



**US Army Corps
of Engineers®**
Engineer Research and
Development Center



Optimized Low Size, Weight, Power and Cost (SWaP-C) Payload for Mapping Interiors and Subterranean on an Unmanned Ground Vehicle

Garry Glaspell, Steven Lessard, Benjamin Christie,
Kyle Jannak-Huang, Noah Wilde, Weiyu He, Osama Ennasr,
Daniel Pham, Daniel Hasemann, Phil Devine,
and John Kiene

March 2020

The U.S. Army Engineer Research and Development Center (ERDC) solves the nation's toughest engineering and environmental challenges. ERDC develops innovative solutions in civil and military engineering, geospatial sciences, water resources, and environmental sciences for the Army, the Department of Defense, civilian agencies, and our nation's public good. Find out more at www.erdclibrary.usace.army.mil.

To search for other technical reports published by ERDC, visit the ERDC online library at <http://acwc.sdp.sirsi.net/client/default>.

Optimized Low Size, Weight, Power and Cost (SWaP-C) Payload for Mapping Interiors and Subterranean on an Unmanned Ground Vehicle

Garry Glaspell, Steven Lessard, Benjamin Christie,
Kyle Jannak-Huang, Noah Wilde, Weiyu He, Osama Ennasr,
Daniel Pham, Daniel Hasemann, Phil Devine, and John Kiene

*Geospatial Research Laboratory
U.S. Army Engineer Research and Development Center
7701 Telegraph Road
Alexandria, VA 22315-3864*

Final report

Approved for public release; distribution is unlimited.

Prepared for U.S. Army Corps of Engineers
Washington, DC 20314-1000

Under Program element number 62784
Project number 855
Task number 23

Abstract

Section 3 of the FY15 Force 2025 Maneuvers Annual Report indicates that in Dense Urban Areas (DUA), specifically in a subsurface, surface, or super-surface structure, the ability to identify threats will be diminished. Most commercially available Light Detection And Ranging (LIDAR) systems are specifically designed for high-resolution aerial imaging and mapping applications. As a result, they tend to be large, heavy, power-hungry, data bandwidth intensive, and expensive. They also employ lasers that are not typically eye-safe, which limits their overall effectiveness in subterranean and the interiors of subsurface or super-surface structures. However, due to recent advances in the automotive industry, there are new generations of Size, Weight, Power, and Cost (SWaP-C) sensors that are eye-safe, making them suitable for use indoors and in subterranean environments. While these tradeoffs limit their effective use to hundreds of meters (compared to kilometers for their more expensive counterparts), they are ideal candidates for use in subterranean and building interiors. While cameras fill this niche to some extent, the volumetric calculations provided by these sensors provide additional intelligence to shape the security of the environment and offer more precision when maneuvering troops. These sensors would provide the warfighter with situational understanding in previously inaccessible locations. Therefore, to aid in the Army's need to obtain and maintain situational understanding in DUAs, the authors propose utilizing low size, weight, power, and cost (SWaP-C) sensors, on a robot platform, for surveying and mapping underground structures and building interiors. Rapid/near real-time data processing is possible by utilizing open-source software and commercial off the shelf (COTS) components. Using the preferred sensor payload autonomously was also explored.

DISCLAIMER: The contents of this report are not to be used for advertising, publication, or promotional purposes. Citation of trade names does not constitute an official endorsement or approval of the use of such commercial products. All product names and trademarks cited are the property of their respective owners. The findings of this report are not to be construed as an official Department of the Army position unless so designated by other authorized documents.

DESTROY THIS REPORT WHEN NO LONGER NEEDED. DO NOT RETURN IT TO THE ORIGINATOR.

Contents

Abstract	ii
Figures	iv
Preface	v
1 Introduction.....	1
1.1 Background.....	1
1.2 Objective	2
1.3 Approach	2
2 Sensors.....	5
3 SLAM Modules.....	14
4 Sensor Fusion	23
5 Virtualization	34
6 Wireless Ranging.....	38
7 Automation.....	45
8 Future Advancements	57
9 Conclusion.....	59
References	61
Appendix A: Sensor List.....	63
Appendix B: Setup Procedure for Astra and D435.....	69
Appendix C: Installation of SLAM modules	75
Appendix D: RTABMAP Launch files	128
Appendix E: POZYX	163
Appendix F: Automation	197
Unit Conversion Factors	208
Acronyms	209
Report Documentation Page	

Figures

Figures

Figure 1. Arduino controlled LIDAR lite. The resulting point cloud is shown in the inset.....	6
Figure 2. Stereo, depth and point cloud generated from a DUO MLX.	7
Figure 3. Point clouds generated from ZED photogrammetry sensor.....	8
Figure 4. Depth image generated from Kinect Xbox One.	9
Figure 5. RGB point cloud collected using Kinect Xbox One.	10
Figure 6. Infrared point cloud collected with Kinect Xbox One.	11
Figure 7. RGB point cloud collected with and Orbbec Astra.	12
Figure 8. RGB point cloud collected using the RealSense D435.	13
Figure 9. High Density point cloud collected using ElasticFusion.....	16
Figure 10. Semi-dense point cloud collected using RTAB-Map.	18
Figure 11. Point cloud collected using only wheel encoders.	21
Figure 12. Schematic showing the relationship of frames in ROS.....	25
Figure 13. Top-down view of NVESD's firing tunnel.....	29
Figure 14. NVESD's 80m long firing tunnel.....	30
Figure 15. Collected point cloud overlayed with Leica C10.....	31
Figure 16. Different perspective of overlayed point cloud.....	32
Figure 17. Detailed overlay of collected point cloud.	32
Figure 18. Heat map of overlayed point cloud.	33
Figure 19. Various anchor and tag configurations.	39
Figure 20. Preferred configuration using three anchors and a tag.....	40
Figure 21. Preferred configuration using four anchors and a tag.	41
Figure 22. Point cloud collected leveraging ultra wide band.	44
Figure 23. Flow diagram of recovery behaviors.	47
Figure 24. Integrating the costmap into automation.....	48
Figure 25. Publishing the odom from RTAB-Map.....	52

Preface

This study was conducted for the Engineer Research and Development Center, Geospatial Research Laboratory (ERDC GRL) of the U.S. Army Corps of Engineers under PE 62784, Project 855, Task 23 (GeoICUE), FWI 68825D (MOWLES FY19). The technical monitor was Dr. Jean Nelson.

The work was performed by the Data Representation Branch of the Topography Imagery and Geospatial Research Division, U.S. Army Engineer Research and Development Center, Geospatial Research Laboratory (ERDC-GRL). At the time of publication, Mr. Vineet Gupta was Branch Chief; Ms. Martha Kiene was Division Chief; and Mr. Ritchie Rodebaugh was the Technical Director for the Geospatial Research and Engineering (GRE) business area. The Deputy Director of ERDC GRL was Ms. Valerie Carney and the Director was Mr. Gary Blohm.

The authors would like to acknowledge the following individuals for their contributions to this project: Dr. Jean Nelson, Mr. Vineet Gupta, and Mr. Rich Curran.

COL Teresa A. Schlosser was the Commander of ERDC, and Dr. David W. Pittman was the Director.

1 Introduction

1.1 Background

Robots and mobile platforms have recently begun to solve many of the goals for remote sensing. Mobile laser scanning devices have enabled the identification and classification of roadside objects (Lehtomäki et al. 2015). Human-mounted solutions, like backpacks equipped with laser scanners, allow for human-in-the-loop navigation of indoor spaces (Wen et al. 2016). These mobile solutions afford operators access to previously inaccessible regions that stationary systems cannot sense.

Autonomous and semi-autonomous mobile remote sensing platforms, including robots, further improve the flexibility that mobile sensing offers. Robotic agents can operate without the need of constant human presence and input (Tsubouchi et al. 2004; Qin et al. 2017). Therefore, environments can be remotely sensed without involving or endangering humans, making surveying safer and more efficient. Many robotic platforms outperform their human counterparts in similar sensing and mapping tasks as well (Ekval et al. 2006), which increases the reliability and accuracy of produced geospatial intelligence. This process of simultaneously localizing a robot in its surroundings and mapping these surroundings is known as Simultaneous Localization and Mapping (SLAM). The fine level details and autonomy afforded by SLAM robots have contributed to the ubiquity of these platforms as mobile and adaptable exploration solutions.

One result of the increased use in robotic remote sensing platforms is a greater portfolio of environments and spaces that can be explored. Urban and subterranean environments pose a unique challenge because they typically feature tight spaces and are often poorly lit. Additionally, underground caves are often rocky and difficult to traverse. Nonetheless, robots have been produced to explore these spaces. Urban-focused robots use three-dimensional (3-D) lidar to scan areas due to the high failure rate of visible-light dependent cameras (Sequeira et al. 1999; Niemeyer et al. 2014; Brenner et al. 2008; Wen et al. 2013). As a result, robots can produce highly detailed maps for the precise environments for which they are selected, including deep mines and archaeological sites (Pollefeys et al. 2000; Jacobson et al. 2018; Shi et al. 2017).

1.2 Objective

This report specifically addresses the Army Warfighting Challenges 2015 and the Army RAS (Robotic and Autonomous System) near-term objective of “Enhancing stand-off from threats and improve situation awareness” and their mid-term objective to “Enhance Platoon, Squad, and Soldier situation awareness.” At a theatre level (CNA 17-21), the Army Critical Capability Gap(s) specifically list the following items:

- Gap #500459: “The Army units at Theater level lacks the capacity to integrate host-nation or third nation security forces into friendly sustainment area and base security operations during decisive action 100% of the time”
- Gap #500915: “The Army at theater level lacks the capability under decisive operations to detect personnel, material and facilities that are subsurface down to 100 meters”
- Gap #500917: “The Army at theater level lacks the capability under decisive operations to maintain situational awareness of subsurface structure down to 100 meters”
- Gap #500918: “The Army at theater level lacks the capability under decisive operations to Map a subsurface structure as specified in the mission with Standard Shareable Geospatial Foundation (SSGF) data standards down to 100 meters.”

1.3 Approach

The goal is to provide the warfighter a cost-effective solution that provides situational awareness, in near real-time, of building interiors and Subterranean (SubT) environments. In order to address the gaps listed above, merging a mapping technology with a robotics platform that has limited computational and power resources will need be done successfully. However, most commercially available LIDAR systems are designed for use outdoors, specifically for high-resolution aerial imaging and mapping applications. As a result, they tend to be large, heavy, power-hungry, data bandwidth intensive, and expensive and not suitable for integration on a robotics platform. Also, the amount of data collected is quite large (gigabits) and requires post-processing (days). In the last few years, the automotive industry’s push for autonomy has resulted in a number of industries producing SWaP-C sensors that should be suitable for a robotics platform and still be capable of surveying and mapping underground structures and the interior of buildings. This capability would provide the

warfighter with data intelligence to develop situational understanding and support troop maneuverability. Utilizing these commercial off the shelf (COTS) low-power sensors to provide a near real-time solution has yet to be implemented, and the benefits provided when compared to their larger, expensive, and power hungry counterparts are unrealized.

In regards to the robotics platform, the Maneuver Center of Excellence (MCoE) SubT equipment list has a variety of robot platforms, including the Firstlook and Packbot 510. Both of these robots are equipped with cameras, but neither system generates point clouds of the environment. In addition to the cost of the robotics platform, additional revenue is also required for mapping sensors (i.e., GeoSLAM ZEB-REVO or Carnegie Robotics Multisense). However, these sensors merely sit atop the robotics platform and do not leverage the robot's on-board sensors for enhanced positioning and localization. The robotics platform that is being leveraged for this report is the Army Ground Vehicle Robot (GVR-BOT) Gen 1.1 platform. This platform is the reference design standard for Project Manager (PdM) Unmanned Ground Vehicle. By combining the robot's localization and positioning sensors with the mapping solution, highly accurate data can be generated.

For this effort, most of the components for the sensors are available COTS items and the manufacturers have supplied the necessary drivers to implement them. Therefore, the technological barrier of getting a closed-source component "A" to play nicely with closed-source component "B" is avoided, which is often clunky at best and requires long lines of code. Instead, the focus was placed on integrating the pieces in a useful and meaningful way for rapid/near-real-time analysis. A key outcome of this project is the ability to survey and map (in 2-D or 3-D) subterranean structures as well as the interior of buildings. Other objectives include object recognition, change detection inside the structures, volumetric and inventory computations, and rapid/near real-time data collection and processing.

The main payoff is the development of a low cost, low power mobile survey, and mapping sensor suite for indoor uses and for use in subterranean environments that is integrated onto a robotics platform. The LIDAR data collected can provide input to mission planning applications such as Sage (Terrain Analysis Toolkit for Mission Planning), situational awareness, operational security, and maneuverability. Finally, the collected data would

be utilized in Army training and virtual reality (Modeling and Simulations) applications. Unmanned Ground Vehicle (UGV) based mapping of interiors will utilize multiple field experiments at various facilities and subterranean environments to evaluate success. Tests will involve stand-alone low-power data collection/analysis (ensure sensor and algorithms functioning) and integration of sensors with the robots and deployment tests. Technical risks in a Sub-T environment include degraded communications, dynamic terrain, and environmental obstacles; all of which affect the ability to localize, position, and map the environment.

This research is part of the Geo-Intelligence and Complex Urban Environments (Geo ICUE) effort, which is an internally funded 6.2 effort at the Engineer Research and Development Center - Geospatial Research Laboratory (ERDC-GRL). Ultimately, this Geo ICUE research effort will involve a multi-modal mapping approach to include the use of high and low altitude airborne sensors as well as ground and subterranean based sensors and platforms. This diverse suite of exploitation tools will address capability gaps unique to dense urban terrain. The innovation expands beyond novel data collection approaches to the tiered integration and utilization of the resulting geospatial products (collected, processed, and exploited) at the point of need. Alignment with dynamic modeling and data visualization applications will enable comprehensive and tactically-relevant map coverage for DUAs, within both the desktop and mobile/hand-held computing environments.

2 Sensors

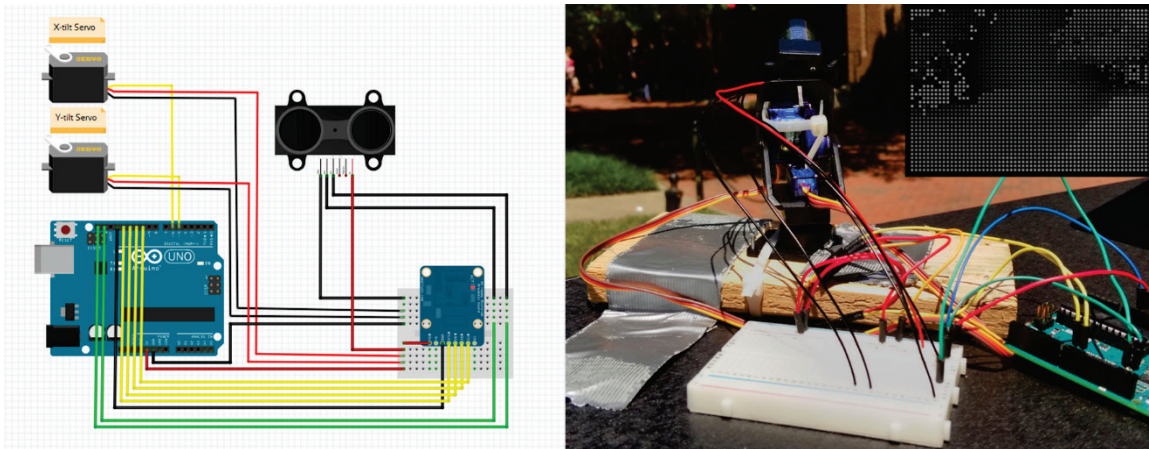
To accommodate challenging environments, exploratory and mapping robots are equipped with state-of-the-art sensors and programmed with powerful algorithms. LIDARs are a popular sensor for exploration and mapping due to their robustness in dark spaces and their pinpoint accuracy. Through the generation of point clouds, Lidars have produced detailed 3-D images of intricate spaces (Niemeyer et al. 2014; Börcs et al. 2017). Red, Blue, Green, and Depth (RGBD) cameras, a visual sensor that can detect both color in a scene as well as depth, are useful for exploring areas that are well-lit within the visible light spectrum (Yang et al. 2019). Varied sensors such as these, address a wide range of situations that many sensing systems, including robots, face. Through the combination of sensors, exploration and mapping can be conducted more robustly than otherwise (Castellanos et al. 2001; Zhang et al. 2012).

In this study, four different types of sensors were looked at; LIDAR, photogrammetry, light coding, and Time of Flight (TOF). The sensors used in this study are briefly listed below, with a more thorough description listed in Appendix A.

