Mayan Multi-Camera Project 2.0

University of California, San Diego
CSE 145/237D Spring 2019
Aravind Mahadevan, Neil Sengupta, Jonathan Lam, Samer Sabri, Yikai Chen

Abstract

Archaeologists today still document their discoveries by hand despite the technology we have today. Manual reconstructions/hand drawings of these tunnels are prone to error and also very time consuming due to extreme topological conditions and complexity of tunnel structures. Using technology, we can tackle this problem by rendering a 3D model of the tunnels. In the past, there has been work done using a single camera setup succeeded by a multi-camera Kinect setup for better 3D model accuracy. This paper presents the use of a dual-camera system consisting of two Intel Realsense Cameras that provide reliable tracking and depth information to effectively render 3D models of Mayan tunnels. The proposed setup is cheaper, portable and easy to set up.

Introduction

With the continued exploration of historic sites of the Mayan civilization, archaeologists have to explore tunnels. Documenting these explorations involve sketches and hand drawings, both of which can be quite time-consuming. In addition, these tunnels are often very complex to navigate and this may induce errors in the drawings. In the past, there has been work in using camera systems to map these tunnels. The first iteration involved a single camera setup for capturing data but failed to provide reliable tracking. This is not helpful for mapping complex tunnel structures because the camera system would lose tracking between frames thereby leading to error-prone or incomplete 3D models.

New methods use a combination of LiDAR, Kinect Cameras, and Realsense Z300 cameras. The problem with LiDAR is that besides being cumbersome it requires post-processing, which means the system cannot indicate if there is an erroneous frame during recording, which could lead to wasted recordings. This problem is overcome with a multi-camera setup which uses a SLAM algorithm for stitching together images in real time. In addition, this system also provides reliable tracking and depth information. Our project uses two Intel Realsense cameras - Intel T265 (released Early 2019) and Intel D435 as a part of the multi-camera setup. We are able to reliably capture RGB-D data from the D435 camera and tracking and pose information from the T265 camera.

Prior to any camera data retrieval, we throttle the camera feeds and perform camera calibration. We perform calibration for the dual camera setup in order to get transformation and rotation matrices, intrinsic and extrinsic parameters for the system. We use RTAB-Map (Real Time Appearance-Based Mapping) for SLAM that combines the tracking, depth and RGB-D data from the dual camera setup to generate a point cloud map that can be visualized on a laptop or in AR/VR. **Figure 1**, below, is an overview of the entire pipeline.

