Multi-Camera Mayan Project

Etsu Nakahara[1], Kaushik Ramasamy[2], Duke Lin[3]

University of California, San Diego
CSE 145 Project

[1]etnakaha@ucsd.edu
[2]kramasam@eng.ucsd.edu
[3]d3lin@ucsd.edu

1. Abstract

Archaeologists have been studying the buried Maya temples for many years through excavating tunnels around the temples. However, there are various issues with the hand-drawn documentation of the tunnels such as that it is time-consuming and hard to correctly capture the complex temple structures. To tackle this problem, this project works on creating a system to generate 3D models of the tunnels that can accurately document the temple structures within a reasonable amount of time. Originally, the team was developing a system using single camera to record the tunnel excavation. However, there are problems in accuracy of the generated 3D model due to the system's likeliness to lose track of itself while recording. We propose a dual-camera system that can solve this problem by increasing the accuracy of the generated 3D model through its wider field of view. Furthermore, the two cameras will reduce the problem of losing track.

2. Introduction

Remnants of Mayan civilization are being discovered by archaeologists as they dig extensive networks of tunnels into sites. The archaeologists document their journey and findings with hand drawn sketches and more recently, LiDAR. However, sketches are time consuming, prone to error, and do not truly depict the complexity of the network, while LiDAR requires significant coordination and time to do just one scan. Furthermore, LiDAR does not provide a real time result, which could cause a wasted trip if scans are incorrect. Therefore, it was proposed to use SLAM algorithms for mapping. The benefits of SLAM algorithm are that they do provide real-time results in a quick and easy manner to allow the user to know if a mistake was made and re-do a scan in the moment. However, while the SLAM approach meets the necessities of archaeology excavations, there was an issue with the accuracy.

We extend this proposal to the logical next step, multi-camera SLAM. Adding an additional camera not only increase accuracy but provide additional robustness for the system as a whole. We test three multi-camera SLAM algorithms on two Realsense ZR300's and two Kinect V2's.

As these tunnels have complex structures and tight turns, the single camera system loses track in these sudden turns due to drastic scene change and limited field of view. It fails to correlate and map points between different frames in these tight turns. As a result, it loses track and the system stops generating the 3D model resulting in incomplete models. One more issue faced by the single camera system is that the models are generated with small errors as pose estimation was done with just single camera and it may not be exactly same as the ground truth. Over the time, these small errors tends to build up and in the end it becomes enormous and results in drifting issue, as a result of which the models generated are not accurate. Sometimes the straight tunnel paths may appear drifted along a particular direction and may not appear straight anymore. These issues could be rectified using wider field of view and with multiple cameras. With multiple input feeds, the SLAM algorithm will be effectively able to estimate the pose with minimum or little error, as a result will generate more accurate and better 3D models.

RTAB-Map or Real-Time Appearance-Based Mapping uses RGB and depth data to create a graph whose nodes are the pose of the camera at a point in time and edges are the constraints between each pose. This graph is generated as the camera moves and is used to construct a point cloud in real time.

MultiCol SLAM is an extension of ORB-SLAM2 algorithm supporting multiple cameras at the same time. It tries to estimate the camera pose and relative motion between the cameras with input frames from multiple cameras. It can also work with any camera types be it fisheye , omnidirectional or perspective cameras. It introduces MultiKeyframe for handling keyframes from different cameras, hyper-graph model for multi-camera SLAM, multi-camera loop closing method, minimal solver for pose estimation and a different initialization method map using the essential matrix. It uses generic camera model as in (Scaramuzza et al., 2006, Urban et al., 2015) covering any type of cameras (fisheye, omni directional, perspective).

MCPTAM or Multi-Camera Parallel Tracking and Mapping is an extension of Parallel Tracking and Mapping (PTAM) SLAM algorithm, which maps the real world by estimating the position of the cameras and mapping the feature points in the environment. MCPTAM resolves some of the problem of PTAM with the wider field of view of the multi-camera system, and improves the quality of feature point capturing and pose estimation.

3. Multi-Camera SLAM Algorithms

3a. MCPTAM:

MCPTAM is extended from the single camera SLAM Algorithm, PTAM software. The algorithm of PTAM on which MCPTAM is based maps the real world by estimating the position of the cameras and mapping the feature points in the environment. MCPTAM is designed for multiple cameras that are rigidly connected together into a cluster. If connected together, MCPTAM allows any number of cameras, and any configurations of the cameras. Hence, this flexibility enables MCPTAM to achieve different missions in different environments. The algorithm is based on the ROS development environment. In our system, we use two Kinect V1 to run the MCPTAM algorithm.