- Lidar: Garmin lite v3 (formerly LiDAR lite), SF30-C Laser Rangefinder, LEDdar VU-8, Hokuyo UST-20, Ouster OS-1 (16)
- Photogrammetry: DUO Developer Kit, ZED Stereo Camera, Realsense D415, D435, T265
- Time of flight: Microsoft Kinect Sensor (Xbox One)
- Light Coding: Structure Sensor, Orbbec Astra, Realsense D415, D435

Initially, the sensor evaluation began with the LIDAR Lite (currently the Garmin lite v3). The range finder was integrated with a pan-tilt module and connected to an Arduino board. The Arduino board was used to control the pan-tilt module and receive depth data from the LIDAR. The depth data was passed to a laptop running processing (<https://processing.org/>), which was used to view the data as a 3-D point cloud. An example of the setup and the resulting point cloud (inset) can be seen in Figure 1.

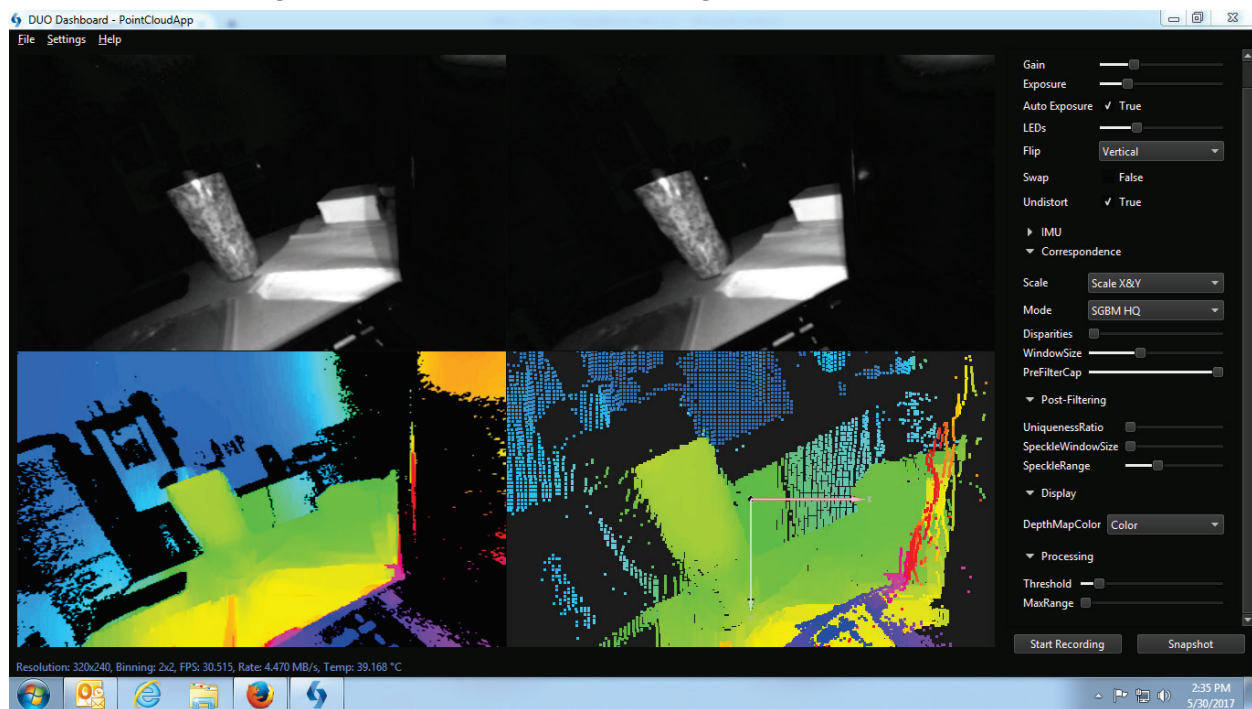
Figure 1. Arduino controlled LIDAR lite. The resulting point cloud is shown in the inset.



The range of the LIDAR lite was ~40 m. Switching to the SF30-C Laser Rangefinder increased the range from 40 m to 100 m. However, manually controlling a pan-tilt module with a microcontroller is quite slow. It took approximately 3 min to capture the point cloud located in the inset. Switching to the LEDdar VU-8 was also investigated, which increases the number of points per scan from one to eight, but it was clear something more efficient was needed.

Consequently, the attention was turned to photogrammetry sensors. The main advantage that the photogrammetry sensor has over the lidar configuration mentioned above is the entire scene is collected at once. Since the distance between the two cameras is fixed and the field of view of the cameras is known, depth is derived from the difference in the resulting images. The first photogrammetry sensor looked at was the DUO MLX. It was especially attractive since it was low power (2.5 W), utilized a global shutter, and contained an Inertial Measurement Unit (IMU). The global shutter reduced motion blur when the robot collected data while moving. This solution is in contrast to a rolling shutter where it might produce a blurred and warped image if the camera is moving. The integrated IMU could also be used to localize the camera and the robot. This camera, however, is a monochrome camera with an effective range of 1-2 m, limiting its overall effectiveness for mapping a Sub-T environment. An example of the point cloud collected with a DUO MLX is shown in Figure 2.

Figure 2. Stereo, depth and point cloud generated from a DUO MLX.



The ZED stereo camera is another photogrammetry sensor that uses two side by side 1920 x 1080 cameras (with a rolling shutter) and has a longer range than the DUO MLX. The effective range of the ZED stereo camera is ~20 m. It also has four resolutions to choose from ranging from Video Graphics Array (VGA) to 2k. However, due to hardware limitations, only VGA and HD720 resolutions could be evaluated. This camera also has two sensing modes to choose from, “sensing” and “fill.” The “sensing” mode is suited for applications such as collision avoidance and navigation, while “fill” is targeted at augmented reality and computational imaging where a fully dense depth map is needed. ZED's depth computation mode was also studied; specifically, “quality” and “performance” were compared. An example of the point cloud collected using the various resolutions and modes is shown in Figure 3. It is significant to note that the point clouds generated with the ZED have RGB data.

Figure 3. Point clouds generated from ZED photogrammetry sensor.



It was found that utilizing the “fill” sensing mode with the “quality” depth computation mode with the highest resolution possible, generated significantly better point clouds. In order to generate the highest resolution point clouds on a mobile robotics platform, it was investigated as if it was realistic to integrate a computer with a dedicated graphics card. It was eventually settled on a ZOTAC MAGNUS EN1080K that came equipped with an NVIDIA GeForce GTX 1080 graphics card. To power the system, two GoalZero Sherpa 100’s were purchased. However, during testing, the computer would randomly shut off. Also, it is significant to note that with the two external batteries powering the ZOTAC, the estimated run time is ~1 hr before the batteries discharged to critical levels. As a result, a different computer with lower power requirements that could still leverage a Graphics Processing Unit (GPU) for parallel processing was investigated. One example of a small form factor single board computers (SBC) with integrated graphics cards, is the NVIDIA Jetson TX2 motherboard (\$200), the preliminary results have shown that the TX2 gets quite hot when integrated with the ZED, even with an attached heat sink. Perhaps a more active cooling approach, such as a fan or closed-loop liquid cooling, would help and something that may be investigated at a later time. However, the computational costs and power requirements of the ZED were too steep to meet this demand.

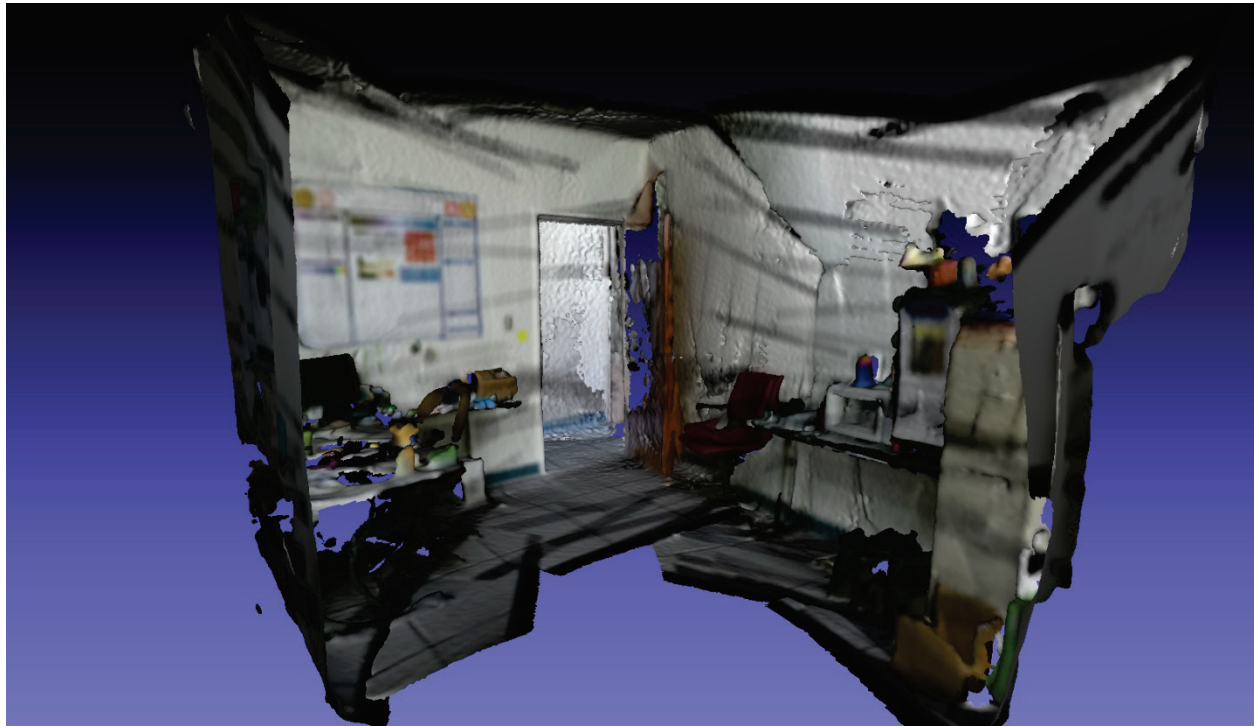
The next sensor evaluated was the Kinect Xbox One (aka Kinect version 2), which uses a wide-angle 640×480 TOF camera to render depth. TOF works by resolving the distance between the camera and the subject by measuring the round trip time of a photon for each point of the image. An example of the depth image from the Kinect One is shown in Figure 4.

Figure 4. Depth image generated from Kinect Xbox One.



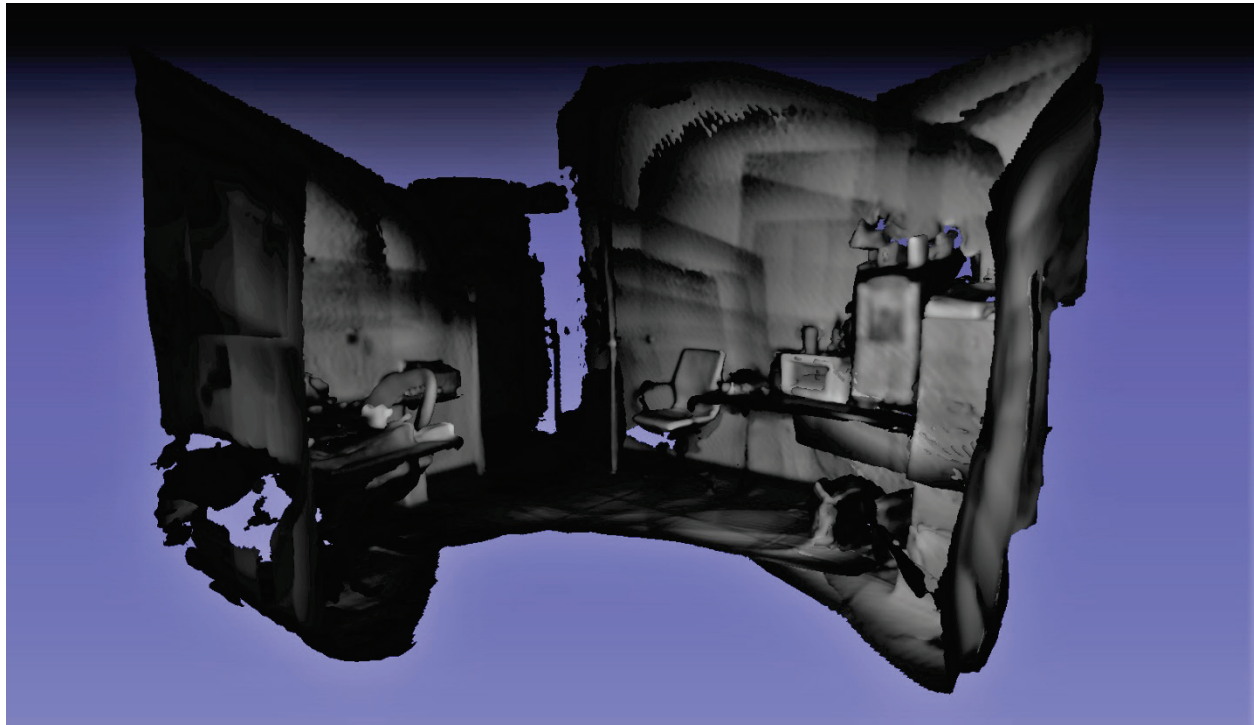
The Kinect v2 is also equipped with a 640 x 480 RGB camera, which allowed for overlaying the RGB values on the depth profile, which resulted in an RGB point cloud. An example of an RGB point cloud collected with the Kinect v2 is shown in Figure 5.

Figure 5. RGB point cloud collected using Kinect Xbox One.



The Kinect is also equipped with an Infrared Radiation (IR) emitter, which in conjunction with the TOF camera, is the main advantage of this technology. The TOF camera allows the Kinect v2 to operate in complete darkness. The point cloud in Figure 6 was collected in the dark with the IR image, instead of the RGB image, overlaid onto the depth profile. Unfortunately, the Xbox Kinect One was discontinued in October 2017.

Figure 6. Infrared point cloud collected with Kinect Xbox One.



The final suite of sensors evaluated was based on light coding technology. The process codes a scene with near-IR light (810 nm, just outside the visible spectrum). The image sensor reads the coded light back from the scene using various algorithms to triangulate and extract the 3-D data. The advantage of using this method over a traditional painting LIDAR is that the entire scene (307,000 data points) is collected every half a second. A potential drawback of this technology is its inability to operate in direct sunlight. However, since the focus was on SubT environments and indoor areas, this was not a significant concern. One example of a sensor that leverages this technology is the Astra by Orbbec, which is equipped with a 640 x 480 camera. An example of a point cloud collected with an Astra is shown in Figure 7.

Figure 7. RGB point cloud collected with and Orbbec Astra.



It is significant to note that the Astra is powered by a single USB connection and doesn't require a high-end graphics card. Also, the point clouds generated by the Astra had the fewest deviations and didn't require post-processing to remove outliers when compared to the photogrammetry counterparts. In regards to SWAP-C considerations, it appears that light coding sensors are the best match for this research. The Structure Sensor is another light coding based sensor that was attractive since it had its own battery pack. However, the integration of an internal battery did not outweigh the advantage of using the Astra, which contained a built-in RGB sensor.

In January of 2018, Intel released a family of sensors that leverages both light coding and photogrammetry. This combined capability provides the superior accuracy observed with light coding, and the ability to map brightly lit environments, sunlight, using the stereo cameras. Intel released two sensors, the D415 and the D435. The Realsense D415 features two side by side stereo cameras as well as an IR emitter/sensor combination. The optical sensors have a resolution of 1920 x 1080, and the depth sensor has a resolution of 1280 x 720. Depth readings come in at 90 fps, while the optical readings come in at 30 fps. The Realsense D435 has a similar setup as the Realsense D415, but utilizes a global shutter

instead of the D415's rolling shutter. The advantages of using a global shutter were discussed previously. An example of the point cloud collected with the Realsense D435 is shown in Figure 8. Going forward, since either the Astra or D435 is used throughout the rest of the paper, their respective setup procedures are listed in Appendix B.

Figure 8. RGB point cloud collected using the RealSense D435.



3 SLAM Modules

SLAM robots are often programmed to follow algorithms that leverage very diverse sensor readings. Iterative Closest Point (ICP) uses a closest-neighbor comparison for the point cloud data to determine the most similar fit between two given scans. These changes can then be used to determine the movement of the robot, provided the sensors performing measurements do not shift in position unexpectedly (Wen et al. 2013; Shi et al. 2017). Another popular technique in SLAM robotics is the use of visual and non-visual odometry for loop closure. To determine if an observed space has been previously explored, many robots rely on their sensors to detect phenomena that would indicate a full rotation of the robot, thus prompting the robot to observe the same scene as before. Previously detected features (Jacobson et al. 2018) and camera images (Yang et al. 2019) can indicate a closed loop for the exploratory robot. Some of these robots can even perform these tasks in both dynamic as well as static environments (Yang et al. 2019). Algorithmic solutions like these further extend the capability of SLAM robots as highly accurate and independent exploratory agents.

Now that potential sensors have been identified, namely the Orbbec Astra or RealSense D435, that will do the heavy lifting for the generation of RGB 3-D point clouds, it was time to look at SLAM. In an unknown environment, SLAM would be used to map the environment and provide information in regards to the robot's current position. There are several open-source SLAM modules that are available online that have built-in support for the light coding (aka RGB-D) sensors. The SLAM modules that were evaluated were ORBSLAM2, DSO_SLAM, ElasticFusion, Co-fusion, and RTAB-Map. The installation procedures for the various SLAM modules are provided in Appendix C.

