Visual Simultaneous Location and Mapping for Autonomous Personal Mobility Vehicles

Wee Yan Ru Mervyn

Abstract— **Autonomous vehicles are becoming a part of our everyday on road commute. In addition, the development of personal mobility vehicles such as autonomous scooters and wheelchairs are helping us to easily traverse the first and last miles of our journeys for a complete coverage of our trips. Such personal mobile platforms can be vital for long and narrow passages at airports, hospitals and departmental stores, among many other indoor-outdoor places which are inaccessible by a car. We propose to retrofit a motorized wheelchair with a suite of sensors used for autonomous operations.**

I. INTRODUCTION

The interest in self driving cars has risen considerably over the past decade. In order to augment the benefits of autonomous vehicles, there is consideration to develop the use of autonomous personal mobility vehicles such as scooters and wheelchairs in order to cater to the first and last miles of a commuter's journey. These personal mobility vehicles will be able to transport commuters within areas that self driving cars are unable to reach. These include traveling within airports, hospitals and outdoor areas between buildings.

Currently, at the Future Mobility lab at the Singapore MIT Alliance for Research and Technology, there are already existing platforms that have transformed motorised wheelchairs into autonomous personal mobility vehicles. These motorised wheelchairs have been outfitted with LIDAR sensors for Simultaneous Location And Mapping (SLAM) in order to attain robot localization.

As further development, this project focuses on the use of a camera to perform visual SLAM in order to replace the LIDAR sensors. Using a camera for visual SLAM provides two advantages. Firstly, visual SLAM can be achieved by using a simple webcam, which is much cheaper than a LIDAR sensor. Secondly, it reduces the amount of effort needed to map out the environment. Currently, the system is set up such that the mapping process requires manual loop closure. After mapping the desired area, researchers need to go back to the lab and manually indicate the starting and ending points on the map so that the software may perform loop closure. With visual SLAM, the loop closure is performed automatically, thus simplifying the process and reducing possible errors that may arise from performing it manually.

*This work was not supported by any organization

Wee Yan Ru Mervyn is with Faculty of Electrical Engineering, School of Computing, National University of Singapore. mervyn@u.nus.edu

II. RELATED WORK

There are generally two approaches in visual SLAM feature based methods and direct methods. Feature based methods start with extracting invariant features from each camera image. Feature matching is performed using succeeding frames. Camera motion and structure is then recovered through epipolar geometry and then refined through reprojection error minimization[1].

The other approach is the direct method, recovers camera motion and structure directly from measurable image quantities (such as image brightness) from pixels in the image[2]. This approach uses all the information in the image, as opposed to feature based methods that discards information that does not produce the desired features[3].

Typically, the computational cost of feature based methods comes from the fact that feature extraction and matching needs to be performed at every frame. For direct methods, the main computational cost comes from requiring a semi-dense or dense reconstruction of the environment.

Much research has gone into increasing the computational efficiency, accuracy, and robustness of these visual SLAM methods so that visual SLAM may be achieved in real time. Due to advances in visual SLAM, the latest visual SLAM algorithms are even able to run in real time on smart phones[1].

A few noteworthy advances in the field include semi-direct visual odometry, LSD-SLAM[3], and ORB-SLAM[4]. Even though semi-direct visual odometry is not strictly a visual SLAM technique because it does not perform loop closure, it is mentioned because it is recent work that provides accurate results.

Semi-direct visual odometry combines direct and feature based methods. A sparse image alignment algorithm, which is a direct method, is used to estimate camera motion in between frames by minimizing the photometric error of features lying on intensity corners and edges. After feature correspondence has been achieved through Bayesian depth estimation, bundle adjustment is then used to optimize the structure and camera pose estimation to achieve high accuracy[1].

An advantage of semi-direct visual odometry is its computational efficiency. Semi-direct visual odometry excels at computational efficiency for three reasons: a parallel thread is used to extract features from selected keyframes, the novel sparse image alignment technique allows feature mapping to be accomplished robustly and quickly, and unlike other direct methods, it requires simply a sparse reconstruction of the environment[1].