Traditional single camera SLAM algorithm suffers from the aperture problem, "which is the inability to discern different types of motion when looking at a scene through a narrow FOV." MCPTAM eliminates this aperture problem with the large field of view of multiple cameras, and hence increase the robustness of tracking. Furthermore, multi-camera system allows MCPTAM to achieve high quality of feature point capture and pose estimation. Especially with overlapping field of view, feature points that don't match among different cameras could be discarded. The algorithm even allows non-overlapping field of view by its original extrinsic calibration method which would be explained later in the Milestone section.

Also, unlike the ordinary PTAM algorithm, MCPTAM system can utilize the overlap of the cameras' field of views to correctly estimate the cameras' motion. As a point feature is observed by the cameras in the cluster at the same time, the depth value of that point can be accurately triangulated based on the extrinsic calibration of the system. The algorithm still can function even without sufficient overlap of the cameras' field of view. Based on the extrinsic calibration and the motion of the system in relative of the surrounding environment, MCPTAM still can estimate the cameras' motion even without the overlap of field of view. Without the restriction to have overlaps between the cameras, MCPTAM allows a configuration that maximizes its collective field of view and improves the localization estimation.

3b. MultiCol SLAM:

It uses a body frame concept and calibrated cameras for pose estimation. Thus both intrinsic and extrinsic calibration parameters are calculated before hand. It uses the following transformation to map a 3D scene point to corresponding image point.

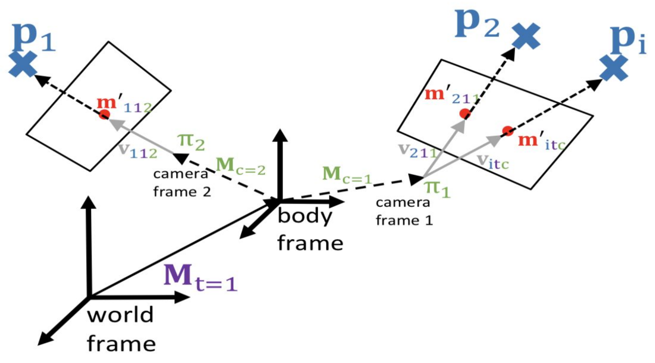


Figure 3.1: Depicting the body frame concept and Observed scene point p_i and its transformations.

$$\boxed{\mathbf{m}_{itc} = \pi_c^g(\mathbf{p}_{itc}) = \pi_c^g(\mathbf{M}_c^{-1}\mathbf{M}_t^{-1}\mathbf{p}_i)}$$

where p_{itc} is the scene point observed at camera c at time t, and the transformation is given by transformation matrices, Mc for the cth camera and Mt at time t , and π is the generic camera model introduced by [Scaramouzza et al 2006,Urban et al,2016]. m_{itc} is the projected point i in camera c at time t.

Algorithm Overview:

This algorithm requires slight overlap in the FOV between different cameras. The algorithm works in the following way, It has:

1. Tracking thread
2. Mapping thread
3. Relocalization thread
4. Loop closing thread

Tracking Thread:

It uses similar tracking method used in ORB-SLAM2 system. It uses image feature points for tracking using ORB feature extractor with multiple scale levels. The feature points are represented using ORB descriptor for efficient performance unlike the image patch matching method. Then it uses constant velocity model to calculate the current camera pose. It uses 3 point correspondences instead of the EPnP solver, which uses n point correspondences to estimate the initial camera pose. First the Essential Matrix for a single camera is calculated for different multi-camera system poses and and try the project the same point into different cameras and do bundle adjustment over all other observations between two multi-camera system poses, thus initializing the mapping more robustly.

Relocalization Thread:

Once the tracking fails, it kicks off the relocalization thread. This converts the frames into bag of words and queries the MultiKeyframe data base for closest match, and it tries to match the mapped points with respect to that particular MultiKeyframe entity. Instead of n points correspondence used in ORB-SLAM2 , this algorithm uses 3 points correspondence to estimate the current camera pose. If more than a particular number of points are retained, then relocalization is done, and tracking is started as usual.

Mapping Thread:

It runs in parallel to the tracking thread, where it tried to refine the estimated or already mapped points. It also removes the redundant map points and redundant keyframes if there is no significant change in the scene. Once the map points are updated , it is then fed to the tracking thread to track the refined points. It handles the

MultiKeyframe insertion and map point deletion. It also does triangulation of the new map points over multiple cameras. All the redundant and duplicate map points observed in multiple cameras are fused in this thread. Then it does local bundle adjustment and optimizes over poses and map points. Similar to ORB-SLAM, it does multi keyframe culling , i.e., if two multi keyframes shares more than 90% of the map points, then the key frame is deleted.