ORB-SLAM2 is a real-time SLAM library for Monocular, Stereo, and RGB-D cameras that compute the camera trajectory and a sparse 3-D reconstruction (in the stereo and RGB-D case with true scale). It can detect loops and localize the camera in real-time. Thus, it is an ideal candidate for deployment in unknown areas where time is limited and accuracy is important. There are a number of dependencies that had to be met for ORBSLAM2 to work. For instance, OpenCV, a library for graphics, had to be installed from source code. Also, packages were required to aid with graphics computing. Packages such as g2o, gtsam, and Eigen were

installed. Pangolin was installed to manage OpenGL displays, and freenect2 was installed to handle device drivers. ORB-SLAM2 worked very well when the goal was to localize the position of the camera quickly. The point clouds ORB-SLAM2 creates are not ideal for a detailed analysis of the surrounding environment. As a result, this module would be best as a dead reckoning navigation tool with a previously generated map. Also, it is significant to note that the code has to be modified in order to save the generated point cloud.

The next SLAM module that was investigated was DSO_SLAM, which stands for Direct Sparse Odometry SLAM. DSO_SLAM is another real-time SLAM program that uses only monocular cameras to create camera trajectories and reconstructs the surrounding environment. It is also able to detect loops and is able to relocalize the camera in real-time. As far as dependencies, Suitesparse is a package for Compute Unified Device Architecture (CUDA) enhanced matrix computation, and eigen3 is another library used for easy programming with matrices. OpenCV was also installed alongside Pangolin, and both are used to help with 3-D graphics and the Graphic User Interface (GUI). Ziplib is an optional install that allows the user to operate off of image datasets. For integration into the robot DSO_ROS was also installed; written by the same author as DSO, to integrate DSO_SLAM into a Robot Operating System (ROS) functionality. Unfortunately, the program did not work as advertised and DSO_SLAM was unable to initialize or produce any point clouds with a live camera feed. However, DSO with pre-collected datasets was able to be tested. Overall, the point clouds that DSO_SLAM produced were denser compared to those using ORBSLAM2, and the program has the benefit of saving the point cloud directly to the disk afterward. The downside to this module is that the cameras have to be calibrated before it can be used.

ElasticFusion was investigated due to the high quality point clouds it could create at near real-time speed. However, to generate these extremely dense point clouds a GPU capable of over 3.5 Teraflops of processing power had to be leveraged. The computer of choice for this SLAM module was Zotac PC that wielded a GeForce GTX 1080 graphics card. This computer had two power inputs, one to power the CPU and one to power the GPU. The computer was set up with Ubuntu 16.04, and the latest CUDA from Nvidia was installed. Qt-GUI was installed for GUI handling, libusb and libudev to handle USB inputs from cameras, openjdk for the Java language, freeglut as an OpenGL alternative to 3-D rendering, libglew to extend the

functionalities of OpenGL, and suitesprase, and eigen3 were also installed. Also, Pangolin was installed with jpeg and png support, and OpenNI2 was installed to handle input from devices such as the Astra. Finally, ElasticFusion core, GPUPTest, and then GUI were compiled. The GPUPTest was run to analyze the processing power of the computer and GPU, those values were then pasted into a file containing GPU specifications. The ElasticFusion core was rebuilt using the new modified file, and to launch ElasticFusion, and the GUI was run. ElasticFusion works well in capturing near realistic point clouds of the surrounding environments with just a single RGB-D camera. The only disadvantages would be the massive power consumption and that ElasticFusion only accepts input from a singular camera, meaning sensor fusion for localization and positioning requires significant edits to the source code. One possibility to alleviate running a high performance GPU on the robot itself was to transmit raw depth data from an onboard sensor. However, since network communication is a significant concern in a Sub-T environment, this would be a less than ideal solution. An example of a point cloud collected using ElasticFusion is shown in Figure 9.

Figure 9. High Density point cloud collected using ElasticFusion.

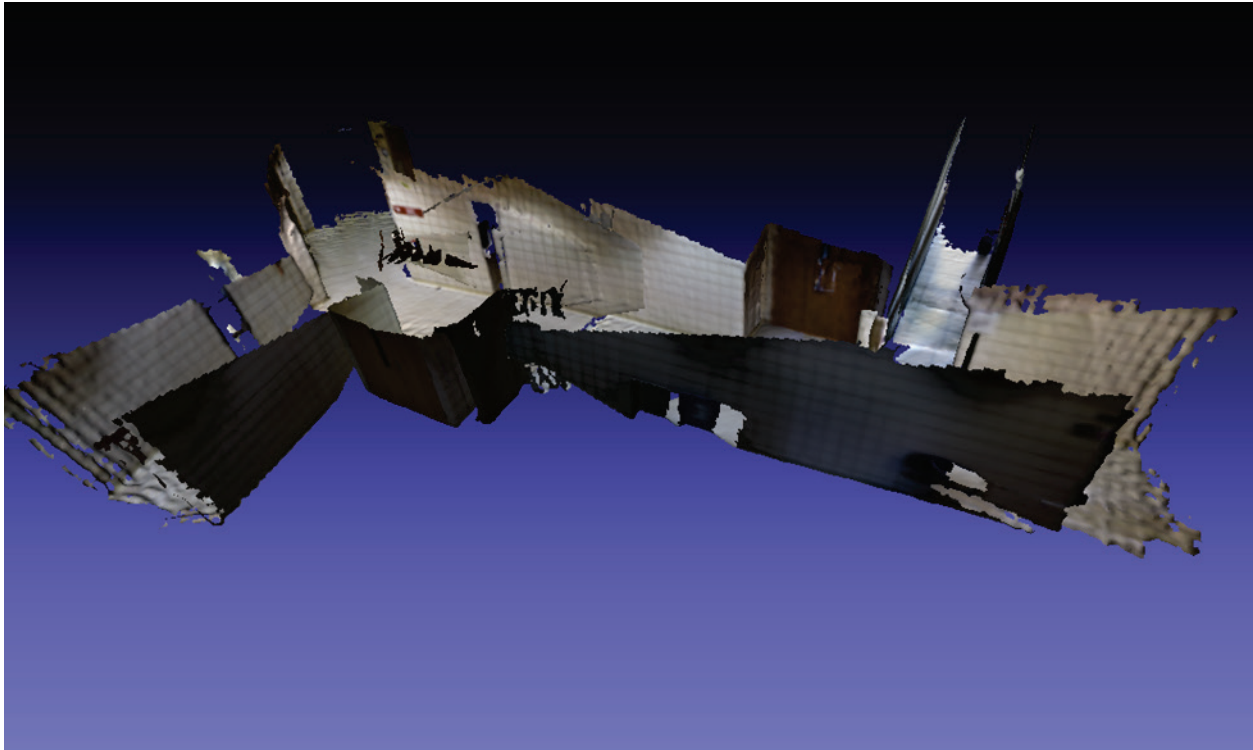


Co-Fusion is another SLAM module built off of ElasticFusion with an added benefit. Co-Fusion has the ability to track moving models within the field of vision. This ability is extremely useful in situations such as an urban

environment where there are moving vehicles and people. Typically, moving entities would disrupt the point cloud collection and show up as noise in the final point cloud. Co-Fusion is able to track these moving objects and remove them, resulting in cleaner data collect. Since Co-Fusion is based on ElasticFusion, most of the libraries were already installed. However, Co-Fusion installs the latest CMake to build its programs and updates the c++ compilers. Also, it installs zeroc-ice35, which is a networking tool. The build for Co-Fusion is exactly the same as the build for ElasticFusion, and the GPUPTest values are the same as well. The main advantage of Co-Fusion is the ability to track moving objects in urban environments. However, that comes with the downside of requiring massive amounts of RAM to keep track of all the models as well as the point cloud, which would sometimes cause the program to crash expectantly.

The final SLAM module that was tested was Real-Time Appearance-Based Mapping (RTAB-Map). RTAB-Map is a Graph-Based SLAM approach based on an incremental appearance-based loop closure detector (Labbé and Michaud 2019). This module was the most interesting to use as it had the most integrations with ROS as well as sensor inputs. The base RTAB-Map module was installed from Ubuntu's package manager as well as the ROS RTAB-Map integration. GTSAM and g2o were installed as modules to assist with loop closure. Freenect2 was installed to assist with using Kinect based cameras such as the Kinect or Astra cameras. An example of a point cloud collected using RTAB-Map is shown in Figure 10.

Figure 10. Semi-dense point cloud collected using RTAB-Map.



Since RTAB-Map is currently the preferred SLAM module, the next section provides a more in depth look in regards to optimization. There are many different parameters contained in the launch file, which changed depending on what configuration was being used. Typically, the parameters were set as follows:

The first parameter to notice is about the saved .db file.

```
<node name="rtabmap" pkg="rtabmap_ros" type="rtabmap"
output="screen" args="--delete_db_on_start">
```

--delete_db_on_start sets RTAB-Map to delete the previous database before starting. There are ways to allow RTAB-Map to continue adding to a previous map; however, this was not done. Launching RTAB-Map will overwrite ~/.ros/rtabmap.db, so be sure to rename or save the .db elsewhere after a scan to prevent accidental data loss.

```
<param name="frame_id" type="string"
value="base_link"/>
```

This line lets RTAB-Map know what base frame for the robot. The default is `base_link` in accordance with ROS best practices. The frame `base_link` is rigidly attached to the mobile robot base and the robot sensors' coordinate frames are transformed into the `base_link` frame. The robot determines its location in the world by the transform from the `odom` frame to the `base_link` frame; likewise, the same can be done from the `odom` frame to the map or world based frame.

```
<param name="odom_frame_id" type="string"
value="RLOdom" />
```

This line lets RTAB-Map know the odometry frame the robot is using. This frame **MUST** be accurate as misnamed frames will cause RTAB-Map to lose odometry and fail to navigate accurately. This line will be changed depending on the robot configuration and the node used to compute odometry. In this example, it is set to `RLOdom`, the `odom` frame used by `robot_localization`.

```
<param name="subscribe_depth" type="bool"
value="true"/>
```

This parameter tells RTAB-Map to use a depth image for mapping. This is set to `true` so RTAB-Map can use the depth image from the D435 to create a point cloud. If depth images are not used for a scan, this should be set to `false`.

```
<param name="subscribe_scan_cloud" type="bool"
value="false"/>
```

```
<param name="subscribe_scan" type="bool"
value="true"/>
```

3-D lidar exports a scan cloud, 2-D lidar exports a scan. Set the logic accordingly for whether 2-D or 3-D lidar is used.

```
<param name="subscribe_odom_info" type="bool"
value="false"/>
```

The default value of “`true`” had some issues as it caused RTAB-Map to fail when no odometry info was published. It is used when RTABP creates its

own odometry through visual odometry, which was never used, so it was set to false. If visual odometry through RTAB-Map is ever used, set this to true or remove the line entirely.

```
<remap from="odom" to="/odometry/filtered"/>
```

This line should be changed to reflect the odometry topic used. Robot localization publishes to the /odometry/filtered topic while the T265 publishes to /cam_t265/odom/sample. Update this accordingly to ensure RTAB-Map uses the right odometry. This topic should be the odometry topic corresponding to the node that is publishing the odom frame. See section “Frames” for more info.

```
<remap from="scan" to="/scan"/>
```

Sets the 2-D scan topic for RTAB-Map. The Hokuyo lidar publishes to the /scan topic. This should be set to whatever topic the robot’s 2-D lidar is publishing. If 3-D lidar is being used, this line should be removed; it may or may not cause interference.

```
<remap from="scan_cloud" to="/os1_cloud_node/points"/>
```

Sets the 3-D scan cloud topic for RTAB-Map. The Ouster 3-D lidar publishes to the /os1_cloud_node/points topics. This parameter should be set to whatever topic the robot’s 3-D lidar is publishing. If only 2-D lidar is being used, this line should be removed; it may or may not cause interference.

```
<remap from="rgb/image"  
to="/cam_d435/color/image_raw"/>
```

```
<remap from="depth/image"  
to="/cam_d435/aligned_depth_to_color/image_raw"/>
```

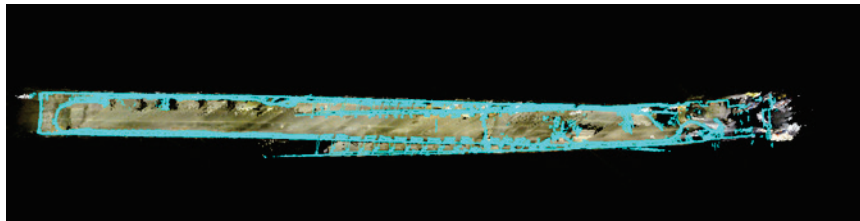
```
<remap from="rgb/camera_info"  
to="/cam_d435/color/camera_info"/>
```

These parameters should be set to match the corresponding topics published by the depth sensor, in this case, the D435. Examples of launch files can be found in Appendix D.

RTAB-Map can be used in multiple sensor configurations that will be explored in depth. It requires a depth camera and an odometry input. For depth, RTAB-Map can use a light-coding camera, a stereo camera, or a combination of both. Light-coding cameras have the ability to transmit invisible infrared light and drawn patterns on objects. By analyzing the patterns using its infrared camera, the light-coding camera then can gather the 3-D information of the objects in front. On the other hand, stereo cameras have more than one lens. They can estimate the depth of objects by using the same method the human eye uses called binocular vision, a way to estimate depth by overlapping the vision of an object from a different angle.

The most basic odometry input is visual odometry. Visual odometry is a way to estimate the position and orientation of a robot by analyzing features on camera images. From a SWAP-C perspective, visual odometry is ideal since it only requires a depth sensor. However, in an environment that has little or no visual features, such as edges, corners, or in darkness, visual odometry fails. Alternatively, odometry input can also be generated from a 2-D LiDAR such as the Hokuyo UST-20. Specifically, the UST-20 has a max range of 60 m and rotates at 2400 rpm. Generating odometry from a laser is much more reliable than visual odometry in most situations. However, since this technique utilizes a spinning lidar, drift can occur if it turns too fast. For robots equipped with wheel encoders, their wheel encoders can be leveraged to provide odometry. During rapid turning, wheel-based encoder odometry tends to perform more accurately than laser or visual odometry. However, the wheels (or treads) slip, while moving and accumulate error over time. Over short distances, wheel odometry is ideal, but is prone to errors the longer it runs. An example of drift using only wheel encoders as an odometry target can be seen in Figure 11.

Figure 11. Point cloud collected using only wheel encoders.



Finally, odometry can also be generated by fusing data from sensors such as IMU and wheel encoder. Another approach is to fuse measurements from different sensors into a ROS node called `robot_localization`. However, it is important to note that RTAB-Map was by far the optimal choice for integration onto a robotics platform and is used through the rest of this report. First and foremost, the point clouds were significantly dense and easily identified details in a given scene. Also, RTABAMP didn't require a GPU and ran quite comfortably on an Intel Next Unit of Computing (NUC) equipped with an i5 processor. Finally, RTAB-Map is highly integrated into the ROS framework, which leads into the next topic of sensor fusion.

4 Sensor Fusion

The `robot_localization` ROS package provides easy-to-use implementations of the extended Kalman filter (EKF) and the unscented Kalman filter (UKF). Since the UKF is a bit more complicated, it has not been tested. From this point forward, only the EKF will be discussed. `Robot_localization` requires two files: the “launch file” and the “parameter file.” The launch file is simple, composed of three lines, and the only required modification was changing the pathname of the parameter file. The parameter file, on the other hand, is more complex. At the top of the parameter file is “frequency,” which is the output rate of the EKF. The output topic “odometry/filtered,” utilizes this frequency. Generally, 20 Hz was used. Anywhere from 20-30 Hz is typical, but lower than that is not recommended. The second parameter, `sensor_timeout`, determines how long to wait before the sensor should be regarded as being timed out. The units of this parameter are seconds. After this amount of time without getting a value from a sensor, the EKF will perform the predict step, but not the update step. This value was not modified from the default, but it is important to note. The third parameter, `two_d_mode`, allows the user to specify that the robot is moving on a plane. For this project, this was generally set to false, since the goal was to be able to map on rough terrain. There are several more general parameters available, but they were not deemed essential to this project.

There are four frame parameters in `robot_localization`: “`map_frame`,” “`odom_frame`,” “`base_link_frame`,” and “`world_frame`.” The first three should be set to the system’s appropriate frame names, such as “`map`,” “`odom`,” and “`base_link`,” which are the defaults. “`World_frame`” should be set to the same name as the “`odom_frame`” or “`map_frame`,” depending on which frame is providing positioning data for the EKF.

A frame is a 3-D coordinate system in ROS that follows the right-handed rule. Transforms between frames is done with the ROS `tf` library. The relative pose between two frames is represented as a translation and rotation. A single frame can be thought of as a frame of reference with its own individual coordinate system. Similar to how the earth is in the solar system, which is in the galaxy, frames have a hierarchy. A parent is the larger frame, which contains a smaller frame (the child frame). For example, the earth is a child frame for the parent frame of the solar system. The solar system is a child frame of the parent frame the galaxy.