LSD-SLAM is a direct method that provided two key novel ideas over existing direct methods at that time. It introduced a direct method to align two keyframes while detecting scale-drift, and a new probabilistic approach to incorporate noise from estimated depth maps into tracking.

There are two noteworthy mentions of LSD-SLAM. One is that it is able to maintain and track a global map of the environment, which was a step forward from other direct methods that focused on visual odometry[3]. The other is that it is fast enough to be run in real time without GPU acceleration[4].

ORB-SLAM is the latest feature based SLAM technique. It combines several key ideas from previous work, including Parallel Tracking And Mapping, place recognition research done by Galvez-Lopez and Tardos, scale aware loop closing by Strasdat et. al, and the use of covisibility information for large scale operation[4]. Feature based methods typically use bundle adjustment in order to obtain accurate motion estimation[1], which is computationally expensive. ORB-SLAM improves on this by performing reducing the number of edges in the graph that would be traversed by bundle adjustment so as to reduce computational cost. ORB-SLAM also introduces a survival of the fittest approach to map point and keyframe selection. This approach relaxes the criteria for keyframes so that keyframes are easily selected, but also liberally removes redundant keyframes later[4].

The combination of novel features and key ideas from several other works makes ORB-SLAM the most reliable and complete solution for monocular SLAM thus far[4].

III. ORB-SLAM

ORB-SLAM has two components that represent information regarding the environment. It has a map component that stores information about the environment, and a place recognition component which is used to determine if the camera has returned to a previously visited location.

It consists of three threads that run in parallel - tracking, local mapping and loop closing. These threads extract information from images of the environment and update the map and place recognition components.

A. Map

In ORB-SLAM, the map of an environment is represented by map points and graph of keyframes. Map points are points in 3D space associated with an ORB descriptor that was detected by ORB-SLAM. Keyframes are a subset of all the frames captured by the camera. Each keyframe will have a list of associated map points.

ORB-SLAM constructs a covisibility graph to keep track of how keyframes are related to each other. The covisibility graph is an undirected weighted graph whose nodes are keyframes. Two keyframes are connected if they both detect the same map points. The weight of the edge will be the number of common map points[4]. In essence, the covisibility graph keeps track of which keyframes share map points.

In addition, ORB-SLAM uses a spanning tree to build an Essential Graph, which consists of the same nodes, but with a reduced number of edges. This helps to lessen the computational burden of loop closure. In short, the covisibility graph and Essential Graph, which are composed of map points and keyframes, contain information about the environment seen by the camera.

In summary, the components that need to be saved are the map and recognition database components, where the map contains map points and a graph of keyframes.

B. Place Recognition

The place recognition component consists of a visual vocabulary and a recognition database. The place recognition's function is to aid in recognizing when the system has returned to a location that it has been to before.

The visual vocabulary is a bag of visual words that is pre-generated and loaded when ORB-SLAM initializes.

The recognition database contains an inverted index, which takes in a visual word query and returns the corresponding keyframes that contains that visual word[4].

C. Tracking

The tracking thread uses each frame to localize the camera, and selects which frames are suitable to be keyframes for insertion.

In the localization stage, the thread will compute ORB descriptors in each frame, then attempt to predict the new camera pose based on the camera pose from the previous image. If localization is lost, the frame is converted into a bag of words so that the recognition database in the place recognition component may be queried to identify a list of possible keyframes in order to establish global relocalization. The estimated camera pose will then be optimized.

For the keyframe insertion stage, the idea is to perform keyframe insertion quickly. This allows tracking to be able to handle difficult camera movements better, such as rotations. The local mapping thread will then later remove redundant keyframes.

D. Mapping

The local mapping thread processes these new keyframes and performs local bundle adjustment to achieve an optimal reconstruction in the surroundings of the camera pose[4]. It also exerts a strict policy on which keyframes can be kept, and liberally removes redundant keyframes.

When a keyframe is inserted, the local mapping thread will update the covisibility graph. With every new keyframe, a new node is inserted into the graph and edges are established with other nodes. The spanning tree that the Essential Graph is updated by connected the new keyframe with the keyframe that shares the most number of map points with it. Finally, a bag of words representation is computed for the keyframe, which will be used to perform data association with new map points.