Loop closing thread:

It is the extended version of the loop closing thread introduced in ORB-SLAM2 , but with the support for fisheye and omni-directional cameras. It uses visual cues to identify the possible loop candidate. Possible similar MultiKeyFrame candidates are then queried from the database to compare with the current MultiKeyframe. Then similarity transformation is done to match it with the closest multikey frame match. Loop correction is done to all the Keyframes connected to it and to all the map points being observed by those keyframes. This helps in eliminating the error build up and also removes the drift issue which occurs due to error accumulation over the time.

Advantages over single camera ORB-SLAM:

It outperforms the ORB-SLAM2 in Keyframe accuracy, because of better estimation with multi-camera model and using multiple keyframes for pose estimation. It performs really well in map initialization because of it robustness. It also outperforms the ORB-SLAM algorithm in rotational and translational accuracy by significant margin. Thus yielding a better and more accurate 3D model addressing the drift issues. It also employs several performance improvements for achieving real-time results with multiple keyframes and tracking in multiple input feeds. It is able to achieve performance of 25fps.

3c. RTAB-Map:

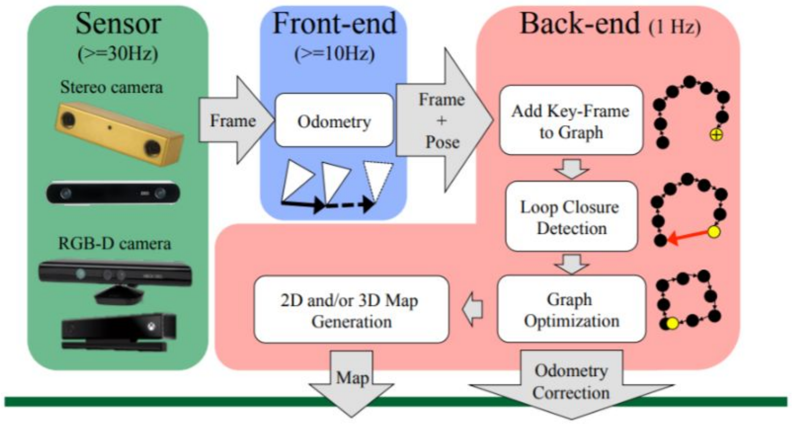


Figure 3.2: RTAB-Map Overview, from a RTAB-Map presentation [7]

RTAB-Map is a RGBD-SLAM approach meaning it can utilize just RGB and depth data to map. It is also graph-based approach where the nodes of the graph represent the pose of the camera at a point in time and edges represent the constraints between each pose. The graph is generated as the camera moves around and takes more pictures, and it helps construct a point cloud in real time. The node (or the pose) of the camera is generated from the features found in an image and is stored via the bag of words approach. As the camera moves, the features are checked repeatedly by a loop closure detection to see if the camera has moved to a new location or an old location. When the loop closure hypothesis is accepted, the node is completed, and a constraint to a new node is created. So the edge (or the constraint) is created when the hypothesis is accepted. Furthermore, after the hypothesis acceptance, the graph is optimized to minimize errors.

RTAB-Map uses visual odometry to determine the position and orientation of the cameras by repeatedly looking at captured frames and depth data. It first looks at 2D features found in a frame and compares it to other features found in recent frames. It also establishes the optical flow between consecutive frames by seeing where the image features match and overlap. To compare 2D features, the algorithm uses both 3D correspondences and reprojection to estimate the transformation between frames.

Loop closure detection is used to determine if a new image is of a new location or an old location. The algorithm uses the stored bag of word vectors of 2D features to quickly compare to see if there is a loop closure. If the algorithm has a strong confidence and accepts the hypothesis, then the location is set and the map is adjusted. Loop closure helps increase the precision of pose estimation because recognizing previous locations adds a level of robustness to a previous mapping and gives certainty when moving from an old location to a new one.

RTAB-Map uses Vertigo, a graph optimization, to reduce the amount of errors in the graph. Without a robust graph optimization in play, the occurrence of similar objects in different locations can lead to incorrect loop closure detections.

4. Milestones

We defined carefully identified the potential tasks required to bring up the systems and finally came with the following milestones to help us track and progress with the project. Initially we started with comparing the different cameras used in SLAM algorithm, but as we progressed through, the milestone got changed to comparing different SLAM algorithms. As comparing different SLAM versions would be really helpful instead of comparing just different camera systems.