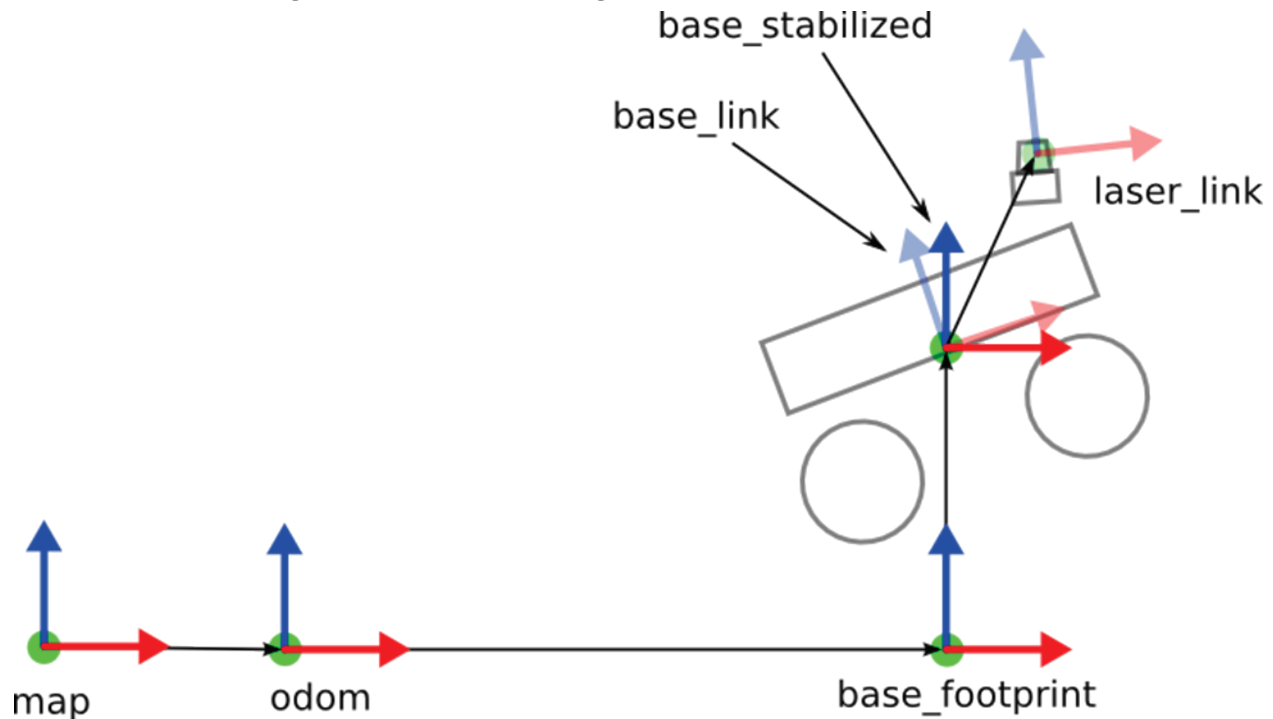
The hierarchy would be galaxy > solar system > earth. Frames and transforms are used to translate sensor readings on the robot to a single reference frame, usually the `base_link` frame, and for determining the location of the robot in both a robot-centric coordinate system (`odom`) and world-centric coordinate system (`map` or `world`).

`Map` is the frame at the top of the hierarchy for these purposes. If GPS is integrated then this may change to include an earth frame above `map`. The `map` frame contains the room or location being mapped, this area should not move relative to the `map` frame. The `map` frame determines the robot location and orientation on a given map as well as the location of other objects (such as obstacles) on that map.

The `odom` frame is a child frame of `map`. It is the frame on which the robot moves continuously. Robot localization publishes the location of the robot within the `odom` frame using sensor data. Odometry within this frame is accurate within a short period of time, however as drift accumulates, it loses accuracy. RTAB-Map accounts for drift through loop closure and resets the accumulated error. As the robot drifts, RTAB-Map calculates the drift and moves the `odom` frame within the `map` frame to account for it.

The `base_link` frame is a child of the `odom` frame and contains the robot. The robot does not move relative to `base_link`, and all the sensors are configured as transforms relative to `base_link`. An example is shown in Figure 12. Robot localization publishes the transform between `base_link` and `odom` using sensor data.

Figure 12. Schematic showing the relationship of frames in ROS.



In general, `world_frame` should be set to “odom” when the localization is done by sensors on the robot, continuously taking measurements so that it can perform the `odom`→`base_link` transform. The `world_frame` should be set to “map” if the measurements are taken discretely and externally, such as GPS or UWB. In this case, `robot_localization` does the `map` > `odom` transform, and another node needs to perform the `odom` > `base_link` transform. The “odom” frame is where the robot’s movement is measured. Robot localization treats this as the ground the robot drives around on. `Base_link` is the base of the robot. As the robot moves, robot localization publishes a transform between `base_link` and `RLodom`. The name was changed from `odom` to `RLodom` to prevent conflicts with other `odom` frames being published and to have a better name for the frame.

There are four types of sensors that can be specified in `robot_localization`, which correspond to the four possible message types. The parameters are “odom0,” “pose0,” “twist0,” or “imu0,” with the number being incremented if there are multiple of that type. The value of the parameter should be set to the topic that the sensor is publishing. For instance, to specify the Pozyx IMU using the reference code, one would type, “odom0: Pozyx_imu.” The following config parameter is indicative of “odom0_config.” The value should be a length 15 Boolean vector, but formatting it visually as a 5 x 3

matrix makes it easiest to understand. Each Boolean value specifies whether or not that sensor should contribute to the EKF for that given state variable. The structure of the vector is:

```
[ x, y, z,

  roll, pitch, yaw,

  vx, vy, vz,

  vroll, vpitch, vyaw,

  ax, ay, az]
```

Specifically, the GVR-Bot odoms were set as follows:

```
odom0: /gvrbot/odom

odom0_config: [true, true, false,

               false, false, true,

               true, false, false,

               false, false, false,

               false, false, false]

odom0_queue_size: 4

odom0_nodelay: false

odom0_differential: true

odom0_relative: true
```

Here, the /gvrbot/odom is the topic published for the GVR-Bot's wheel odometry. The true/false logic is explained in the robot_localization documentation. They correspond to (X,Y,Z,roll,pitch,yaw,X',Y',Z',roll',pitch',yaw',X'',Y'',Z''). The true/false

logic can be determined through rostopic echo as well as observing the robot. For example, this robot could drive in the x direction as well as rotate; therefore, the measured velocity wasn't used in the y or z directions.

For the T265 Camera Odom, the structure was as follows:

```
odom1: /cam_t265/odom/sample

odom1_config: [true, true, false,

               false, false, true,

               true, false, false,

               false, false, false,

               false, false, false]

odom1_queue_size: 4

odom1_nodelay: false

odom1_differential: true

odom1_relative: true
```

Altering the true/false logic may improve the accuracy of robot_localization, but care must be taken to ensure sensors are only set to input data that is measured accurately. For example, GPS is notoriously inaccurate for measuring altitude and is typically not used. To avoid using altitude from the GPS, the z value would be set to false. Also, note that the T265 IMU axes are different from what's expected by ROS. The Y axis is vertical and measures yaw as opposed to the Z axis (standard). This can be accounted for by changing the orientation in the transform from the T265 IMU to the base_link.

If the third parameter, “odomodifferential,” is set to true, robot_localization takes the position values (x,y,z) and the change in time (Δt), approximates the velocity and uses that in the Kalman filter. Generally, velocities perform better when doing odometry than position. The next parameter is “odomorelative.” This value initializes the first input to robot_localization as the origin, so the odometry starts from zeroes. This was specified as true for every sensor, as it was a convenient way to keep everything uniform. The “queue_size” parameter specifies how many values from the sensor to hold on to for use in the next predict/update step of the EKF and is useful if the sensor has a much higher update rate than the output frequency of the EKF. The next parameters are “rejection_threshold” parameters. These vary for each message type/sensor type used. They can be tuned to reject noisy data, but thus far have not been required for this project. They were commented out, which sets them to the maximum. There is one additional parameter specifically for the IMU, which is the “remove_gravitational_acceleration” parameter. This removes gravity’s contribution to the accelerometer data, which was set to false.

After the sensor specifications, there is a section of advanced parameters. Most of them were not modified for this project. However, the “use_control” parameter was set to true, which took whatever control signal robot_localization detected and used it to influence the state transition function. This parameter is helpful in some cases because it biases the EKF towards the wanted position for the robot; however, the control signal can only describe the desired, projected state. This was set to false for these tests, but it could be experimented with. At the bottom of the parameter, files are two 15 x 15 matrices. The first is the process noise covariance matrix, which describes the randomness in the system, be it caused by inconsistencies in the motors, sensors, environment, or voltage levels of the circuits. The process noise covariance is just the covariance of that randomness. It will be different in every environment and every robot/sensor setup. This matrix was not tested or modified, but it could be used to more finely tune the EKF’s precision and convergence rate. The last matrix is the initial state estimate error covariance matrix. This matrix is dynamic and will increase in error with every predict/update step of the EKF. It describes the predicted error in the EKF’s state estimate. The initial values can be tweaked to affect how fast the EKF converges, but this has not been tested, nor has the need been seen. The performance of the robot has been sufficient.

One of the goals in mind during this testing was to be able to use `robot_localization` with full 3-dimensional space. This usage requires six degrees of freedom (6DoF), including x, y, z, roll, pitch, and yaw. Whether or not `robot_localization` was outputting in 6DoF using the LiDAR scan visualization when running RTAB-Map was able to be tested. If the robot was tilted and the scan tilted accordingly, then it worked. It was also discovered that `robot_localization` will not work with only an IMU as an odometry input. This makes sense logically: the IMU has no idea of any linear pose estimate since it can only determine the velocities and acceleration between poses. An IMU cannot be used as the sole source of odometry. To test the effectiveness of this approach, NVESD's 60 m long firing tunnel was mapped. This test was particularly challenging since the walls are flat and painted black, which induces stress into the system since SLAM relies on visual cues (edges and corners) to localize. Therefore, in order to localize and position, the robot had to rely on the sensor fusion mentioned above and established using `robot_localization`. Figure 13 shows a top-down view of the 60 m firing tunnel collected by fusing the following sensors: RGB-D, 2D LiDAR, IMU, and wheel encoders. In this low complexity environment, odometry was able to be maintained, which resulted in minimal drift.

Figure 13. Top-down view of NVESD's firing tunnel.

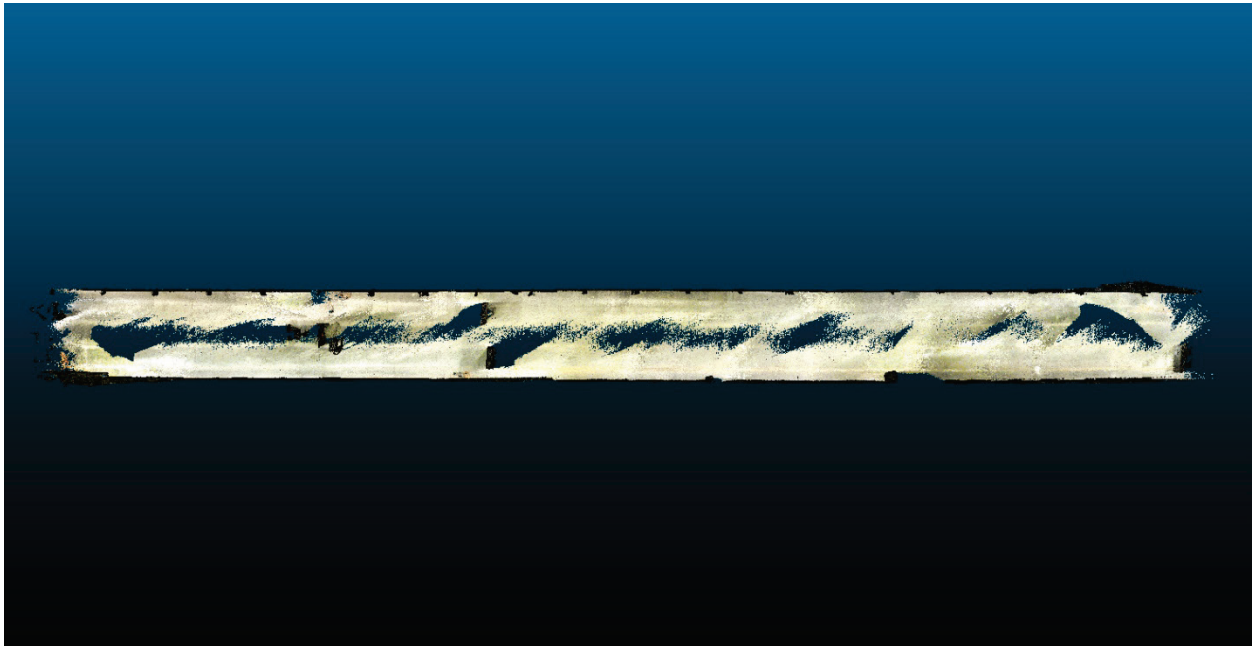
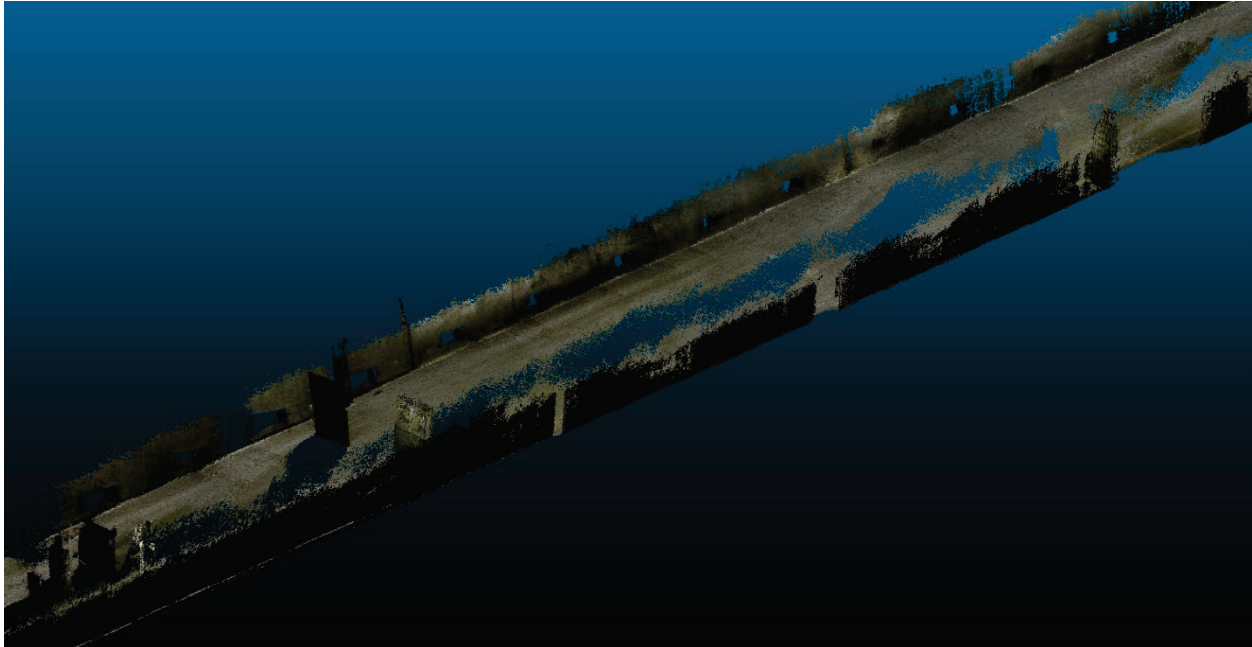


Figure 14 shows the same scan with a different viewpoint highlighting the details of some features collected in the tunnel.

Figure 14. NVESD's 80m long firing tunnel.



Typical configurations used are listed below:

1. T265, D435, RTAB-Map

This is a good combination to show off the T265's odometry and demonstrate handheld mapping. The T265 provides odometry to RTAB-Map and the D435 provides depth image/point cloud to RTAB-Map. RTAB-Map uses visual loop closure from the D435.

2. T265, D435, Kobuki, robot_localization, RTAB-Map

This was one of the most accurate combinations. The grip of the Kobuki base helped provide accurate wheel odometry supported by the precise odometry output from the T265. Both of these odometry inputs were fed into robot_localization, which then published odometry to RTAB-Map. The D435 still provides depth image and point cloud data to RTAB-Map for mapping and loop closure. Another addition to this set up would be adding the Hokuyo lidar as a scan input to RTAB-Map and using ICP loop closure.

3. T265, D435, GVR-Bot, Ouster, robot_localization, RTAB-Map

This is the current setup. It is similar to the previous one, only using the Ouster 3-D lidar instead of the 2-D Hokuyo lidar and the GVR-Bot instead of the Kobuki base. There are some notable issues with accuracy, possibly due to drift from the GVR-Bot wheel odometry.