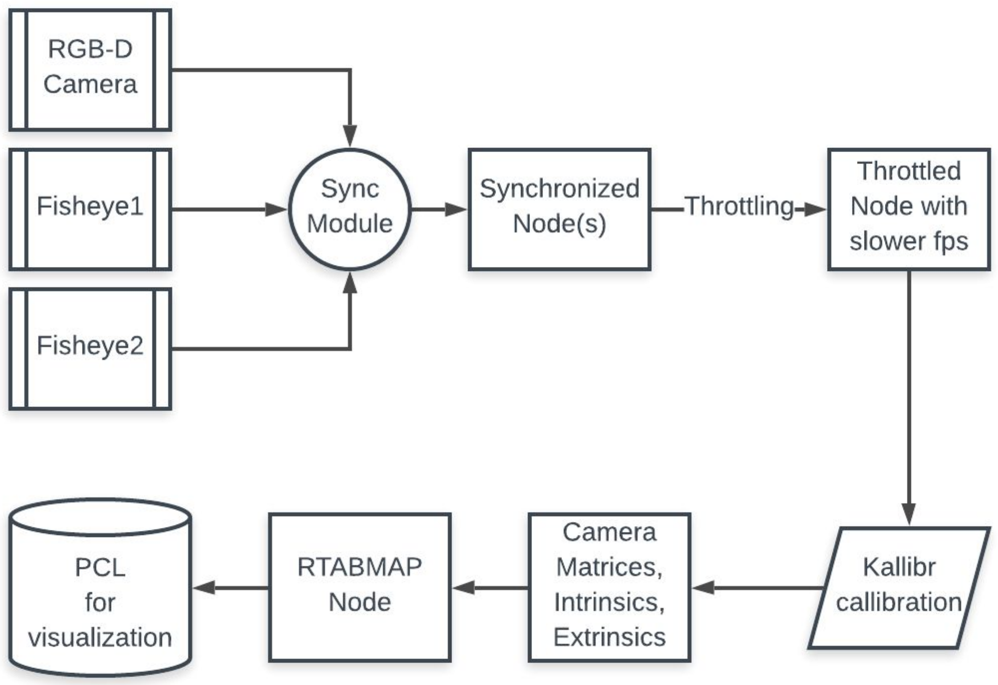

Figure 1: Overall Software Pipeline

<u>Technical Material</u>

Figure 2: Intel T265 (left) and Intel D435 (right) that were used to capture data

Intel RealSense Cameras

D435

The D435 **(Figure 2)** is a Realsense camera that captures both depth data and RGB data. This will be the main camera that will be used to map the recorded environment. The Intel D435 Depth Camera [8] offers accurate depth perception when the device is in motion and covers more field of view and minimize blind spots compared to its predecessor.

T265

The T265 **(Figure 2)** is a big improvement from the previous iteration since it uses inside-out tracking thereby not depending on external sensors for environment mapping [9]. It has 6 degrees of freedom and uses input from the two internal fish-eye cameras. This is supported by the on-chip VSLAM algorithm which is able to construct environment maps using the device's location. The on-chip computation also means that this process is platform independent. This makes it further adaptable to the current problem allowing us for efficient location tracking in the tunnel.

Kalibr:

When doing any SLAM based algorithm for a multiple camera system, it is very important to retrieve the transformation matrices between the multiple cameras. Since we used an Intel D435 color and depth camera and Intel T265 tracking camera, we needed to calibrate the two cameras and retrieve camera intrinsic and extrinsic parameters. For this, we used Kalibr, an open source multi-camera calibration tool that is readily available on Github. Kalibr is able to calculate the intrinsic and extrinsic camera parameters of a multiple system camera as long as the multiple camera system has an overlapping field of view. (cite Kalibr) In order to use Kalibr, we first needed to define a calibration target. In our case, we used a 6x8 April grid. We primarily use April grids because it has a track record of calibrating cameras accurately and easily**.** Furthermore, the previous iteration of this project used April grids for calibration and the Kalibr wiki recommends using April grids as well [1,2]. With the calibration target defined, the next task is to create a rosbag that takes in a camera feeds from the T265 camera and D435 camera that shows different angles of the calibration target or April grid. In order to do this, we needed to throttle the camera stream because having too much data fed to Kalibr will lead to a longer calibration time. Furthermore, it is recommended to reduce the frequency of the camera streams to around 4 Hz [3]. Once the camera feeds are throttled to a lower frequency, we begin capturing the April grids to store it in a rosbag. When doing this process, the target has to be viewed by both cameras in all positions. What this means is that both cameras have to not only have overlapping views but the April grid has to be captured in all poses and angles by both cameras. With respect to our camera setup, we only needed the camera feed from one of the fisheye cameras from the T265 and the D435 camera due to the fact that T265 is already internally calibrated. Furthermore when during this process, we made sure at least 9 points on the April grid was viewable at all times so that Kalibr considers the image as a data point for calculating the intrinsic and extrinsic parameters. Once the rosbag has been captured with both the camera feeds, we need to set the parameters for running Kalibr. In particular, we need to specify the camera models of the cameras that were used to record the April grid along with the distortion model. For camera models, Kalibr supports the following camera models: pinhole camera, omnidirectional camera, double sphere camera, and extended unified camera. For the distortion models, Kalibr supports the following distortion models: radial-tangential, equidistant, fov, and none [4]. These parameters are extremely important to specify correctly because putting the wrong parameters can lead to Kalibr erroring out and giving bad results. Although these camera models and distortion models are supported, Kalibr does not support all combination of camera models and distortion models. Once these parameters and rostopics from the rosbag are specified, we can launch Kalibr. Once calibration is completed, there are three outputted files. In these files, specifically, the pdf, provides us with plots and also information about the reprojection error and allows us to visualize the translation between multiple cameras. In the outputted txt and yaml files, the camera intrinsics can be found along with the rotation matrices and translation vectors which describe where each camera is with respect to one another. With the outputted files, we use the information from these files for RTABMAP.