Our Milestones are:

Milestone 1: Learn ROS to a level where we can record RGB and depth data and replaying through use of ROS and Kinect/Realsense

1. Setup ROS with Ubuntu.
2. Get live feed from cameras.

Milestone 2: Show a synchronized feed between the two cameras

1. Get live feed from two cameras.
2. Sync data from the 2 cameras.

Milestone 3: Produce point cloud using the multi-camera SLAM algorithm

1. Research and choose one of the following SLAM algorithms to implement with two kinects and two realsense.
    a. RTAB-Map
    b. MultiCol - ORBSLAM
    c. MCPTAM
2. Get the SLAM algorithm working so that it can produce point cloud based on the data received from the dual-camera system

Stretch Goal: Test our generated point cloud against the ground truth (obtained by currently working single camera SLAM algorithm) with Go-ICP script.

Milestone Modification:

Earlier we planned to test only two SLAM algorithms RTAB-Map , MultiCol ORBSLAM2 , but as the project progressed ,we decided to test one more SLAM algorithm (MCPTAM) as we had enough time and resource to do so. We dropped the plan to document the pros and cons of different camera systems as it was not that helpful in choosing the SLAM algorithms and for the whole system. Instead we decided to test one more SLAM algorithm (MCPTAM).

Milestones Status and Tasks Accomplished:

Deliverable	**TimeLine**	**Status**
1. Live Camera running with ROS showing RGBD data	Jan 27	Done

2. Two Live Cameras running with ROS showing RGBD data and sync status	Feb 7	Done
3. 3D model of a room (lab environment) a.Research on SLAM Algorithm b.Make SLAM Work **Stretch :** Test against ground truth, and document the result	Mar 14 Feb 21 Mar 10 Mar 14	1. RTAB-Map - Done 2. MultiCol ORB-SLAM2 - Model not accurate 3. MCPTAM - Issue with running algorithm
Final Video	Mar 19	Done
Final Report	Mar 22	Done

Milestone 1 : Learn ROS and Get feed with Cameras:

We did a brief study of the ROS and got familiarized with the ROS environment. We successfully demonstrated the live feed RGBD data from intel real sense ZR300 camera and Kinect v1 camera. We used Rviz to visualize the data feed from the cameras.

Team Members:	Real Sense - Duke and Kaushik
	Kinect - Etsu Nakahara

Milestone 2 : Get Synchronized Feed from two cameras

Since our system uses multiple cameras, we have to synchronize the feed from two cameras and feed the SLAM algorithm with frames which are closest in time. We utilized the approximate time filter for synchronizing the feeds which gives set of frames from two different topics , which are closest in time stamp. We faced some issue with synchronizing the feed,as the delay between the synchronized frames was increasing over time. We tried to explore the synchronizing queue sizes and experimented with different queue sizes for this problem. But it was not helpful , as it was not effectively affected by the queue sizes. Queue size had limited effect on the synchronization. Next we tried to investigate the Frame data. We found that higher the frame resolution and frame rate, delay was significant. This might be due to the bandwidth limitation of the connected port. Decreasing the frame rate and the frame resolution helped us solve this issue. Currently we use fisheye lens with 30fps and with frame resolution of 640x480.

Milestone 3 : Get SLAM working with system.Produce the Point Cloud

This milestone had three different sub-tasks within itself

1. Research SLAM algorithm which can support multiple cameras
2. Integrate the SLAM algorithm into the system
3. Make SLAM algorithm work

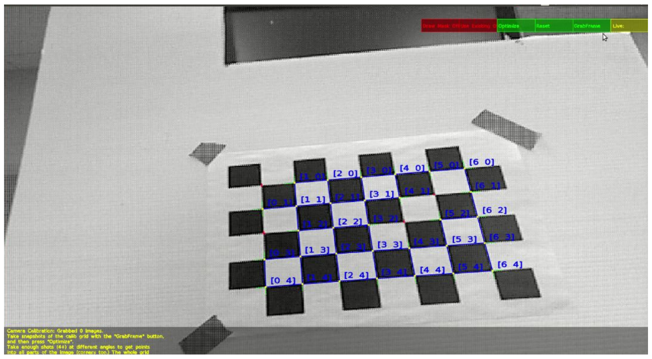
After the thorough study, we decided to pick up and work with 3 SLAM algorithms based on the previous results and support for multiple cameras. Each team member picked up a SLAM algorithm and worked with it.

1. MCPTAM - Etsu Nakahara
2. MultiCol ORBSLAM - Kaushik Ram
3. RTAB-Map - Duke and Ferrill

MCPTAM:

MCPTAM requires intrinsic calibration done with each cameras in the cluster and extrinsic calibration done with the whole cluster by using the customized calibrators. The calibration tools are provided as ROS launch files (and so is the file to run the MCPTAM algorithm). As we are using two Kinect V1 for MCPTAM, for this milestone, we calibrated the two Kinect V1 cameras with the provided intrinsic calibrator and the extrinsic calibrator.