In order to evaluate accuracy, the point clouds generated (interior) were overlaid to those collected with a surveyed terrestrial LiDAR data (Leica C10, exterior). The root mean square (RMS) when aligned to world coordinates was determined to be 6cm, which is the average error that any given point could have after being transformed to world coordinates. Figures 15 and 16 show the overlaid point clouds.

Figure 15. Collected point cloud overlaid with Leica C10.



Figure 16. Different perspective of overlaid point cloud.

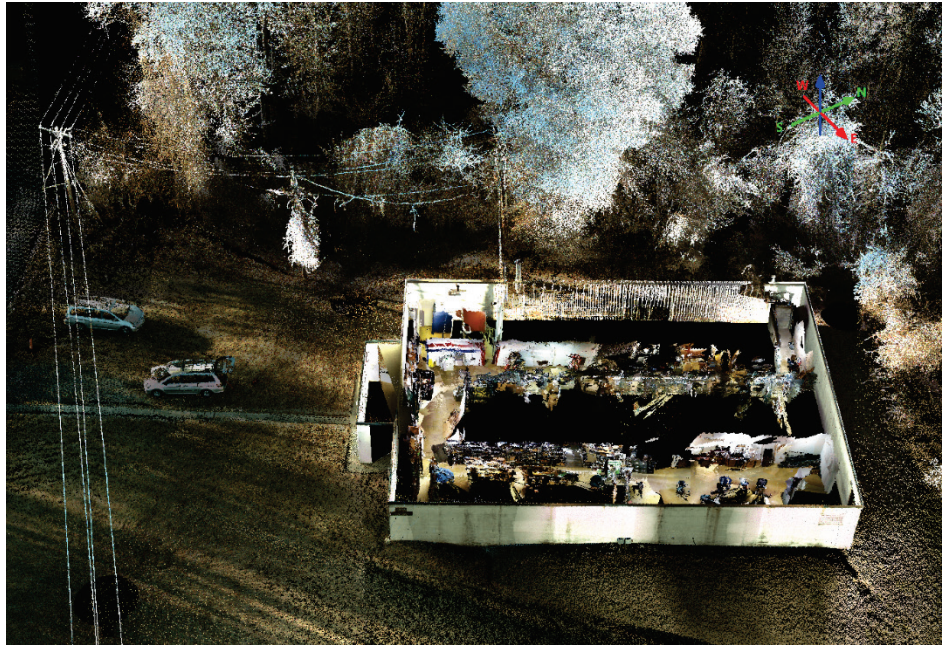


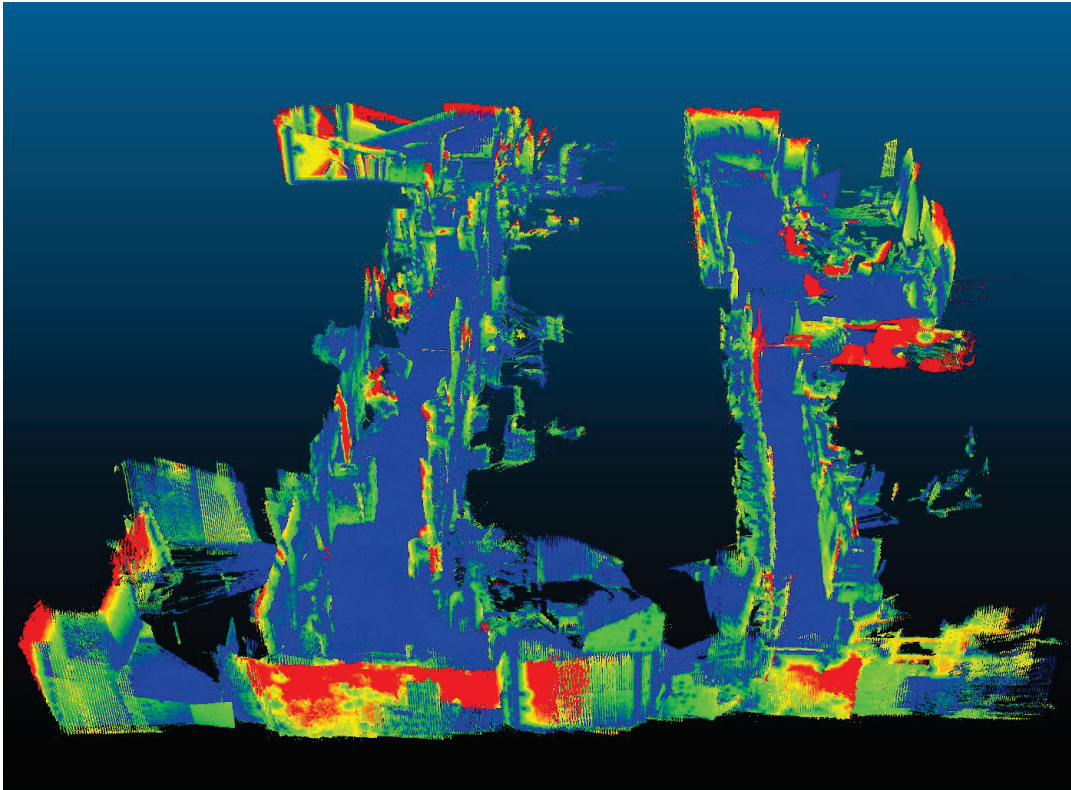
Figure 17 shows a portion of the overlay zoomed into to show finer details. The reference point cloud is grayscale and the point cloud collected with the robot is in color. Since the sensor on the robot is close to the ground, it doesn't capture the details on top of the desk.

Figure 17. Detailed overlay of collected point cloud.



Aside from doors and chairs being moved between scanning, the heat map shown in Figure 18 shows the most significant deviations are near the ceiling. This again is due to the sensor being low to the ground. Raising the sensor on the robot platform should remediate this issue and improve the RMS value.

Figure 18. Heat map of overlaid point cloud.



5 Virtualization

To rapidly test out new ideas and evaluate different algorithms, the authors set out to create a simulation environment that could adequately represent the robot while still using ROS. The main goal of this work was to create a virtual world that could interface with ROS and provide sensor measurements from this virtual environment to test out mapping algorithms (such as RTAB-Map and LeGO-LOAM) that would eventually be used on the real robot. In this section, the process of running this simulation environment and how things were set up to aid with future development or building new case studies are explored.

The robot was simulated using Gazebo, a simulation environment that integrates nicely with ROS. There are quite a few tutorials on the Gazebo website to help with the basics and getting started. The GVR-Bot simulation package was the main package for running the simulation. The package modified the work from the West Point repository. The launch folder included all the different startup files used by the simulation. The models folder contained a description of all the Unified Robotics Description Format (URDF) descriptions of the robot in the robot's subfolder and all the mesh used for visualization in the meshes subfolder. The worlds folder contained the Gazebo model and world files. The src and include folders included the C++ and header files that make the Gazebo plugins.

To install the simulation, copy the GVR-Bot simulation and hector gazebo plugins to your catkin workspace, then proceed with building the packages by doing catkin make. The main point of entry for the simulation is the GVR-Bot gazebo.launch file in the launch folder of GVR-Bot simulation. A bare-bone example of running the Gazebo simulation is by typing:

```
$ roslaunch gvrbot simulation gvrbot gazebo.launch
```

This basic launch command runs Gazebo and loads up a GVR-Bot model with a 2-D LiDAR in an empty world. A GUI interface is also loaded for moving the robot around. The launch file has several options that can be adjusted depending on the simulation that need to run:

- control - to specify how to control the robot. Options are
 - gui (default), which launches rqt robot steering

- key - uses teleop twist keyboard to control the robot using the keyboard. This requires the terminal to be selected to move the robot.
- joy - launches joy node from joy package and teleop twist joy to control the robot using a joystick.
- Lidar - to specify which LiDAR sensor to simulate. Options are
 - 2d to simulate the 2D LiDAR sensor.
 - 3d to simulate the 3d LiDAR sensor.
- world - to specify which world to in the gazebo simulation. To ensure that all the worlds can be loaded properly, the Gazebo models used in these worlds need to be copied from the /worlds/gazebo models folder to the location of the Gazebo models on your local machine (which is usually located in /.gazebo/models/). Options are:
 - empty - (default), which does not load any world.
 - Café - which loads cafe.world upon startup. Make sure that the model folder cafe in /worlds/gazebo models is copied to the Gazebo models on your local machine.
 - Office - which loads willow garage.world upon startup. Make sure that the model folder willow garage in /worlds/gazebo models is copied to the Gazebo models on your local machine.
- mapping - to specify which mapping strategy to load for the simulation. Options are:
 - none (default) - to not load any mapping algorithm during simulation.
 - RTAB-Map - to specify that RTAB-Map is being used for mapping. Depending on which LiDAR sensor is used, the launch file will either include /launch/rtabmap/rtabmap 2d lidar.launch or /launch/rtabmap/rtabmap 3d lidar.launch
 - lego loam - to specify that lego loam is being used for mapping. This should only be used with the 3-D LiDAR setup since LOAM requires a 3-D lidar.
- rtabmap_viz - a true (default) or false flag specifies whether or not RTAB-Map viz
 - should be loaded when RTAB-Map is used as the mapping option.
- rviz - a true (default) or false flag specifies whether or not rviz should be loaded when either mapping option is enabled.

A more complete example for launching the simulation with different parameters:

```
$ roslaunch gvrbot simulation gvrbot gazebo.launch  
control:=joy world:=cafe lidar:=3d mapping:=rtabmap
```

The GVR-Bot gazebo.launch file processes the URDF representation of the robot in /models/robots to generate the robot model to be passed to Gazebo. The URDF parsing starts with /models/robots/gvr_bot.urdf.xacro which loads:

- /models/robots/gvr_base.urdf.xacro, that describes the visual and colliders of the GVR-Bot base and wheels. The track on the robot is simulated using 6 wheels on each side. This file also implements an IMU sensor, an odometry publisher, and the diffdrive plugin multi wheel.cpp from hector gazebo plugins to drive the robot based on the command velocity. The diffdrive plugin multiwheel plugin has the option of publishing the odometry TF, which is disabled when the lego loam mapping option is enabled.
- /models/robots/top frame.urdf.xacro, which creates the mounting frame on the GVR-Bot
- /models/robots/cameras.urdf.xacro, which loads a depth and color camera
- /models/robots/lidar hokuyo ust20lx.urdf.xacro if the 2D LiDAR option is selected, which simulated the Hokuyo 2-D LiDAR.
- /models/robots/lidar ouster os 1 16.urdf.xacro if the 3-D LiDAR option is selected, which simulated the Ouster OS1-16 3-D LiDAR.

Sometimes it is desirable to create a new Gazebo world to put the robot. To do that, the Gazebo world needs to be created and saved, then link it to the launch file to give the option of loading this new world with the robot:

1. Launch Gazebo (not necessarily gazebo ros)
2. Import the Gazebo model that is wanted in this world. If a brand new model (e.g., a tunnel from a 3-D file) is being created, you will need to follow steps similar to the Gazebo tutorial on creating a new model, then import this model to your Gazebo world.
3. Save the Gazebo world as a file, and copy it to the /worlds folder in the GVR-Bot simulation package. For redistribution purposes, it is a good idea

- to copy the custom made Gazebo model (/.gazebo/models/) to the /worlds/gazebo models folder in the GVR-Bot simulation package.
4. In the launch/gvrbot gazebo.launch file, add the segment for loading this new world, which should be similar to those for the cafe and office options.

6 Wireless Ranging

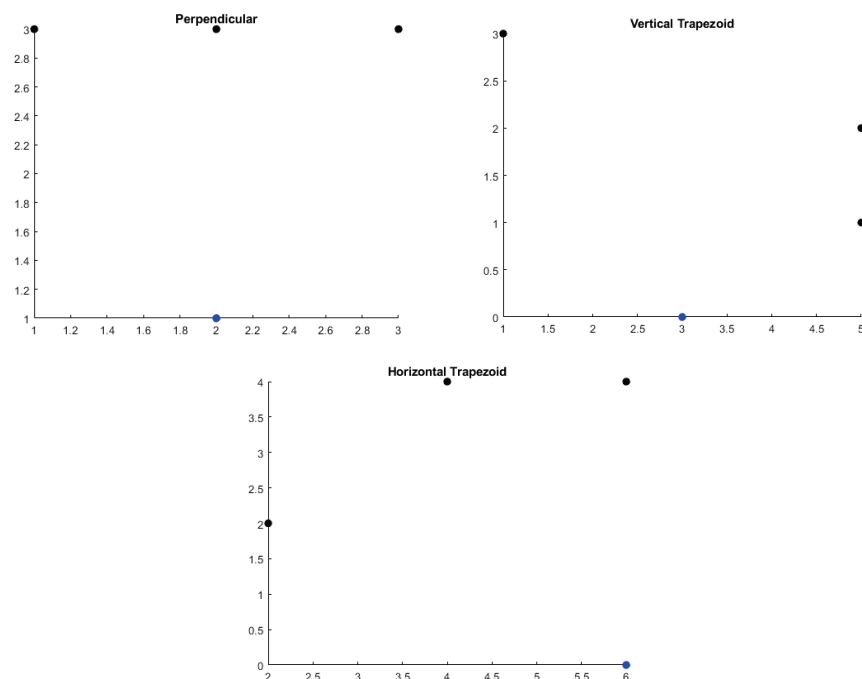
A constraint of this project was the assumption that it would be operated in a GPS denied environment. Typically, GPS is the popular choice for providing the map coordinate frame for the robot—in other words, it gives the robot reference points in space, which allows the robot's localization algorithm to correct for error that occurred in its odometry. Without the map frame, the robot always has some amount of drift, which accrues over time, causing straight hallways and tunnels to become curved or skewed. In short, two plans of attack for improving localization on the robot were made. One was to provide the map coordinate frame for the robot, without the use of GPS, and the other was to minimize drift from the odometry. This section will discuss the method for providing the map frame. The COTS Pozyx kit was chosen as a potential option for providing the map coordinate frame for the robot, via its ultra-wideband (UWB) capabilities. Additionally, it provides an IMU, which could provide odometry to help minimize drift. Pozyx Labs advertises 10 cm precision of localization, an average of 30 m range, and a great feature called automatic calibration.

The Pozyx kit has two types of boards: the tag, which is the Arduino shield, and the anchors, which are placed in the surrounding area and provide the coordinate frame. There are four anchors included in the localization kit, which is required for 3-D positioning; however, more could be used. Setting up requires placing each anchor in the desired location, then performing either manual or automatic calibration. The automatic calibration feature allows the Pozyx tag to find the anchors and automatically determine the position of each anchor. This is possible since each tag is able to act as an anchor. Automatic calibration would be a great benefit to this project, as it would speed up initialization time, as well as better integrate with a fully autonomous system. However, upon testing, several issues were identified. In testing, automatic calibration failed a majority of the time. Also, the precision was significantly larger than 10 cm when using automatic calibration. While it may be possible to take multiple automatic calibrations and filter out bad results, Pozyx's automatic calibration algorithm already averages ten measurements by default. In addition to the inaccuracy, another problem with automatic calibration was the orientation of the coordinate frame was not predetermined. When fusing the odometry from the UWB with other sources of odometry, it was required that the coordinate frame match the other sensors' frames (forward being +x at initialization, left being +y, up

being +z). As the automatically calibrated frame is indeterminate, it was not viable for this project. As a result, to test the effectiveness of integrating UWB for localization, it was decided to use manual calibration. This involves measuring the position of each anchor with millimeter precision in all three dimensions, using a tape measure or range finder. While this isn't realistic for this particular situation, it does provide 10 cm precision, as Pozyx Labs advertises.

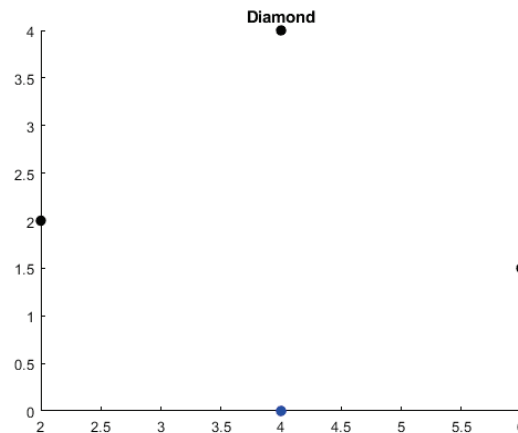
In the early stages of project design, it was envisioned that there would be three or four “anchor bots,” each with one Pozyx anchor, all following the primary mapping robot. The anchor bots would then park as the mapping bot conducted its mission. To this end, it was decided to test different configurations of anchors, in which all anchors were positioned behind the robot. This is contradictory to Pozyx Lab's guide on optimal anchor placements, which affects both the range and precision of localization. First, configurations of three anchors were tested, instead of four. Of course, this meant it would be localizing in 2-D. If 2-D localization was all that was required for this application, then fewer anchors mean more efficiency, but less precision. Figure 19 shows some failed configurations. The black dots are anchors, and the blue dot is the tag. The units are arbitrary since it's the shape that matters.