RTAB-MAP (Real-TIme Appearance-Based Mapping):
When deciding what SLAM algorithm to use for the camera setup we have, RTAB-MAP was chosen due to its compatibility with our current ROS environment. RTAB-MAP is a SLAM algorithm that is able to do RGB-D, Stereo and Lidar mapping of its environment through a graph-based approach. In real time, the RTAB-MAP algorithm generates a graph of its surroundings as the camera is panning around its environment. When generating nodes on the graph, a loop closure detector is used to determine if the new image is from a new or previous position. A new node is generated when a loop closure hypothesis is accepted and the constraints have been added to the graph. From there a graph optimizer attempts to minimize the errors in the map. [5]

RTAB-MAP also has the function to use visual and lidar odometry to determine the position and orientation of the cameras using the data captured from the cameras. These types of odometry look for features within the image and compare them to previous frames to track the location of the camera. RGBD odometry uses rgbd images to compute odometry from visual features extracted from them. Stereo odometry uses stereo images from a left camera feed and attempts to find similar features in the right camera. ICP odometry uses laser scan to compute odometry through iterative closest point registration. [6]

For our project, we chose RGB-D mapping due to the D435 providing RGB and depth data for us to create an RGB-D map while the T265 would be providing odometry information for improved tracking of the environment due to the D435 lacking onboard instruments to generate IMU data.

Milestones

Milestone name	Milestone Description	Date of Completion	People Responsible	Deliverable	Status
ROS Learning	Get familiar with the basics of ROS	April 26, 2019	Everyone other than Neil	Screenshots of completed tutorial/ Certification	Finished
Camera Calibration	Calibrate Intel RealSense cameras	May 10, 2019	CCT Team	Video of the two cameras running	Finished
Camera Data Retrieval	obtain RGB and depth data from the camera	May 10, 2019	CDRT Team	BAG file containing raw camera data	Finished
Reprojection Error Optimization	Update Kalibr software and experiment with Kalibr configuration to attain minimized reprojection error.	May 31, 2019	CCT Team	Camera parameters that minimize reprojection errors	Finished

Refactoring and Documentation	Document code and refactor code to reduce setup overhead.	May 31, 2019	CCT Team	Documented code and setup project documents	Finished
Implement SLAM	Apply SLAM to data collected to obtain a 3D model	May 31, 2019	CDRT team	3D model	Finished
Final Presentation and Video	Fulfilling Class Requirement	June 7, 2019	All	Get Final Video and Presentation Submitted	Finished

ROS Learning:

Members of the group who didn't know ROS started out by following the ROS tutorial on https://www.robotigniteacademy.com/en/course/ros-in-5-days/details/. As we got more acquainted with the project and consulted with our mentor, Ferrill, we realized that most of our ROS work is related to message passing. We decided that it is best if we focus our efforts on the following capabilities: being able to view ROS topics as well as subscribing and publishing to topics. So, everyone started getting more acquainted with that section of the tutorial instead of learning additional topics that are not directly relevant to the project. We were able to complete this part of our planning on time.