By launching the ROS launch file of the intrinsic calibrator, the window for the calibration will be instantiated. By capturing key frames from different angles of a checkerboard, the calibrator will calculate the reprojection error and save the calibrated parameters to a .yaml file and the camera. As the Figure 1 shows, if the camera detects the corners in the checkerboard, the display of the calibrator will have blue indicator on the checkerboard. By pressing the "Grab KeyFrame" button or by pressing space, the calibrator will capture that specific keyframe. After repeating these processes with key frames taken from different angles of the checkerboard, by pressing the "Optimize" button, the calibrator will calculate the reprojection error based on the grabbed key frames. The reprojection error is shown in red line. If the error is small enough, then the calibrated parameters can be saved to a .yaml file and to the camera by pressing the "Save" button.



```
Camera Calibration: Grabbed 0 images.
Take snapshots of the calib grid with the "GrabFrame" button,
and then press "Optimize".
Take enough shots (4+) at different angles to get points
into all parts of the image (corners too.) The whole grid
```

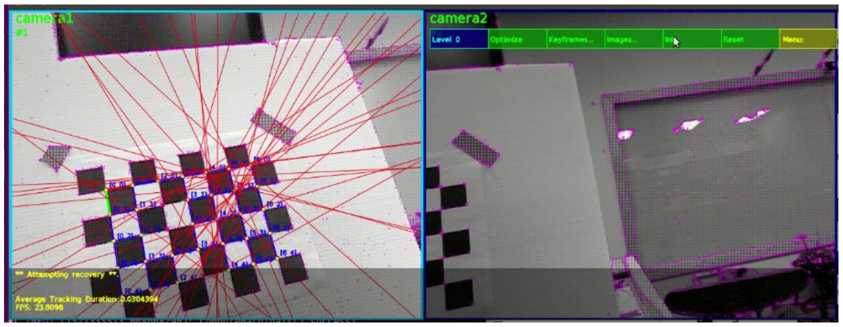
Figure 4.1: The window of the intrinsic calibrator.

In order to calibrate the Kinect V1, we had to edit the provided ROS launch file by specifying the camera info and the raw RGB data from the Kinect camera so that it can calibrate the Kinect V1. However, the first problem that I had encountered is that the calibrator could not instantiate. This was due to the mismatch of the camera model required by the calibrator and the default camera model of Kinect V1. While the calibrator requires Taylor camera model, Kinect set the camera model to plumb bob as default. Although the calibrator is supposed to change the camera model of the Kinect V1 to Taylor camera model so that it can start calibrating the camera, as I observe the camera info of the calibrating Kinect V1, the calibrator was only able to change the camera model for one moment and the camera model will be reset back to plumb bob again. Hence, when the calibrator rechecks the camera model of the camera, the changed camera model was already reset back to the default camera model, causing the calibrator to keep checking and changing meaninglessly the camera model of the Kinect V1.

The approach that I took to tackle this problem was to use the camera info from the webcam of the computer. The developer of MCPTAM also provided a calibrator for webcams. What I did was to change the ROS launch file of calibrator so that the calibrator received RGB data from the Kinect V1 but camera info from the webcam. In this way, the calibrator can calibrate the Kinect V1 based on the RGB data of the Kinect V1 and set the desired camera model to the webcam. Since the calibrator was able to change the camera model of the webcam successful to Taylor camera model, using the camera info of the webcam allowed the calibrator to instantiate. Since the calibrator cannot set the calibrated parameters to the Kinect V1, I moved the .yaml file with the saved parameters to the file where the ROS launch file that launches the Kinect V1 camera will take in files with parameters to set its camera info at the beginning. This way, when I used the Kinect V1 for MCPTAM, the Kinect V1 will set its parameters based on the calibrated .yaml file. However, since there was also a possibility that using the camera info of the webcam can cause errors in the calibrated parameters of the Kinect V1 camera, I redo the calibration again but with camera info of Kinect V1 instead of the webcam. This works because now the Kinect V1 is not loading the default parameters but the parameters that were calibrated with the webcam which has the correct camera model to make the calibrator instantiate. Thus, at last, I was able to have two .yaml files for each Kinect V1 with the parameters calibrated with the correct camera info and RGB data from the corresponding Kinect V1.