Figure 19. Various anchor and tag configurations.



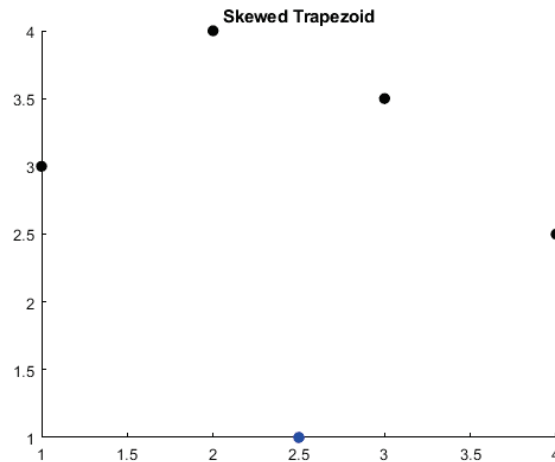
All three of the configurations in Figure 19 were less than 10% reliable, meaning that the Pozyx tag could detect its position less than 10% of the time. Figure 20 shows a diamond shape configuration, which was the only one found to be reliable with three anchors.

Figure 20. Preferred configuration using three anchors and a tag.



This diamond configuration was found to be 80% reliable, which still isn't as acceptable as one would like. It may be that having three anchors all in the +y direction of the tag does not provide enough precision to triangulate the position of the tag along the y-axis. Next, localization was tested with four anchors for both 2-D and 3-D positioning. Setting up the anchors in a planar shape will break the Pozyx's positioning algorithm since it cannot tell if the tag is in one of two equidistant points from the plane. Pragmatically, this just means that a) the anchors must be set at different heights for 3-D localization, and b) do not set up anchors such that one set of opposite corners are high and low, while the other set of opposite corners is in the middle. This is too close to being planar and will cause problems with positioning. Figure 21 shows a configuration that was found to be near 100% reliable. Note that the anchors second and fourth from the left were placed at a higher height in order to test 3-D localization.

Figure 21. Preferred configuration using four anchors and a tag.



Not only was this configuration acceptably reliable for both 2-D and 3-D localization, but it also had a range up to 7 m outside of the configuration (and in another room); meaning this range was after many reflections of the UWB signal. From these tests, it was decided to stick with all four anchors, rather than three.

Another area of concern is range. When it was tested in the NVESD firing tunnel, the configuration of the anchors were able to be expanded to about 25 m in length. Using a configuration similar to the one above, except turned sideways to account for the long shape of the tunnel, the range of the Pozyx was tested. It was found that we could not map much outside of the configuration; once the Pozyx lost signal from the furthest tag at about 30 m, it could no longer localize in 3-D. Furthermore, it was found that at one reflection of UWB signal, the range was decreased to about 20 m. Consequentially, the localization was reliable while mapping the interior of the 30 m long area, but going much outside of the specific area was difficult.

It is worth noting that the Pozyx library's positioning function allows for use of two algorithms, designated as "UWB_ONLY" and "TRACKING." According to the Pozyx Lab's performance tests, the tracking algorithm "uses previous position estimates in its estimation which smooths the result." This implies a) that it is not using the IMU to help track and b) that the measurements are being fused in a simple low-pass filter internally. This second point is especially helpful for a moving robot. Both algorithms were tested and the tracking algorithm was found to have less noise. Therefore, it was decided to stick with the tracking algorithm.

To use the data from the Pozyx for robotic localization, a way to export its data from the Arduino to the CPU was needed. Fortunately, there's a ROS package called `rosserial` that has a node specifically developed for the Arduino. `Rosserial` imports the data through the USB A-to-B cable that connects the Arduino to the CPU and publishes the data to a user-specified topic. It can also publish to multiple topics at the same time, which, in this case, helps to fuse the UWB data and IMU data separately.

Any message type can be imported and used; furthermore, custom message types can be specified in the `ros_lib` Arduino library and be used as well. However, since `robot_localization` was used for sensor fusion, standard message types will do. `Robot_localization` supports the following messages

- `nav_msgs/Odometry`
- `sensor_msgs/Imu`
- `geometry_msgs/PoseWithCovarianceStamped`
- `geometry_msgs/TwistWithCovarianceStamped`

For the UWB data, a position (x, y, z) and orientation (roll, pitch, yaw) can be obtained, so the message type `PoseWithCovarianceStamped` was imported. For the IMU data, orientation, angular velocity (vroll, vpitch, vyaw), and linear acceleration (ax, ay, az) can be obtained, so the message type `sensor_msgs/Imu` was imported.

The units used by the Pozyx are generally different from the standard ones used in `robot_localization` and that are specified by REP-10313, so some unit conversions are required. Since it is not always clear where to find the units or what the units are (Yang et al. 2019), the ones used are discussed here.

- Position. Pozyx gives in millimeters (mm). It needs to be cast to float, then divided by 1000 to get meters (m).
- Orientation. The orientation is fine; both Pozyx and REP-103 use quaternions.
- Pose covariance. Mostly just affects the convergence rate of the `robot_localization` EKF, to be discussed later. Unit conversions were not used.
- Angular velocity. Pozyx gives degrees per second. It needs to be divided by 360 and multiplied by 2π to get radians per second.

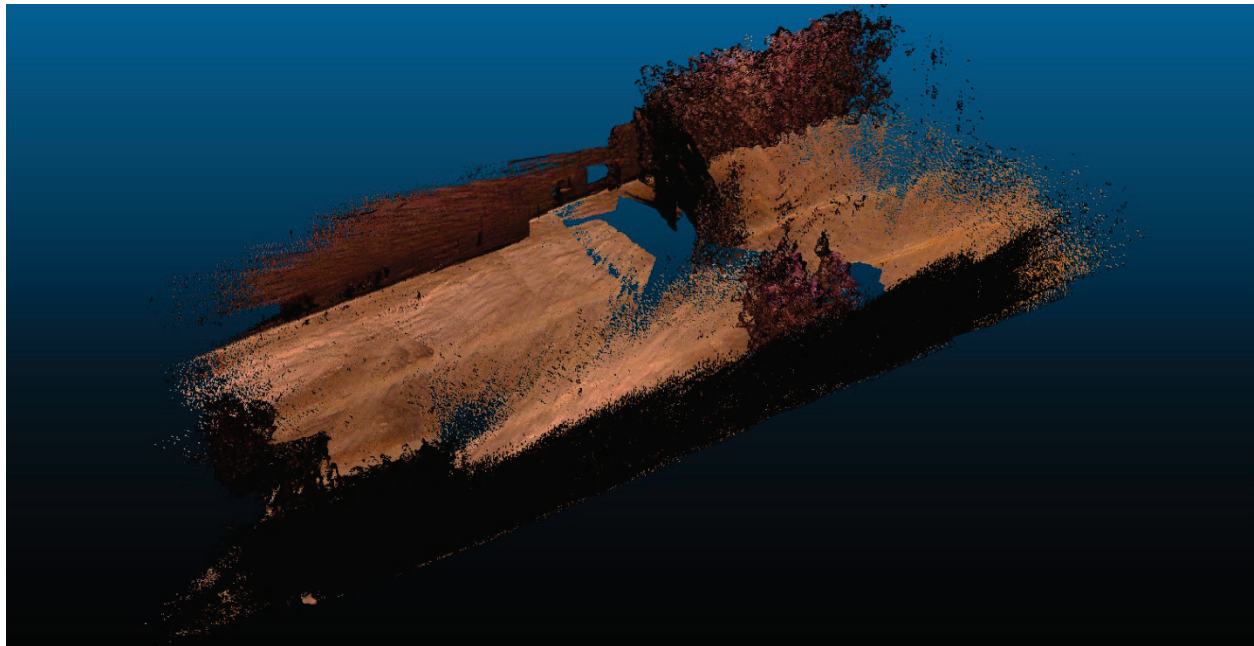
- Linear acceleration. Pozyx gives milli-G's (mg), meaning in multiples of a thousandth of Earth's gravity. It needs to be cast to float, then divided by 1000, and multiplied by 9.8 to get meters per second per second (m/s/s).
- Orientation, angular velocity, and linear acceleration covariances. The Pozyx provides no covariance values for the IMU. The diagonals of each covariance matrix were set to 1.0, for reasons to be discussed in the Covariance Matrices sub-section.

The update rate of the UWB can play a large role in the positioning precision of the robot. Thus, a couple of important code optimizations were made. The first is using the `memset` function to initialize all of the structs for storing Pozyx register data to 0. This allows for a) only modifying the diagonal entries of the covariance matrices and b) removing extra error checking from the code. If all 0's publish to the ROS topic, then there is something wrong (i.e. out of range of the anchors). The second optimization was to reduce register reads. Register reads are very expensive computationally. However, they are mandatory. By having one Arduino program that publishes both UWB pose data and IMU data, and another that publishes just IMU data, the update rate of the latter was greatly increased. This helped minimize drift without the UWB, since the UWB is still range-limited. Not only are there far fewer register reads, but also less memory being used, and the `rosserial` node does not need to flip back and forth between the two output topics (which halves the update rate). Setup files can be found in Appendix E.

Unfortunately, the Pozyx does not provide covariance values for its IMU measurements. Having covariance measurements may dramatically improve the accuracy and convergence rate of `robot_localization`'s EKF. There is no clear guidance on what to use as the input to `robot_localization` when the covariance matrix is unknown, but many say just to use a covariance matrix full of 0's. This was tested using the empty covariance matrix and found the robot was struggling to keep track of its rotation. However, when the diagonals were replaced with 1.0, the localization performed much better, and drift error was completely negated in the firing tunnel tests. Since having these covariance values improved the results so dramatically, it's undoubtedly the better choice for this project. Future work with the localization could include adjusting (hard-coding) these covariance values to see how the localization is affected. Although, this sounds like a bad practice, the covariance matrix of the sensors

directly affects the Kalman gain of the EKF. Since robot_localization currently has no option to input sensor bias, hard-coding the covariance could be a method to bias the sensor instead. An example of a point cloud collected leveraging UWB is shown in Figure 22. What is not evident is the extremely complex path the robot took. The robot performed a number of turns and revisited similar locations multiple times. In short, UWB helps mitigate the drift incurred by the IMU and wheel encoders over time. While the concept of integrating UWB can improve performance has been shown, the limitations above are prohibitive for using this technology at this time.

Figure 22. Point cloud collected leveraging ultra wide band.



7 Automation

This chapter seeks to explain the functionality and configuration of various ROS packages for autonomous exploration in bounded unknown environments. This explanation details configurations, issues, advancements, and applications for autonomous exploration with respect to three-dimensional real-time appearance-based mapping using RTAB-Map. Two-dimensional occupancy grids are used for costmap calculation and configuration, while `move_base` is used for path-planning and Twist message publication. Two exploration packages were compared and contrasted for their precision, speed, efficiency, bounds, ease-of-use, and consistency. Lastly, a configuration is described using RTAB-Map for 3-D point clouds and 2-D occupancy grids, `explore_lite` for exploration and frontier-publication, and `move_base` for costmaps, path-planning, and path-execution.

Issues abound in the field of unknown environment exploration, including the problems of SLAM, environment boundaries, known space, object detection, collision prevention, and optimal path-planning. Weighting these issues with respect to importance, and optimizing resource consumption for each issue is a necessity, specifically when computer resources are limited and exploration time is to be maximized. Therefore, finding efficient yet broad packages that have potential usage in future adaptations to this current design is very important. Adjustments to configurations within those packages are highly likely to be necessary, so some potential changes that may improve navigation and exploration performance will be listed.

In order to make a robot properly autonomous, it is necessary to understand how the packages work and interact. Essentially, `move_base` (a package included in the navigation stack) receives a goal in a map and will attempt to reach the provided goal by publishing to the common topic `/cmd_vel`. `Move_base` uses the Dynamic Window Approach (DWA) algorithm to calculate an optimal velocity for a safe trajectory within a given search space. The math and code are extensive and complicated; the pseudo-code is more manageable; for example, below.

```
BEGIN DWA(robotPose,robotGoal,robotModel)

    desiredV = calculateV(robotPose,robotGoal)

    laserscan = readScanner()

    allowable_v = generateWindow(robotV,
robotModel)

    allowable_w = generateWindow(robotW,
robotModel)

    for each v in allowable_v

        for each w in allowable_w

            dist =
find_dist(v,w,laserscan,robotModel)

            breakDist =
calculateBreakingDistance(v)

            if (dist > breakDist) //can stop in
time

                heading =
hDiff(robotPose,goalPose, v,w)

                clearance = (dist-breakDist)/(dmax
- breakDist)

                cost =
costFunction(heading,clearance,
abs(desired_v - v))

                if (cost > optimal)

                    best_v = v

                    best_w = w
```

```

        optimal = cost

        set robot trajectory to best_v, best_w

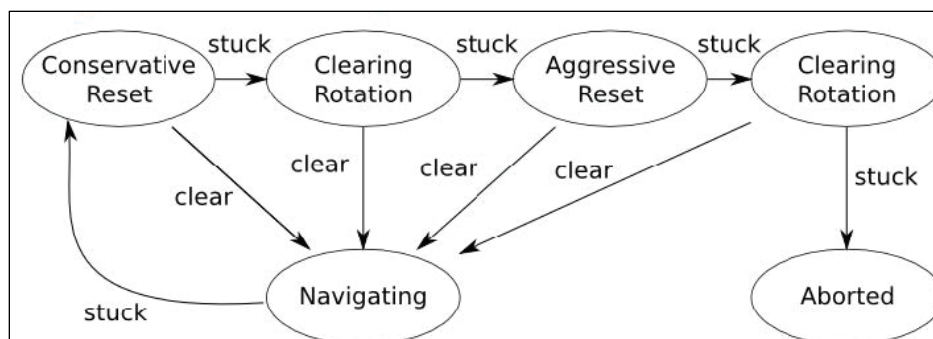
    END

```

Essentially, the robot's pose and the navigation goal are used to calculate the desired velocity (fast if the goal is far away, slow if the goal is close). Then, for each allowable angular and linear velocity, the closest obstacle is determined. If the robot cannot stop in time, then the velocity is discarded. Otherwise, the velocity is 'kept,' and among the 'kept' velocities, the velocity with the best cost is labeled as the best option, and all others are discarded. This process is computed frequently for a global and local costmap, enabling short term and long term decision making.

In the event that no velocities are 'kept,' move_base will initiate a series of recovery behaviors to attempt to unstuck itself (Figure 23). These scenarios are rare in a sufficiently static, open environment where move_base performs as expected and reaches its goal within the user-specified tolerance. However, in a more dynamic and cramped environment, move_base detects the robot as stuck more often. This is likely due to global costmaps not being updated at a high enough rate, but move_base ordinarily recovers and resumes navigation.

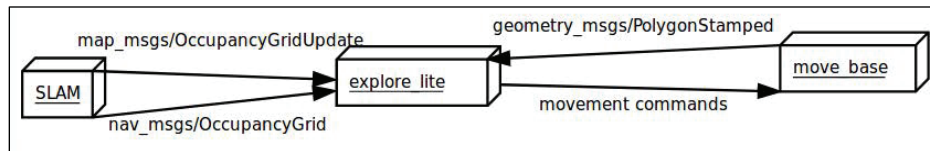
Figure 23. Flow diagram of recovery behaviors.



Helpfully, move_base reports frequent status and goal updates with real-time data through the 'move_base/status,' 'move_base/feedback,' and 'move_base/result' topics. Since these topics publish updates frequently, they are very useful in debugging navigation failures and recovery option executions.

Deciding the goal for `move_base` is ordinarily an operator's responsibility, but this process can be automated via several different exploration packages. The exploration package merely decides the goals on the map and publishes these goals to `move_base`. `Explore_lite`, for instance, subscribes to a 'nav_msgs/OccupancyGrid' message published by RTAB-Map to locate frontiers (the boundary edge between known space and unknown space). `Explore_lite` can also be configured to subscribe to a costmap constructed from the same message, in this case, published by `move_base`. After analyzing the occupancy grid, `explore_lite` sends navigation goals to `move_base` relating to frontiers that `explore_lite` has located (Figure 24). `Explore_lite` does not require client-side visualization to operate as it is entirely autonomous. Frontiers can be visualized in Rviz, but no boundaries can be made. This means that exploration is unbounded and will stop when no more frontiers can be identified. Because of all these factors, `explore_lite` is rather quick compared to other exploration packages. In fact, `explore_lite` located and eliminated a dozen frontiers in a small lab in four minutes consistently, only failing when gmapping failed.

Figure 24. Integrating the costmap into automation.



