. Neira, I. Reid, and J. J. Leonard, "Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age," *Trans. Rob.*, vol. 32, no. 6, pp. 1309–1332, Dec. 2016. [Online]. Available: https://doi.org/10.1109/TRO.2016.2624754

[2] M. J. M. M. Mur-Artal, Raúl and J. D. Tardós, "ORB-SLAM: a versatile and accurate monocular SLAM system," *IEEE Transactions on Robotics*, vol. 31, no. 5, pp. 1147–1163, 2015.

[3] R. Mur-Artal and J. D. Tardós, "ORB-SLAM2: An open-source SLAM system for monocular, stereo and RGB-D cameras," *arXiv preprint arXiv:1610.06475*, 2016.

[4] A. Elfes, "Occupancy grids: A probabilistic framework for robot perception and navigation," Ph.D. dissertation, Pittsburgh, PA, USA, 1989, aAI9006205.

[5] M. Quigley, K. Conley, B. P. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "ROS: an open-source Robot Operating System," in *ICRA Workshop on Open Source Software*, 2009.

[6] R. Goeddel, C. Kershaw, J. Serafin, and E. Olson, "Flat2d: Fast localization from approximate transformation into 2d," in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Oct 2016, pp. 1932–1939.

[7] J. Huesman, "Converting 3D Point Cloud Data into 2D Occupancy Grids suitable for Robot Applications," Online: https://library.ndsu.edu/repository/handle/10365/25535.

[8] A. Beutel, T. Mølhave, and P. K. Agarwal, "Natural neighbor interpolation based grid dem construction using a gpu," in *Proceedings of the 18th SIGSPATIAL International Conference on Advances in Geographic Information Systems*, ser. GIS '10. New York, NY, USA: ACM, 2010, pp. 172–181. [Online]. Available: http://doi.acm.org/10.1145/1869790.1869817

[9] P. Fankhauser and M. Hutter, *A Universal Grid Map Library: Implementation and Use Case for Rough Terrain Navigation*. Cham: Springer International Publishing, 2016, pp. 99–120. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-26054-9_5

[10] S. Thrun, "Learning occupancy grid maps with forward sensor models," *Autonomous Robots*, vol. 15, no. 2, pp. 111–127, 2003.

[11] A. M. Santana, K. R. Aires, R. M. Veras, and A. A. Medeiros, "An approach for 2d visual occupancy grid map using monocular vision," *Electronic Notes in Theoretical Computer Science*, vol. 281, pp. 175 – 191, 2011.

[12] P. Bovbel and T. Foote, "pointcloud_to_laserscan," Online: http://wiki.ros.org/pointcloud_to_laserscan.

[13] G. Grisetti, C. Stachniss, and W. Burgard, "Improved techniques for grid mapping with rao-blackwellized particle filters," *IEEE Transactions on Robotics*, vol. 23, no. 1, pp. 34–46, Feb 2007.

[14] J. M. M. M. Raul Mur-Artal, Juan D. Tardos and D. Galvez-Lopez, “Orb-slam2,” Online: https://github.com/raulmur/ORB_SLAM2.

[15] “How to calibrate a monocular camera,” Online: http://wiki.ros.org/camera_calibration/Tutorials/MonocularCalibration.

[16] A. Geiger, P. Lenz, and R. Urtasun, “Are we ready for autonomous driving? the kitti vision benchmark suite,” in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.

[17] J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. Cremers, “A benchmark for the evaluation of rgb-d slam systems,” in *Proc. of the International Conference on Intelligent Robot Systems (IROS)*, Oct. 2012.

[18] “The bresenham line-drawing algorithm,” Online: https://www.cs.helsinki.fi/group/goa/mallinnus/lines/bresenh.html.

[19] “gazebo,” Online: http://wiki.ros.org/gazebo.

[20] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents).* The MIT Press, 2005.

[21] “amcl,” Online: http://wiki.ros.org/amcl.

[22] “rviz,” Online: http://wiki.ros.org/rviz.

[23] M. Muja and D. G. Lowe, “Fast approximate nearest neighbors with automatic algorithm configuration,” in *International Conference on Computer Vision Theory and Application VISSAPP’09).* INSTICC Press, 2009, pp. 331–340.