By launching the ROS launch file of the extrinsic calibrator, or the pose calibrator, the window for the calibration will be instantiated. The pose calibrator requires all cameras used in the multi-camera system to be launched, and the display of the calibrator will show the RGB data of each camera. As the Figure 2 shows, there are two windows that each show the RGB data of one of the two Kinect V1 that we are using. The extrinsic calibration also uses a checkerboard. In order to calibrate the two Kinect V1, we had to edit the provided ROS launch file by specifying the camera info and the raw RGB data from the Kinect cameras, and furthermore, the number of corners and length of the edges of the boxes in the checkerboard. The pose calibrator will first, by pressing space, initialize a map based on the checkerboard captured by the first camera, as shown in Figure 3. Then, as we translate the second camera to the checkerboard while the first camera is still maintaining the generated map, by pressing space again, the second camera will be added to the map generated previously. Then, color dots that indicates features in the environment will be shown in the window. To allow more accurate tracking and mapping when running the MCPTAM, before optimizing the calibration, one should more around the cameras to let them undergo different motions and capture more map points. By pressing the "Optimize" button, the calculate the calibrated parameters. If the residual error is small enough, one can save to a .dat file by pressing "Save" button.

The first problem with the extrinsic calibration that I ran into was again that the calibrator could not instantiate. Every time when I launch the ROS launch file, the system threw an error due to assertion failure. This error was also observable when I ran MCPTAM. The assertion failure was due to not initializing a variable in the cpp files. As I examined the code, it turned out that the initialization of the variable was positioned right after the assertion statement. Hence, I moved the initialization statement one line above so that there would be no more assertion failure due to the variable that would never be initialized otherwise. With this fix, both the pose calibrator and the MCPTAM algorithm were able to instantiate. However, as I tried to initialize the map by the first camera, the pose calibrator cannot initialize the map due great reprojection error shown in the red line in Figure 2. Unfortunately, I wasn't able to identify the cause of this error by the end of this quarter. In the future, I will continue to work on MCPTAM, primarily solving the problems in the extrinsic calibration.



** Attempting recovery **
Average Tracking Duration 0.0304394
FPS: 23.9098

Figure 4.2: The window of pose calibrator with the reprojection error

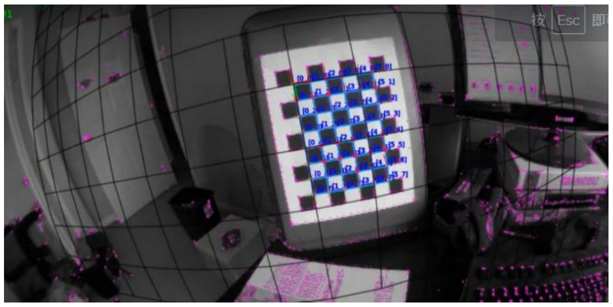


Figure 4.3: The map generated by the first camera during the pose calibration if working properly

The first problem with the extrinsic calibration that I ran into was again that the calibrator could not instantiate. Every time when I launch the ROS launch file, the system threw an error due to assertion failure. This error was also observable when I ran MCPTAM. The assertion failure was due to not initializing a variable in the cpp files. As I examined the code, it turned out that the initialization of the variable was positioned right after the assertion statement. Hence, I moved the initialization statement one line above so that there would be no more assertion failure due to the variable that would never be initialized otherwise. With this fix, both the pose calibrator and the MCPTAM algorithm were able to instantiate. However, as I tried to initialize the map by the first camera, the pose calibrator cannot initialize the map due great reprojection error shown in the red line in Figure 2. Unfortunately, I wasn't able to identify the cause of this error by the end of this quarter. In the future, I will continue to work on MCPTAM, primarily solving the problems in the extrinsic calibration.

MultiCol ORB SLAM:

Since ORBSLAM gave good results with single camera system, we decided to go with MultiCol SLAM which is extended version of ORBSLAM2 and supports multiple camera feeds. There was no ROS build available for this algorithm. So we made ROS integration resolving all the dependencies and successfully integrated and ported to ROS. MultiCol SLAM requires calibrated cameras, both intrinsic and extrinsic parameters in taylor camera model. We used some available MATLAB tool box to get the camera parameters. But the fisheye lens with ZR300 was not that clear. So the calibration tool was not able to get the camera parameters properly. Next we tried using Opencv to get camera parameters, but the results given by opencv camera calibrators had large errors and not accurate. So we resorted to using the MCPTAM which has an internal calibrator module with Taylor camera model. It gave camera parameters better than opencv , but still with significant error. The SLAM algorithm works really well with well calibrated information for a sample dataset. Since we were not able to get accurate calibration information, the trajectory given for live camera feed was not that accurate. It had some trouble estimating the camera pose and it was losing track often. This algorithm gives trajectory which can post processed to get the point cloud information. It was generation trajectory, but the results were not accurate for live camera feed.