After keyframe insertion, recently created map points will be reviewed by a policy that will decide if they should be deleted, and new map points would be created from ORB features.

The mapping thread will then run local bundle adjustment to optimize the current keyframe, its neighbouring keyframes, and the complete set of map points observed by these keyframes. Finally, keyframes will be deleted if they are redundant.

E. Loop Closing

The loop closing thread checks each keyframe to detect loop closure. If a loop has been detected, a similarity transform will be computed which will provide information regarding how much drift has occurred within the loop. Both sides of the loop would be aligned and replicated points would be merged. An optimization will then be conducted over map component so that it will be globally consistent. This optimization is performed over the Essential Graph rather than the covisibility graph in order to reduce computational complexity.

IV. Experimental Approach

Since the focus of the project is to adopt visual SLAM for a motorised wheelchair, an initial step would be to create a map of the desired environment using a visual SLAM system.

Literature suggests that ORB-SLAM is the state-of-the-art approach to visual SLAM, thus ORB-SLAM was the most ideal SLAM technique to be implemented on the motorised scooter.

The original code released to the open source community by the authors of ORB-SLAM did not come with the functionality to save and load a mapped environment. This meant that after mapping the environment, once the program was closed, the map would be lost. When ORB-SLAM was run again, a new map would have to be created. Obviously, this was inadequate. Ideally, a map would be created once, which would subsequently be reused for localization.

The initial approach in saving the map was to use ROS to rosbag record the images that were captured by the camera during the mapping stage. Subsequently, these images would be played back to the ORB-SLAM system to recreate the lost map. Essentially, this approach was equivalent to mapping out UTown each time, but by using a recorded set of images rather than having to manually walk around UTown.

However, this approach led to abysmal results. When the images were played back to ORB-SLAM, ORB-SLAM would often lose localization or fail to perform loop closure even though it had done it on the first time using the exact same set of images.

This gave way to the second approach where ORB-SLAM's place recognition and map components would be saved and restored.

The boost serialization library was used to save and restore these components to and from a binary file. A crucial benefit of the boost serialization library was that it had the object tracking feature. If there were two pointers pointing to the same object, upon restoration, the object tracking feature allowed the boost serialization library to intelligently create only one object, and have the two pointers point to it again. This was immensely useful in reconstructing the map component because there were many lists of pointers that pointed to the same objects.

The boost serialization library was successfully used to save and load the map. Multiple mapping trials showed that the map could be recreated when ORB-SLAM was restarted. This was a huge step forward in obtaining a usable map of UTown.

The code can be found `https://github.com/mervynwee/ORB_SLAM2` under the `save_load` branch.

V. Experimental Setup

University Town (UTown) in the National University of Singapore served as the testing environment for this project. Previously, a map of UTown had been created using the LIDAR sensors, so the immediate focus was to create a map of UTown using visual SLAM.

The first thing to do was to calibrate the camera to correct for distortion. This project uses a Logitech C270 webcam.

Camera calibration is done in order to correct for lens distortion and to calculate the intrinsic parameters of the camera. There are two kinds of lens distortion, radial distortion (refer Fig. 1 and tangential distortion (refer Fig. 2).

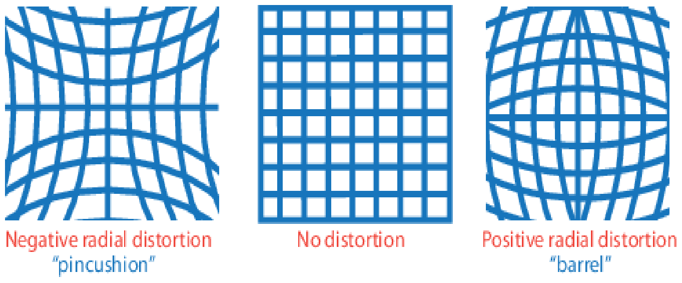


Fig. 1. Radial Distortion

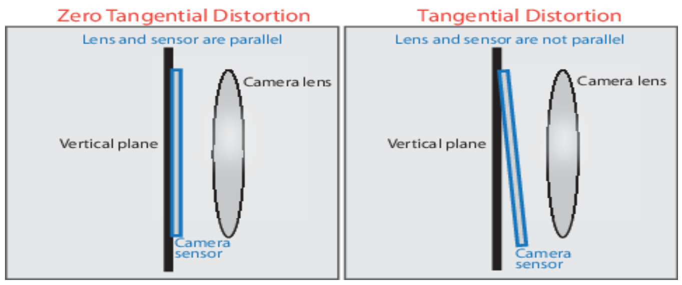


Fig. 2. Tangential Distortion

For radial distortion, a pixel at position x, y can be corrected using the following formula[opencv]:

$$x' = x(1 + k_1 r^2 + k_2 r^4 + k_3 r^6)$$
$$y' = y(1 + k_1 r^2 + k_2 r^4 + k_3 r^6)$$

For tangential distortion, a pixel at position x, y can be corrected using the following formula[opencv]:

$$x' = x + [2p_1xy + p_2(r^2 + 2x^2)]$$
$$y' = y + [p_1(r^2 + 2y^2) + 2p_2xy]$$

These formulas indicate that there are 5 coefficients, k_1, k_2, k_3, k_4, k_5 that need to be calculated to correct for distortion.

The intrinsic parameters of the camera are its focal length and its optical center, in pixels. The intrinsic parameters are used to where an object in 3D space captured by the camera would be in terms of pixel coordinates in the captured image. The relationship is given by:

$$\begin{bmatrix} x \\ y \\ w \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}$$

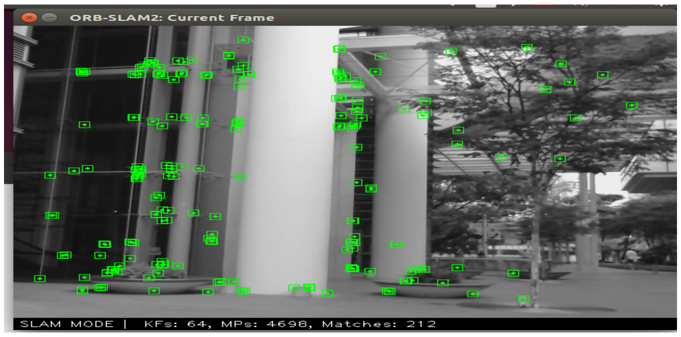
where f_x, f_y are the focal lengths and c_x, c_y is the optical center.

These parameters can be calculated using MATLAB or OpenCV. Once they have been calculated, the camera is then ready for the mapping process.

VI. Experimental Results

Multiple mapping trials were conducted at Town Plaza in the day to map an area of roughly 20m by 10m. There were key observations that were in agreement with the literature review.

The first observation was that ORB-SLAM had problems dealing with rotations. It became easy to lose localization during mapping when the environment forced the user to turn a sharp ninety degrees around a corner. The second observation was that ORB-SLAM had difficulties when a large part of the image was covered by a plain wall or the floor.

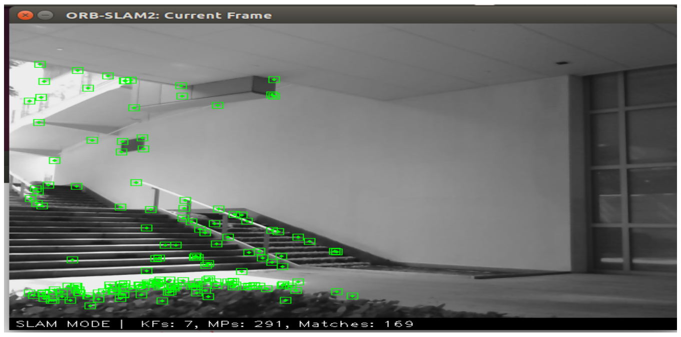


```
SLAM MODE |  KFs: 64, MPs: 4698, Matches: 212
```