Deliverable

https://www.youtube.com/watch?time_continue=4&v=f_bh6WEQJ90 This video shows how we finished a quiz of ROS topics publisher and subscriber. In this video, we created a ROS publisher to subscribe to the laser topics("kobuki/laser/scan") of this robot. From the subscriber, we could get the distance between the robot and the wall and we made a callback function to turn distance into moving direction and speed. Then we created a publisher, which published the moving command(direction and speed) to the robot, using the topic "cmd_vel", depending on the result of the callback function from the subscriber. In the context of this project, in order to synchronize all the camera feeds and throttle all camera feeds this requires subscribing to certain rostopics and publishing different modified topics. This will be talked about more in detail below but this demonstrates our ability of message passing in ROS.

Camera Calibration

After learning ROS and being able to work with ROS nodes, our first goal was to get both the camera nodes running and being able to view the feed from both the nodes. We wrote a launch file to first individually launch both the cameras and then a script to launch both the cameras at once. The hardware was a bit finicky since we encountered multiple tries where the T265 camera was not being recognized -- the system failed to recognize a multi-camera launch. Over the course of the quarter, we added some synchronization changes to the realsense codebase to make sure that we would be able to consistently launch both the cameras in the future. As we carried out the calibration process, we first implemented camera synchronization - a node to have an approximate synchronization between the messages that were delivered by the two camera nodes. Next, we throttled down the feeds of both the cameras initially to one frame every four seconds. Over time, as we carried out the calibration process, we experimented and bumped this up to four frames every four seconds. We then used the throttled and synced feeds to first carry out calibration for individual cameras. We experimented with different camera models and figured out that individually the models that work the best are Pinhole Camera Model with

Radial-Tangential Distortion Model for D435 and Omni-Direction Model with Radial-Tangential Distortion Model for T265.

Deliverable

We started off by updating Kalibr libraries and Realsense libraries to make sure they worked reliably. This took a long time since the development environment was not configured correctly. We were able to launch both camera nodes successfully and consistently without errors. In addition, we carried out preliminary camera calibration with rather high reprojection errors.

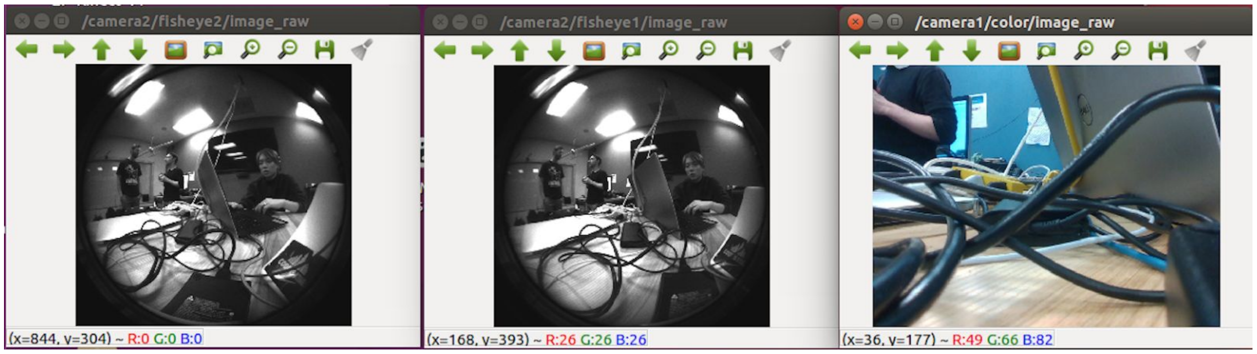
/camera2/fisheye2/image_raw
(x=844, y=304) ~ R:0 G:0 B:0
/camera2/fisheye1/image_raw
(x=168, y=393) ~ R:26 G:26 B:26
/camera1/color/image_raw
(x=36, y=177) ~ R:49 G:66 B:82

Figure 3: All Three Camera Feeds

Camera Data Retrieval

In order to retrieve data, we wrote a custom launch script to subscribe to the camera nodes and store the synced and throttled data coming from the two nodes into a bag file.