Another exploration package that may be used is `frontier_exploration`. `Frontier_exploration` and `explore_lite` are very similar in the respect that they both send navigation goals to `move_base` regarding frontiers. In fact, `explore_lite`'s algorithm is based on `frontier_exploration`'s. However, `frontier_exploration` enables more features, including boundary visualization, multi-nodal publications, and subscriptions via `explore_client` and `explore_server`, `BoundedExploreLayer` utilization, and autonomous exploration with or without a global '/map' information source. Overall, the `frontier_exploration` package seems to have immense potential for autonomous exploration. However, even while running solely `move_base`, gmapping (a 2-D mapping package), and `frontier_exploration`, exploration was slow. Exploring a mere dozen original frontiers in a relatively small lab took approximately 15 min, and the robot was often stuck, or `frontier_exploration` failed entirely. These issues were worsened when gmapping was substituted for RTAB-Map, and additional packages were run, including `robot_localization` and

`find_object_3D`. On the whole, `frontier_exploration` seems like a very useful package in the long-term, but not a package that can be used given the limited hardware configuration. In addition, `frontier_exploration` requires visualization to be initiated. Specifically, the boundaries for exploration must be specified in Rviz after the package is launched. This can be a hindrance or a benefit, depending on the situation. If the robot is mapping in an open area, then selecting boundaries is desirable. However, if the region is completely unknown, then it is advantageous to enable unbounded exploration.

Although the differences between the two exploration packages are minimal, the performance differences are measurable. The hardware configuration for autonomous exploration is very flexible since `move_base` and the exploration packages rely on the output of RTAB-Map. If RTAB-Map's performance is improved, then so is the performance of the robot's autonomous features. Configuring the hardware such that noise is minimized and known area around the robot is maximized provides the best results.

`Frontier_exploration` is customizable and can be used with or without a global map topic. The package creates its costmaps, which can be useful. However, running `move_base` is necessary and provides costmaps as well, so this feature is unnecessary. The parameters that are used in `frontier_exploration` are nearly identical to the parameters used in `move_base`, so setup is rather simple if `move_base` is working properly. There is a multitude of parameters, but it was found that the most impactful ones were `inflation_radius`, `footprint`, `publish` and `update` rates, and `exploring_clear_space`. Other parameters will be discussed, but these were the most experimented and demanded the most precision.

The `footprint` is the size and shape of the robot and is used to path-plan and avoid collisions. If the footprint is too big, then the robot will not navigate through corridors that it can fit through. If the footprint is too small, then the robot collides with many objects and gets stuck more often. It was found that having the footprint as close to the actual size of the robot as possible was the most effective value and produced the best results.

The `inflation_radius` is essentially the buffer zone between object and `clear_space` and is the region where cost scaling functions are applied.

Having the inflation radius as slightly more than half of the longest side length of the robot was the best value. Essentially, this prevented the robot's center from passing over the inflation_radius 'line,' which prevented collisions and aided in corner turning.

The publish and update rates are very important. If the rates are too high (10 – 20 Hz), then the robot would slow down, and map quality would be poor. If the rates were too low (0 – 2 Hz), then the robot would collide more often. The correct value was found to be in the range of 4 to 7 Hz. This rate could be increased with more powerful hardware, but a rate of 5 Hz seemed sufficient with the current configuration.

Another parameter that was updated was `frontier_travel_point`. This parameter decides what the pose of a new frontier should be with respect to the robot's pose. Essentially, setting this parameter to "closest" tells the robot to go to the closest point on the frontier to itself. Although shorter distance-wise, setting this parameter to "closest" often resulted in collisions and corner cuts. Instead, setting this parameter to "middle" reduced collisions significantly.

`Frontier_exploration` also supplies a variety of `explore_costmap` plugins including `TrajectoryPlannerROS` and `NavFnROS`. These plugins all have parameters of their own that need to be set, including footprints, velocities, accelerations, origin points, map sizes, transform tolerances, frontier tolerance, etc. These parameters were experimented with and changed often, but `TrajectoryPlannerROS`'s parameters are the most important. The parameters in this plugin impact the jerkiness of the robot's movement, which in turn impacts the quality of the map. All of these parameters are loaded in the `frontier_exploration` launch file (Appendix F).

`Explore_lite` also contains several parameters that need to be updated, but the package is much more streamlined than `frontier_exploration`. Of these parameters, the necessary updates were with "`costmap_topic`," "`visualize`," and "`min_frontier_size`." `Costmap_topic` determines where `explore_lite` should receive the costmap. This topic could be the `/map` topic, but the global costmap that `move_base` produces was used. "`Visualize`" enables frontier visualization in Rviz, which is unnecessary but assists with testing. Setting "`visualize`" to false would make the `explore_lite` package faster and is unnecessary in a truly autonomous robot. Finally, "`min_frontier_size`"

was increased to prevent more needless frontier exploration, which in turn mitigated collisions. At the default value, `explore_lite` was setting frontiers to be in corridors that the robot couldn't navigate through; increasing this value removed that issue. In summary, `explore_lite` is far faster and more lightweight than `frontier_exploration`, but it is less customizable.

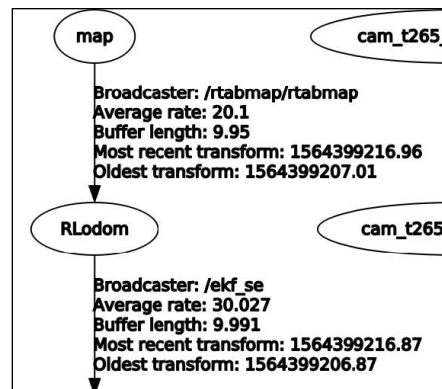
With autonomous mapping, the exploration process should be used alongside a mapping package such as RTAB-Map. Since RTAB-Map provides occupancy grid topics, it can be used for autonomous exploration. Originally, exploration packages were tested using gmapping as a benchmark, since gmapping is consistent, fast, and simple. However, gmapping produces a 2-D occupancy grid purely for navigation, not a 3-D map that can be saved for later analysis, such as RTAB-Map. RTAB-Map produces three different occupancy grid topics that can be used for `move_base` and `explore_lite`, namely `/proj_map`, `/grid_map`, and `/grid_prob_map`. Both `/proj_map` and `/grid_prob_map` were too messy and noisy for accurate path-planning, so `/grid_map` was used instead. These maps are constructed by taking a horizontal region of the robot's camera's field of view and collapsing the region into a 2-D plane that is then constructed into a simple `grid_map`. This construction is done by assigning a value of -1 to areas that are occupied, and nonnegative constants elsewhere. The projection map topic does this as well, except it constructs this region over the entire 3-D point cloud. Theoretically, this is the most practical approach since it allows for more 'out of range objects' to be detected, and therefore not be run into. In practice, however, the projection map appears noisy upon startup and does not continue to get better until loop closure is detected. Therefore, the grid map topic is the only feasible option.

If RTAB-Map did not produce an occupancy grid message type, then it could not be used. This notice is necessary since if there is a better package that is to be used in the future, the message types that it outputs must be analyzed. If there are no occupancy grid message types, then the package cannot be used without an additional conversion package. The package `"grid_map"` might be useful for this, but it would require a transformation from point cloud to octomap, then from octomap to an occupancy grid. This would be difficult and inaccurate, so a package that produces an occupancy grid such as RTAB-Map is objectively favorable.

Even though the grid map topic is functional, there are a few strange configurations that make the topic difficult to incorporate into `explore_lite` and `move_base`. It appears that unknown space in RTAB-Map is regarded as a clear space, which is not always true. This leads to dangerous decision making on the part of `explore_lite`. Often-times, `explore_lite` assigns frontiers directly behind the robot, outside of the 2-D LiDAR's field of view. `Move_base` does not turn around and scan the area, instead telling the robot to go directly backward, since it assumes the area is clear. This can be corrected by rotating the robot 360 degrees after RTAB-Map is running, but before `explore_lite` is launched, or it may be fixed by implementing a 3-D LiDAR that can scan 360 degrees around the robot. That way, all clear space is properly marked and `explore_lite` can begin assigning frontiers.

Gmapping does not have this unknown space marking issue. Unfortunately, it is both non-trivial and heavy on resources to run two SLAM packages simultaneously on one machine. In addition, both RTAB-Map and gMapping publish to the `map→odom` frame, in this case, the `RLodom` frame (Figure 25). It is, therefore, necessary that solely RTAB-Map should be used, both for 3-D mapping and 2-D occupancy grid publishing, unless a more effective alternative to both processes has been discovered and configured.

Figure 25. Publishing the odom from RTAB-Map.



Although highly effective, the autonomous exploration configuration described within this document does occasionally fail. These issues have been documented and analyzed and are listed below. Most of these issues are due to sudden changes within a dynamic map, which can have impacts on all three of the packages previously detailed. Most, if not all, of these issues, can be remedied with software and hardware updates, but some of

these issues may be exclusive to the exploration and path of the environment. Many issues that have been resolved may reappear, so those are listed as well to ease diagnosis and solution. The issue, package at fault, and solution are listed for each issue for clarity.

Issues that have been resolved:

- Issue: The robot is cutting corners too close, resulting in a collision.
- Caused by: `move_base`
- Solution: Verify that local costmap resolution is similar to the lowest sensor resolution that is being imported to `move_base`, in terms of meters per cell. Also, adjust the robot footprint in both the local and global costmap configuration to be slightly larger than the actual robot size and adjust the inflation radii to be slightly larger than half of the longest side of the robot (or slightly larger than the radius of the robot, if the robot is circular).
- Issue: `Move_base` is too slow, resulting in goals being overshoot or not attempted.
- Caused by: `move_base` or `frontier_exploration`
- Solution: Lower the update and publish frequency for the first global costmap within `move_base`, then the local costmap. If the issue still persists, adjust the update frequency within `frontier_exploration` (or `explore_lite`). If the issue still persists, use a faster and lighter exploration package, such as `explore_lite`.
- Issue: The robot is not moving, even though `/move_base/status` is publishing goal data.
- Caused by: Twist messages not being properly published to `/cmd_vel`.
- Solution: Verify that `/cmd_vel` is being properly published by the robot; this node is usually launched upon startup. If it is, then with `move_base` running, publish a 2-D Nav Goal in Rviz. If the robot does not move after the goal is published, verify that `/cmd_vel` is not being remapped to another topic, usually `'/mobile_base/commands/velocity'`. If the robot does move, verify that your exploration package is properly publishing to `move_base` and is subscribed to `/move_base/global_costmap/costmap` (unconnected nodes can be analyzed with `$ roswtf`).
- Issue: Upon the launch of the exploration package, the robot immediately drives backwards at high speed.

- Caused by: RTAB-Map occupancy grid format.
- Solution: RTAB-Map's occupancy grid appears to mark unknown space as clear space. This results in move_base believing that the area not seen by the robot is clear, which it then utilizes for navigation. This can be prevented by, upon the startup of move_base, but before the startup of an exploration package, rotating the robot 360 degrees using a manual controller (or by publishing the proper twist message to /cmd_vel for the correct duration). This can also likely be prevented by installing hardware that has 360 degrees of view around the z axis, which prevents unknown space from being directly behind the robot. It should be noted that this issue is non-existent in gmapping, which Google-Cartographer utilizes in its SLAM capabilities. If there are future fatal issues with RTAB-Map, then Cartographer may be an efficient and effective option.
- Issue: When a Navigation Goal is published relatively close to the robot (as in NW or NE 5-10 feet), the robot drives the incorrect direction, potentially resulting in a collision.
- Caused by: RTAB-Map occupancy grid format and/or odometry issues specific to the robot.
- Solution: If after resolving issue 4, verify that the robot's odometry is correct relative to changes in robot pose. Also, verify that there is no drifting when the robot is still. This issue was resolved by editing the odometry node that launches upon robot startup; left and right rotations were swapped.
- Issue: Robot's movement is very jerky, but does reach its goal.
- Caused by: Move_base trajectory or local_costmap configurations.
- Solution: Adjust the TrajectoryPlannerROS acceleration and velocity parameters for both linear and angular velocities. Specifically, decrease the maximum acceleration and increase the maximum velocities slightly. If the robot's movement is still jerky, adjust the local_costmap to have a higher publish and update frequency.
- Issue: Robot's movement is very jerky, and does not reach its goal, often resulting in collisions.
- Caused by: A variety of potential issues, including TF overwrites, hardware malfunctions, incorrect IMU and odometry data, and severely incorrect parameters.
- Solution: First, check \$ roswtf to verify that all nodes are connected and that there are no Transforms (TF) overwrites occurring. If there

are TF errors, they could be caused by gmapping and RTAB-Map being run simultaneously. Both of these packages publish to the map→odom transform, so there are obvious conflicts that can result in jerky movement. Choose one package or the other, not both. This was the solution in this case. However, this issue could also be similar to issue six, with more incorrect parameters. Make sure that these parameters are precise. Lastly, check that there is no odometry drift by analyzing the odometry topics. This analysis should first be done sensor-by-sensor and then from robot_localization.

- Issue: After launching the exploration package, all navigation goals are tested and fail.
- Caused by: Move_base costmap parameters.
- Solution: Adjust the resolution, inflation radii, and footprint. See issues one and four.

Issues that have not been resolved, but can likely be fixed:

- Issue: After exploring some, the robot reaches a point where it faces an obstacle and pauses indefinitely.
- Potential Cause: Dynamic environment changes while the robot is path-planning, resulting in the blockage of calculated path.
- Potential Solution: Move_base may be able to reset costmaps and paths if an obstacle suddenly intersects the planned path. This issue may be resolved, and does not occur often, around one in every eight scans. This issue can be circumvented by publishing a navigation goal directly behind the robot and facing away while this issue is occurring. The robot should then resume exploring normally. If not, relaunch explore_lite (or frontier_exploration).
- Issue: Frontier_exploration behaving extremely slowly, even with RTAB-Map not running.
- Potential Cause: Unknown, potentially hardware not being powerful enough.
- Potential Solution: This issue was resolved by switching to a lighter and simpler exploration package.

- Issue: Move_base not navigating to the far corners of the map.
- Potential Cause: Messy RTAB-Map occupancy grid scans, specifically projection map, particularly prior to loop closure.
- Potential Solution: This issue could potentially be fixed by using a different mapping software. This issue does not occur often, only approximately one in every fifteen scans, or in areas with very low ceilings.

8 Future Advancements

In the future, many aspects of this current configuration may be changed. Faster and more effective mapping software may be developed, path-planning packages with higher efficiency will be released, and better hardware will become cheaper. Some advancements are virtually guaranteed. But, there are also some that are being proposed that will improve performance in future tests, although they will be extensive and difficult to configure in the short-term. Currently, only one robot is operating during scans. However, it appears possible to have multiple robots operate in a swarm if they communicate with one another. In fact, a package called “multirobot-map-merge” has potential to do just that. This package is included in the `explore_lite` installation and may be utilized in future advancements to this current configuration. In addition to `multirobot-map-merge`, `map-merge-3d`, a package authored by the same person whom wrote `multirobot-map-merge` and `explore_lite`, may prove useful. `Multirobot-map-merge` merges global occupancy grid topics, and `map-merge-3d` merges 3-D point clouds produced by a SLAM software such as RTAB-Map. Together, these two packages could be used to make autonomous swarm a possibility.

In addition, it may be possible to incorporate more advanced SLAM software, such as Google Cartographer in future tests. Cartographer does not yet support occupancy grid publication, which means that it cannot be used together with autonomy. However, Cartographer does utilize gmapping, so translating cartographer data to an occupancy grid format is likely possible. Sometimes, objects are slightly out of range of RTAB-Map and are struck by the robot. This might be done in several ways, but one way that has been experimented with and documented is by using the ROS package ‘`depthimage_to_laserscan`.’ Any RGB-D camera that produces a depth-image can also be approximated to be a laser. This can be used to calculate the distance between the robot and objects, potentially hazardous ones.

In addition to this adaptation to RTAB-Map, some parameters may prove useful in the future. The parameter “Grid/MaxObstacleHeight” can be set to exclude data over a specified height. This is extremely useful in subterranean environments and may reduce the messiness of projection maps and grid maps produced by RTAB-Map. Ideally, the value of the parameter would be the maximum height of the robot plus approximately

ten centimeters to give the robot a buffer when exploring autonomously. Hardware changes may impact performance, as well. Incorporating 3-D LiDAR would aid in mapping precision and path-planning accuracy. This incorporation would eliminate “clear space” problems detailed in issue four.

When performing in a flat environment, `move_base` and two-dimensional exploration packages, such as `explore_lite` and `frontier_exploration`, perform normally. However, in a subterranean environment where elevation changes, `move_base` does not work as well, since it relies on a 2-D occupancy grid map that RTAB-Map produces. However, `move_base_flex` (MBF), a package that is meant to be a reverse compatible replacement for the `move_base` package, may prove very useful. Utilizing 3-D occupancy grids instead of 2-D grids will be useful for traversing three-dimensional environments, which MBF claims to provide. It is very new and is constantly being updated, so issues are common. MBF does have several advantages including a thorough recovery procedure involving about thirty ‘nodes’ and nearly as many branches. `Move_base`’s recovery procedure has a comparably lousy six nodes. If applicable, MBF will be a complete replacement for `move_base`. The package is massive and very customizable, so it may be applicable to MOWLES.