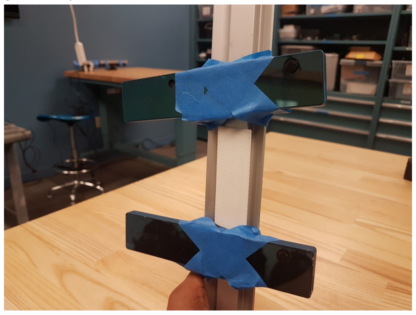


Figure 4.4: Two Realsense cameras in a fixed rig.

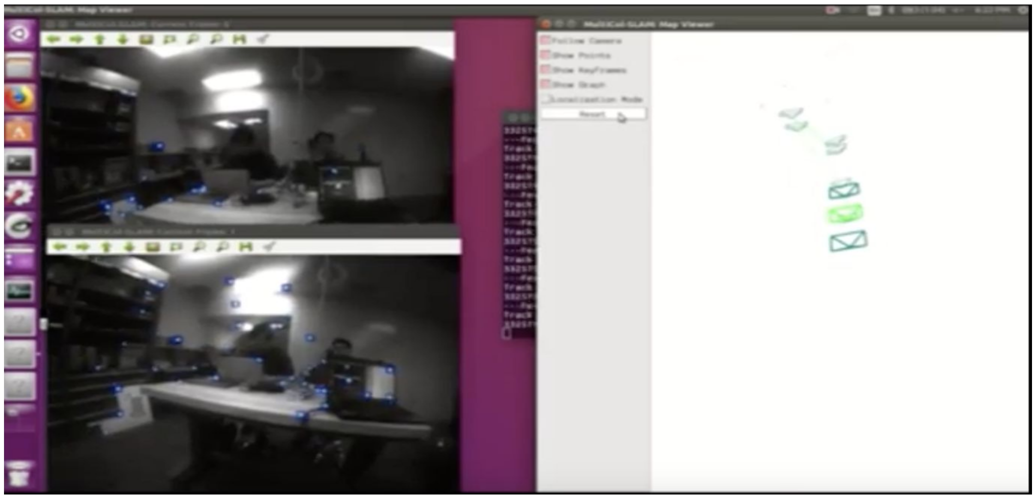


Figure 4.5: This is the multi camera orb slam module. Currently we are getting feeds from 2 cameras. Which you can see on the top left and bottom left windows. Green dots with blue boxes shows the features which are being tracked in those frames.On the right side it shows the trajectory of the camera. The motion of the camera being calculated from the tracked feature points. Green and Dark blue square boxes defines the camera system. Red dots represents the points which are used for estimating the camera pose. Blue crosses represents the key frames generated and it accurately gives the path of the camera motion. This gives the trajectory file which contains the camera pose at each keyframe position. With some post processing , the 3D point cloud can be generated from the trajectory information.

RTAB-Map:

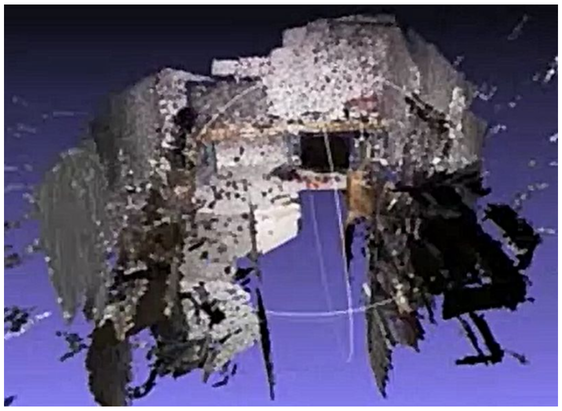


Figure 4.6: Point cloud of part of the lab

RTAB-Map supports multi-camera functionality by combining the graphs of individual cameras into one. Each camera is independent of each other and does not influence the graph of the other camera. This adds the robustness and accuracy that we desire in the most simple sense. If one camera were to fail in the field, the other will be able to continue. Furthermore, because the cameras are independent, we can separate data points during post processing in case of error in a single camera. Having multiple cameras will provide more data points for the increase in accuracy that we require. By utilizing the tf ROS package, we can visualize and keep track of multiple coordinate frames and graphs. The transformation we used is to have the secondary camera relative to the main camera. The tf package provides this transform by specifying the x/y/z offset in meters and rotation about x/y/z in radians. These parameters can be calculated with an extrinsic calibration technique and then converted to the necessary offsets or they can determined by measuring the offsets between the physical cameras. RTAB-Map's real-time generation of the point cloud provides the overlapping point cloud of two separate point clouds. This provides additional points which is useful for post process.