Deliverable

The script allowed us to reliably store data from different camera topics into a bag file. We can actually view all the recorded data using the play command (RosWiki). It also allowed us to access individual data streams by topics.

Reprojection Error Optimization

We were getting a fair amount of data points where the reprojection error was out of bounds. To fix this, we fine-tuned our calibration process. We spent considerable time figuring out sweet spots for both camera lenses to recognize the frames. We also experimented with different camera models. In addition, we tweaked our Kalibr script to give us real-time information on whether a frame will be used in the calibration process or not. This proved very useful to us and saved us a lot of wasted runs. Although we were able to optimize our reprojection errors, Kalibr still does not support all combination of camera models and distortion models. We believe that once Kalibr supports more models, we might be able to get even lower reprojection errors. **Figure 4** shows our final calibration results and shows that there are very few outliers that have high reprojection error.

Deliverable

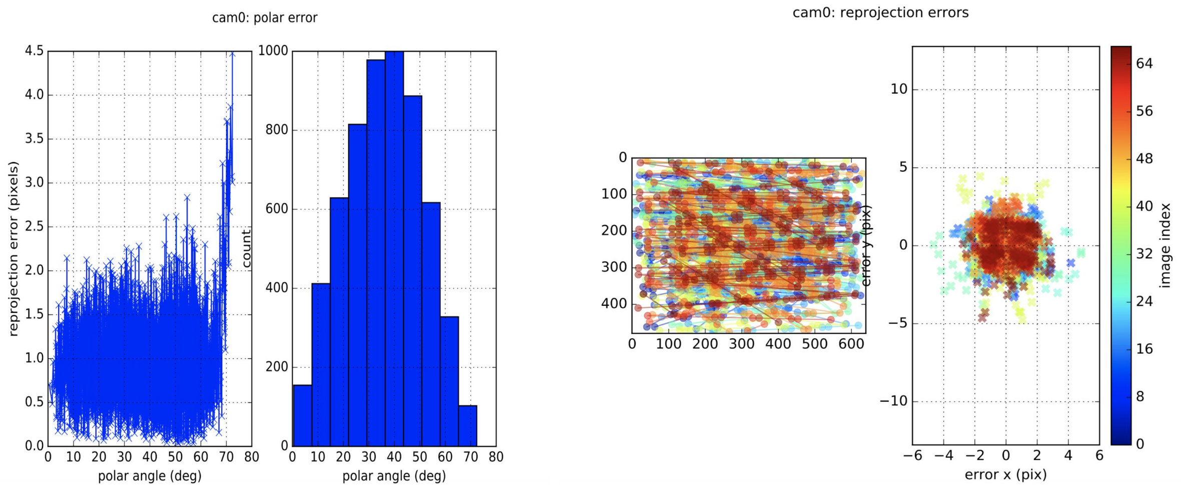

Figure 4: Reprojection Errors

Refactoring and Documentation

Throughout the course of the quarter, we actively documented our development process, noted down errors with their respective workarounds/fixes. This is maintained in our group website (https://indianastones.github.io/index.html). In addition, all the code was uploaded to Github with complete instructions on how to and get running within a manner of minutes. We felt this was necessary because such a thing didn't exist and was a big initial hurdle for us.

Deliverable

Documented codebase and documentation for future teams to easily work with (https://github.com/indianastones/maya-archaeology).

Implement SLAM

In order to create a 3D model with our D435 and T265 camera system, we have to select a suitable multi-camera SLAM algorithm solution that is compatible. RTAB-MAP was selected as the algorithm we will use for this project due to past success the previous iteration of the project has.

As discussed before, camera data retrieval had minor issues that we and our mentors were unable to fix due to strange interactions our development laptop has had with the T265 camera. Since the Linux operating system does not immediately recognize the camera, we were forced to restart our ros launch and unplug the camera multiple times until the camera was recognized by Linux. Our only current explanation with this problem is most likely the hardware itself.