Fig. 3. White Pillar

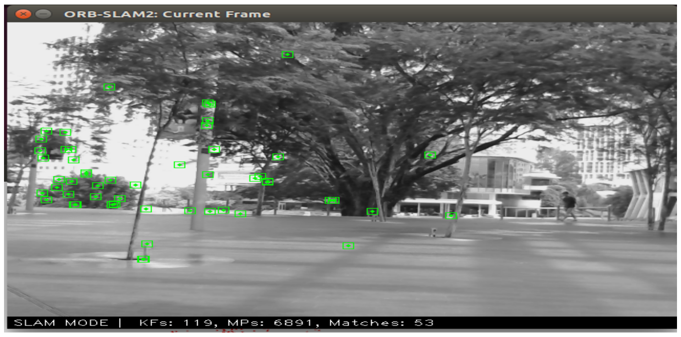
As can be seen from the images, there were no ORB descriptors detected on the white pillar, the white wall, and the floor, while ORB descriptors could be found in other parts of the image. The lack of features in large parts of the image made localization easy to fail in these locations.

This was especially true for Fig. 5 due to the fact that the path required a hard right turn at that exact location. The combination of lack of features in the image and a strong rotation made it difficult to map that particular spot.



```
SLAM MODE |  KFs: 7, MPs: 291, Matches: 169
```

Fig. 4. White Wall



```
SLAM MODE |  KFs: 119, MPs: 6891, Matches: 53
```

Fig. 5. Plain Floor

These observations were not unexpected. In the original paper, the authors faced the same problems with strong rotations and featureless environments when evaluating ORB-SLAM against the KITTI and TUM RGB-D benchmarks[4].

These results suggest that there might be difficulties in mapping and localization in indoor areas resembling corridors and long passageways. It is not uncommon to find long corridors straddling white walls and plain floors inside buildings. Furthermore, many buildings have corridors that are connected other passageways by sharp turns. These sharp turns would also affect the performance of mapping and localization.

With regards to outdoor areas, in order to circumvent problems with rotation, the availability of space may make it possible to map the environment with gently curving paths instead.

VII. Future Work

For future work, there are improvements to be made in both the saving and loading feature, and increasing the robustness of the mapping process.

An obstacle that currently exists is that the system is unable to relocalize based on the loaded map. Despite having loaded the map, ORB-SLAM was unable to identify the locations at which it had been to before. More work needs to be done to fully achieve the saving and loading feature.

To improve the robustness of the mapping process, it would be good to investigate the minimum number of features needed per frame in order to maintain localization and to determine how robust the ORB-SLAM system is in recovering from localization failures. For example, one way would be to determine the maximum number of frames that ORB-SLAM is able to miss before losing localization.

Based on these analyses, there may be some heuristics that could be developed to guide better mapping processes. For example, positioning the camera to face the ceiling might yield better results in terms of the availability of features. Another approach might be combining wheel odometry of the wheelchair with the ORB-SLAM camera pose estimation to help the system relocalize in the event that it loses localization. Knowing the limits of the system would provide better information on how these workarounds could be better designed to support the weaknesses the system.

VIII. CONCLUSION

This report reviews the latest visual SLAM techniques that have been developed, and pinpointed ORB-SLAM as the most ideal solution that exists at the moment. For this reason, it was chosen to be integrated into a motorised wheelchair so that the motorised wheelchair may be converted into an autonomous personal mobility vehicle. The report also discusses the work that has been accomplished, and offers directions to where future work will lie.

IX. ACKNOWLEDGMENTS

I would like to thank A/P Gary Tan from School of Computing, A/P Marcelo Ang from Department of Mechanical Engineering, and Dr. Malika Meghjani from Singapore MIT Alliance for Technology and Research for their support.

REFERENCES

[1] Christian Forster, Zichao Zhang, Michael Gassner, Manuel Werlberger, and Davide Scaramuzza. Svo: Semidirect visual odometry for monocular and multicamera systems. *IEEE Transactions on Robotics*, 2016.

[2] Michal Irani and P Anandan. About direct methods. In *International Workshop on Vision Algorithms*, pages 267–277. Springer, 1999.

[3] Jakob Engel, Thomas Schöps, and Daniel Cremers. Lsd-slam: Large-scale direct monocular slam. In *European Conference on Computer Vision*, pages 834–849. Springer, 2014.

[4] Raul Mur-Artal, Jose Maria Martinez Montiel, and Juan D Tardos. Orb-slam: a versatile and accurate monocular slam system. *IEEE Transactions on Robotics*, 31(5):1147–1163, 2015.