RTAB-Map did struggle with featureless environments and "empty space" environments. The latter refers to the fact when the cameras were more than roughly 4 meters away from the a wall or object. This issue does not affect the end goal of utilizing the algorithm in a small tunnel, if the cameras are moved around as to avoid being pointed down the tunnel and just to the sides of the tunnel. The featureless environment should also not be a problem in the tunnels as the tunnels are not as uniform as the walls in a lab. Another issue to note is the provided method to correct lost odometry. A window is provided to show where the user should reposition the camera to restart the algorithm and hopefully correct the odometry. Even in testing in the lab, sometimes it is hard to restart the lost odometry if the pattern is unrecognizable or a repeat feature. This problem can be exasperated in the tunnels were one dirt wall looks just like the next.

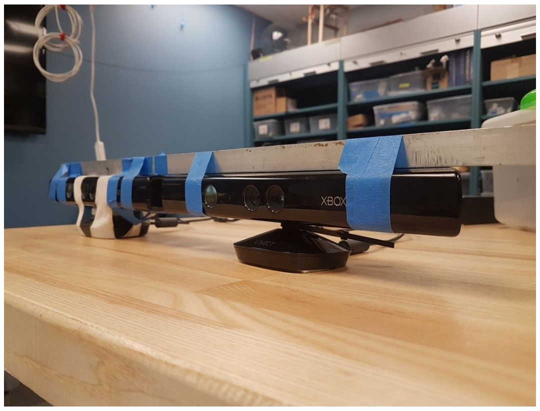


Figure 4.7: Setup for dual Kinect v2. This was a simple set up for testing purposes. RTAB-Map allows for the set-up to be more complex and can easily be changed. It does not require the intrinsic calibration that is needed for the

other two algorithms. The transformation can be measured and if desired, be done with a extrinsic calibration and converted to x/y/z offset and rotational offset.

5. Conclusion

We tried to mitigate the effects of limited field of view of the single camera system and effectively build a robust system , which will not lose track in tight turns and also produces better and more accurate 3D models. We increased the field of view using Fisheye lens and using multiple cameras. We have tested 3 SLAM algorithms RTAB-Map, MultiCol SLAM, MCPTAM. We are able to successfully integrate the SLAM algorithm with our system. With RTABMAP we were able to successfully get the 3D point cloud of the lab. With MultiCol SLAM, the model generated was not accurate and the system was not able to track effectively due to some calibration issue with cameras. Since the fisheye lens in ZR300 were not clear, calibration was little challenging. Similarly, MCPTAM was not able to run because of extrinsic camera parameters.

Future work on this would be to get accurate camera extrinsics with cameras placed in optimal position (with little overlap in the field of view for MultiCol SLAM) and get the point cloud out from the system. Then we have planned to compare the point clouds from different SLAM algorithm with registration script available with us. We have also planned to head to Mud caves near by to get close to real field results to test under different lighting conditions. One more challenge that may arise with multi camera system is the lighting. Currently we have single LED panel for single camera and it sufficiently lights the system. But once we move to multi-camera system, we may need to find a better way to light a larger field of view. We have also planned to collect some data for testing out different SLAM algorithms under different conditions. Post processing is another possibility to explore. While RTAB-Map provides a suitable point cloud for real time mapping, there can be post processing to remove stray data points and smooth out the results.

6. References

1) Mur-Artal, R., Montiel, J. M. M. and Tardós, J. D., 2015. ORB- SLAM: a Versatile and Accurate Monocular SLAM System. IEEE Transactions on Robotics 31(5), pp. 1147–1163.
2) S. Urban, S.Hinz et al, 2016 MULTICOL-SLAM - A MODULAR REAL-TIME MULTI-CAMERA SLAM SYSTEM, arXiv:1610.07336 [cs.CV]
3) Scaramuzza, D., Martinelli, A. and Siegwart, R., 2006. A Tool- box for Easily Calibrating Omnidirectional Cameras. In: Pro- ceedings of the Annual Conference of the Robotics Society of Japan (RSJ), IEEE, pp. 5695–5701.
4) Harmat, Adam. “Aharmat/Mcptam.” GitHub, github.com/aharmat/mcptam/wiki.
5) A Harmat, M Trentini, and I Sharf. Multi-Camera Tracking and Mapping for Unmanned Aerial Vehicles in Unstructured Environments. In *Journal of Intelligent and Robotic Systems*, August 2014.
6) “Monocular SLAM.” *Monocular SLAM | PTAM*, www.doc.ic.ac.uk/~ab9515/ptam.html.
7) Labbe, Mathieu. (2015). *Simultaneous Localization and Mapping (SLAM) with RTAB-Map.* Presentation at Université Laval, Québec, Canada.