After launching publishers for all camera feeds, we fed the RGB and depth data from the D435 with the odometry calculated from the T265 into RTAB-MAP. Below is an example of the generated model done by RTAB-MAP.

Deliverable

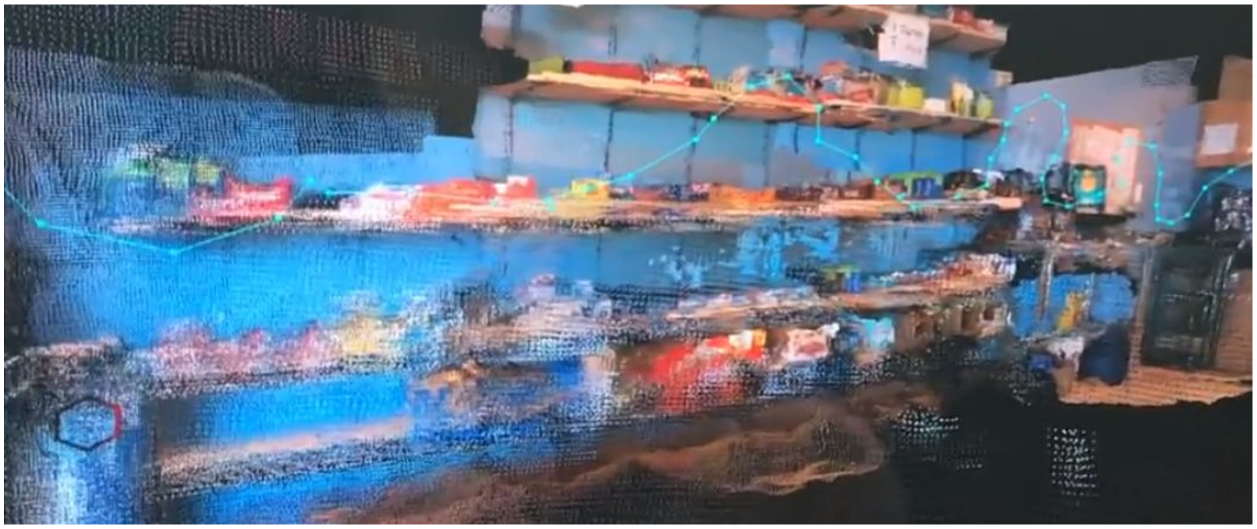

Figure 5 : RTAB-MAP Model

Figure 5 displays a 3D model of the wall we tested our final version of our RTAB-MAP implementation. This implementation only takes the odometry data calculated by the T265 itself. The blue linked line is the path the camera itself traversed when recording the environment. There is some blurriness that occurs but that is due to the point clouds only generating the 3D model from only the specific angle the image was recorded. The model is a decent representation of the recorded environment.

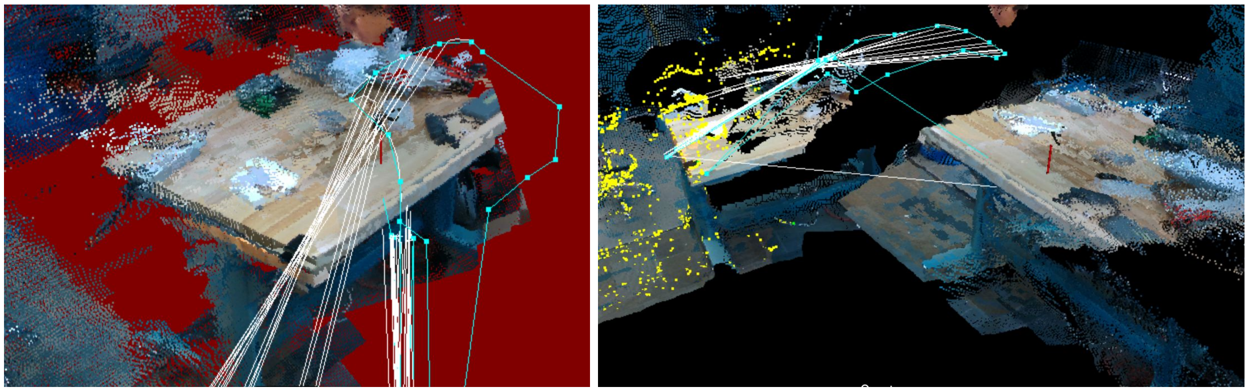

Figure 6: Failed RTAB-MAP with Visual Odometry

Figure 6 shows another implementation that is incomplete due to time constraints. This implementation uses visual odometry to aid in the mapping of the environment instead of purely on the onboard calculations from the T265. Visually, the models created by this approach can be seen as being the most accurate, as seen in the left image in **Figure 6**. However, there is a problem with how RTAB-MAP constantly mishandling its relative position and ends up creating a separate model somewhere else in virtual space, as shown in the right image of **Figure 6**. Our understanding of this error leads us to believe is that there is a conflict between D435's and T265's knowledge of its current position. This causes the camera to jump between two locations in virtual space and map the environment separately. From these issues, we think it is caused by an error in our sensor fusion attempt in both our cameras.
All code related to our RTABMAP implementation is contained within our Github repo below.
(https://github.com/indianastones/maya-archaeology)

Conclusion

In this project, we experimented with a setup built by two Intel realsense cameras to replace the system built up by a Kinect camera and a ZR300 camera that was previously used in the Maya-multicam project. For SLAM, we used RTAB-MAP due to previous success with it in the prior iteration of the project. After implementation and calibration of our new camera system, we are able to get better tracking with our RGB-D data from the D435 and T265 than two identical ZR300 because T265 has onboard odometry calculations. And we are able to obtain similar RGB-D data from D435 compared to the ZR300s and be less bulky than a dual Kinect setup.As a result we obtained decent results with better tracking than previously, but also have more portability than Kinect cameras.

While we have delivered a decent implementation of RTABMAP, we were not able to fix an improved version due to time constraints. For any possible future work, we suggest to look more into sensor fusion to get the visual odometry working and get both T265 and D435 to agree on their positioning within RTAB-MAP's visual odometry.

References

1) Edwin Olson (2011). AprilTag: A robust and flexible visual fiducial system. Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), pp. 3400–3407
2) Michael Kaess. http://people.csail.mit.edu/kaess/apriltags/, Nov. 2013
3) J. Maye, P. Furgale, R. Siegwart (2013). Self-supervised Calibration for Robotic Systems, In Proc. of the IEEE Intelligent Vehicles Symposium (IVS)
4) J. Kannala and S. Brandt (2006). A Generic Camera Model and Calibration Method for Conventional, Wide-Angle, and Fish-Eye Lenses, IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 28, no. 8, pp. 1335-1340
5) Labbe, Mathieu. "RTAB-MAP: Real-Time Appearance-Based Mapping." RTAB-Map. N.p., 2019. Web. 14 June 2019. http://introlab.github.io/rtabmap/
6) Labbe, Mathieu. "Rtabmap_ros WIki." Ros.org. N.p., 2019. Web. 14 June 2019. http://wiki.ros.org/rtabmap_ros
7) "Wiki." Ros.org, wiki.ros.org/rosbag/Commandline#rosbag_play.
8) Admin. "Overview of the Intel® RealSense™ Depth Camera." Intel® Software, Intel, 18 Sept. 2017, software.intel.com/en-us/realsense/d400.
9) "Tracking Camera T265 – Intel RealSense Depth and Tracking Cameras." *Intel® RealSense™ Depth and Tracking Cameras*, www.intelrealsense.com/tracking-camera-t265/.