In addition to MBF for 3-D exploration, the mapping metapackage `grid_map` may be useful. The metapackage is a C++ library with an ROS interface to manage two-dimensional grid maps with multiple data layers. It is designed for mobile robotic mapping to store data such as elevation, variance, color, friction coefficient, foothold quality, surface normal, traversability, etc. The library was a research project for legged UGVs in rough terrain and is very complex. However, the library may be useful for autonomous exploration in the future due to its ability to navigate in very rough terrain. This would be useful for rugged terrain that may be encountered in a subterranean environment.

9 Conclusion

The purpose of the MOWLES project is to develop, test, and employ a lower cost 3-D mapping capability, encompassing low SWaP (size, weight and power) sensors for UGV deployment in GPS denied areas, to more efficiently characterize subterranean and interior environments. In this investigation, a number of sensor types were explored. It was determined that “structured light” was the most promising candidate for the generation of near real time RGB point clouds. Specifically, the Real Sense d435 fills this niche. While most structured light sensors have a hard time in bright environments, the d435 also incorporates stereo cameras to generate depth. Also, the d435 possesses global shutter cameras which reduce image blur while moving. While the range of the d435 is quite low, ~10m, it was found that fusing a lidar, which has a significantly longer range ~60m, allows the robot to close loops effectively. The inclusion of the LIDAR greatly improves the accuracy of the 3-D generated point clouds.

A number of SLAM modules were also explored. Ultimately, it was determined that RTAB-Map had the most promise on a robotics platform. While most SLAM modules include IMU inputs for localization and positioning, RTAB-Map allows for the inclusion of an odometry input. One source of odometry input is from the robot’s wheel encoders. However, it was found to have better accuracy fusing the Real Sense t265 sensor onto the robot. The t265 uses stereo fish eye cameras and a visual SLAM approach, in conjunction with an onboard IMU, to generate an odometry topic that is subscribed in RTAB-MAP. The resulting point cloud has an accuracy of ~6m when compared to a survey grade LIDAR system. It is also significant to note that the current payload costs ~5k definitively fulfilling the low SWAP-C requirement.

In addition, this effort involved the creation of a simulation environment that can adequately represent this robot while still using ROS. This enabled virtual testing for new sensor ideas or tuned algorithms and potentially could be used to train users on the system. Using UWB for wireless ranging was also explored and it was determined that the commercial system evaluated, while promising as a proof of concept, was too premature for incorporation in the final payload design.

In regards to autonomy, Frontier_exploration and explore_lite are both promising in the subject of unknown environment exploration and may

prove crucial to the war-fighter. When combined with real-time appearance-based mapping, autonomy is a powerful tool that enables data collection from afar for a very small price, both monetarily and physically. Frontier_exploration, although slow, is very customizable and may be applicable in the future. The fact that frontier_exploration creates its own costmaps is not useful in the moment, but may become useful in massive scans where an occupancy grid message is unreliable or too large to analyze effectively. Explore_lite, although fast, efficient, and effective, is not very customizable and lacks costmap creation. This streamlines the exploration process, but may be too simple for more complex and large regions in the future. These advancements are the next step for autonomous exploration and will improve exploration capabilities, leading to faster scans with more precision.

References

- Börcs, A., B. Nagy, and C. Benedek. 2017. "Instant object detection in LiDAR point clouds." *IEEE Geoscience and Remote Sensing Letters* 14.7: 992-996.
- Brenner, C., C. Dold, and N. Ripperda. 2008. "Coarse orientation of terrestrial laser scans in urban environments." *ISPRS journal of photogrammetry and remote sensing* 63.1: 4-18.
- Castellanos, J. A., J. Neira, and J. D. Tardós. 2001. "Multisensor fusion for simultaneous localization and map building." *IEEE Transactions on Robotics and Automation* 17.6: 908-914.
- Ekvall, S., P. Jensfelt, and D. Kragic. 2006. "Integrating active mobile robot object recognition and slam in natural environments." *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE.
- Guanci, Y., Z. Chen, Y. Li, and Z. Su. 2019. "Rapid Relocation Method for Mobile Robot Based on Improved ORB-SLAM2 Algorithm." *Remote Sensing* 11.2: 149.
- Jacobson, A., F. Zeng, D. Smith, N. Boswell, T. Peynot, and M. Milford. 2018. "Semi-supervised slam: Leveraging low-cost sensors on underground autonomous vehicles for position tracking." *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE.
- Labbé, M., and F. Michaud. 2019. "RTAB-Map as an Open-Source Lidar and Visual SLAM Library for Large-Scale and Long-Term Online Operation." *Journal of Field Robotics*, 36(2):416-446.
- Lehtomäki, M., A. Jaakkola, J. Hyypä, J. Lampinen, H. Kaartinen, A. Kukko, E. Puttonen, and H. Hyypä. 2015. "Object classification and recognition from mobile laser scanning point clouds in a road environment." *IEEE Transactions on Geoscience and Remote Sensing* 54.2: 1226-1239.
- Niemeyer, J., F. Rottensteiner, and U. Soergel. 2014. "Contextual classification of lidar data and building object detection in urban areas." *ISPRS journal of photogrammetry and remote sensing* 87: 152-165.
- Pollefeys, M., R. Koch, M. Vergauwen, and L. Van Gool. 2000. "Automated reconstruction of 3D scenes from sequences of images." *ISPRS Journal of Photogrammetry and Remote Sensing* 55.4: 251-267.
- Qin, X., G. Wu, X. Ye, L. Huang, and J. Lei. 2017. "A novel method to reconstruct overhead high-voltage power lines using cable inspection robot lidar data." *Remote Sensing* 9.7: 753.
- Sequeira, V., K. Ng, E. Wolfart, J. G. M. Goncalves, and D. Hogg. 1999. "Automated reconstruction of 3D models from real environments." *ISPRS Journal of Photogrammetry and Remote Sensing* 54.1: 1-22.

- Shi, S., Z. You, K. Zhao, Z. Wang, C. Ouyang, and Y. Cao. 2017. "A 6-DOF Navigation Method based on Iterative Closest Imaging Point Algorithm." *Scientific reports* 7.1: 17414.
- Tsubouchi, T., A. Tanaka, A. Ishioka, M. Tomono, and S. Yuta. 2004. "A slam based teleoperation and interface system for indoor environment reconnaissance in rescue activities." *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)(IEEE Cat. No. 04CH37566)*. Vol. 2. IEEE.
- Wen, C., S. Pan, C. Wang, and J. Li. 2016. "An indoor backpack system for 2-D and 3-D mapping of building interiors." *IEEE Geoscience and Remote Sensing Letters* 13.7: 992-996.
- Wen, C., L. Qin, Q. Zhu, C. Wang, and J. Li. 2013. "Three-dimensional indoor mobile mapping with fusion of two-dimensional laser scanner and RGB-D camera data." *IEEE Geoscience and Remote Sensing Letters* 11.4: 843-847.
- Yang, D., B. Shusheng, W. Wang, C. Yuan, Q. Xianyu, and Y. Cai. 2019. "DRE-SLAM: Dynamic RGB-D Encoder SLAM for a Differential-Drive Robot." *Remote Sensing* 11.4: 380.
- Zhang, X., A. B. Rad, and Y-K. Wong. 2012. "Sensor fusion of monocular cameras and laser rangefinders for line-based simultaneous localization and mapping (SLAM) tasks in autonomous mobile robots." *Sensors* 12.1: 429-452.

Appendix A: Sensor List

Lidar:



SF30-C Laser Rangefinder

Range: 100m

Cost: \$434.00

Link: <https://www.robotshop.com/en/sf30-c-laser-rangefinder-100m.html>



Garmin lite v3 (formerly LiDAR lite)

Range: 40m

Cost: \$130.00

Link: <https://www.robotshop.com/en/lidar-lite-3-laser-rangefinder.html>



LEDDar SDK

Range: 34m

Cost: \$690.00

Link: <https://www.robotshop.com/en/leddartech-vu-8-channel-module-100-3-usb-can--serial-interfaces-lidar-module.html>



RPLIDAR 360° Laser Scanner

Range: 25m

Cost: \$600

Link: <https://www.robotshop.com/en/slamtec-rplidar-a3-360-laser-scanner-25-m.html>



Hokuyo UST-20LX Scanning Laser Rangefinder

Range: 60m

Cost: \$2600.00

Link: <https://www.robotshop.com/en/hokuyo-ust-20lx-scanning-laser-rangefinder.html>



Ouster OS-1 (16)

Range: 120m

Cost: \$3500.00

Link: <https://www.ouster.io/product-os1>



Structure Sensor

Range: 4m

Cost: \$399.00

Link: <https://structure.io/structure-sensor>



Orbbec Astra

Range: 8m

Cost: \$150.00

Link: <https://orbbec3d.com/product-astra-pro/>

Time of flight:



TeraRanger One ToF Rangefinder

Range:14m

Cost: \$69

Link: <https://www.robotshop.com/en/teraranger-one-tof-rangefinder-type-b-frame.html>

Photogrammetry:



DUO Developer Kit

Range:

Cost:

Link: <https://duo3d.com/product/duo-ddk-lv2>



ZED Stereo Camera

Range: 20m

Cost: \$450

Link: <https://www.stereolabs.com/zed/>



Realsense D415, D435

Range: 8m

Cost: \$180

Link: <https://store.intelrealsense.com/buy-intel-realsense-depth-camera-d435.html>

Appendix B: Setup Procedure for Astra and D435

Astra:

Astra

```
sudo apt-get install libudev
```

```
#Now that all the dependencies are built
```

```
#We can install the main items
```

```
cd ~/catkin_ws/src
```

```
git clone https://github.com/orbbec/ros_astra_launch.git
```

```
git clone https://github.com/orbbec/ros_astra_camera.git
```

```
git clone https://github.com/ros-drivers/rgbd_launch.git -b indigo-devel  
#dependency of astra launch
```

```
cd ~/Downloads
```

```
git clone https://github.com/raulmur/ORB_SLAM2.git ORB_SLAM2
```

```
#now add these to your .bashrc so that ROS can find your packages
```

```
gedit ~/.bashrc #to open the file
```

```
#Make sure the bottom of the file looks like this
```

```
#source /opt/ros/indigo/setup.bash
```

```
#source ~/catkin_ws/devel/setup.bash

#export
LD_LIBRARY_PATH=/home/*USER*/catkin_ws/devel/lib:/opt/ros/indigo/lib
:/opt/ros/kinetic/lib/x86_64

#export
ROS_PACKAGE_PATH=${ROS_PACKAGE_PATH}:~/Downloads/ORB_
SLAM2/Examples/ROS

#setup our astra

cd ~/catkin_ws

catkin_make

roscd astra_camera && ./scripts/create_udev_rules

cd ~

sudo gedit 56-orbbec.rules

#put the following into the rules file

#SUBSYSTEM=="usb", ATTR{idProduct}=="0401",
ATTR{idVendor}=="2bc5", MODE:="0666", OWNER:="root",
GROUP:="video"

#SUBSYSTEM=="usb", ATTR{idProduct}=="0402",
ATTR{idVendor}=="2bc5", MODE:="0666", OWNER:="root",
GROUP:="video"

#SUBSYSTEM=="usb", ATTR{idProduct}=="0403",
ATTR{idVendor}=="2bc5", MODE:="0666", OWNER:="root",
GROUP:="video"
```

```
#SUBSYSTEM=="usb", ATTR{idProduct}=="0404",  
ATTR{idVendor}=="2bc5", MODE:="0666", OWNER:="root",  
GROUP:="video"
```

```
#SUBSYSTEM=="usb", ATTR{idProduct}=="0405",  
ATTR{idVendor}=="2bc5", MODE:="0666", OWNER:="root",  
GROUP:="video"
```

#the (idProduct) can be extended from 401-405 to 401-410

```
sudo cp 56-orbbec.rules /etc/udev/rules.d/
```

```
sudo udevadm control --reload-rules
```

```
sudo service udev reload
```

```
sudo service udev restart
```

#We can test the camera by running

```
roscore
```

```
roslaunch astra_launch astra.launch #this launches the astra service, from  
now, the camera is on and publishing topics
```

```
roslaunch image_view image_view image:=/camera/rgb/image_raw #this will  
show us the raw image
```

D435:

First you must upgrade to a supported kernel and you must be on Ubuntu 16.04 or Ubuntu 18.04

Download the following files into a new folder

http://kernel.ubuntu.com/~kernel-ppa/mainline/v4.15/linux-headers-4.15.0-041500_4.15.0-041500.201802011154_all.deb

http://kernel.ubuntu.com/~kernel-ppa/mainline/v4.15/linux-headers-4.15.0-041500-generic_4.15.0-041500.201802011154_amd64.deb

http://kernel.ubuntu.com/~kernel-ppa/mainline/v4.15/linux-image-4.15.0-041500-generic_4.15.0-041500.201802011154_i386.deb

Now open a terminal and navigate to that folder

Now run

```
sudo dpkg -i *.deb
```

This will install the new kernel

Now do

```
sudo grub update
```

Now reboot

Installing the realsense packages

```
sudo apt-key adv --keyserver keys.gnupg.net --recv-key  
C8B3A55A6F3EFCDE || sudo apt-key adv --keyserver  
hkp://keyserver.ubuntu.com:80 --recv-key C8B3A55A6F3EFCDE
```

Ubuntu 16 LTS:

```
sudo add-apt-repository "deb http://realsense-hw-  
public.s3.amazonaws.com/Debian/apt-repo xenial main" -u
```

Ubuntu 18 LTS:

```
sudo add-apt-repository "deb http://realsense-hw-  
public.s3.amazonaws.com/Debian/apt-repo bionic main" -u
```

```
sudo apt-get install librealsense2-udev-rules
```

```
sudo apt-get install librealsense2-dkms
```

```
sudo apt-get install librealsense2
```

```
sudo apt-get install librealsense2-utils
```

```
sudo apt-get install librealsense2-dev
```

Now for ROS integration

```
mkdir -p ~/catkin_ws/src
```

```
cd ~/catkin_ws/src/
```

```
#Extract https://github.com/intel-ros/realsense/archive/2.0.2.tar.gz
```

```
#Into this directory
```

```
catkin_init_workspace
```

```
cd ..
```

```
catkin_make clean
```

```
catkin_make -DCATKIN_ENABLE_TESTING=False -  
DCMAKE_BUILD_TYPE=Release
```

```
catkin_make install
```

```
echo "source ~/catkin_ws/devel/setup.bash" >> ~/.bashrc
```

```
source ~/.bashrc
```

```
sudo apt-get install ros-kinetic-rgbd-launch
```

```
#To run
```

```
roslaunch realsense2_camera rs_rgbd.launch
```

```
#and to have both stereo and lightcoding
```

```
roslaunch realsense2_camera rs_aligned_depth.launch
```

Appendix C: Installation of SLAM modules

ORB_SLAM2

#To set up the machine from scratch, we need to configure where we can get packages from

```
sudo add-apt-repository "deb http://archive.ubuntu.com/ubuntu  
$(lsb_release -sc) main universe restricted multiverse"
```

```
sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu trusty main" >  
/etc/apt/sources.list.d/ros-latest.list'
```

```
wget https://raw.githubusercontent.com/ros/rosdistro/master/ros.key -O - |  
sudo apt-key add -
```

```
sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb_release -  
sc) main" > /etc/apt/sources.list.d/ros-latest.list'
```

```
sudo apt-key adv --keyserver hkp://ha.pool.sks-keyservers.net:80 --recv-  
key 421C365BD9FF1F717815A3895523BAEEB01FA116
```

```
sudo apt-get update
```

```
sudo apt-get install libsqlite3-dev libpcl-1.7-all libopencv-dev
```

```
sudo apt-get install ros-indigo-desktop-full
```

```
sudo rosdep init
```

```
rosdep update
```

```
echo "source /opt/ros/indigo/setup.bash" >> ~/.bashrc
```

```
source ~/.bashrc
```

```
sudo apt-get install python-rosinstall
```

```
mkdir -p ~/catkin_ws/src
```

```
cd ~/catkin_ws
```

```
catkin_make
```

```
source devel/setup.bash
```

```
echo "export  
LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/opt/ros/kinetic/lib/x86_64-  
linux-gnu" >> ~/.bashrc
```

```
sudo apt-get install ros-indigo-rtabmap ros-indigo-rtabmap-ros
```

```
sudo apt-get remove ros-indigo-rtabmap ros-indigo-rtabmap-ros
```

```
cd ~/Downloads
```

open latest 2.4.x opencv from releases tab from website

```
cd opencv-2.4.x/
```

```
mkdir build && cd build
```

```
cmake ..
```

```
make && sudo make install
```

```
sudo apt-get install libsuitesparse-dev
```

```
sudo apt-get install build-essential freeglut3 freeglut3-dev qt4-qmake  
libqglviewer2 libqglviewer-dev libeigen3-dev
```

#Since g2o has updated, it now requires an updated cmake

#To update cmake

```
sudo apt-get upgrade
```

```
sudo apt-get install software-properties-common
```

```
sudo apt-get repository ppa:george-edison55/cmake-3.x
```

```
sudo apt-get update
```

```
sudo apt-get upgrade
```

```
sudo apt-get install cmake
```

^^^

if that doesn't work, do this

```
sudo -E add-apt-repository -y ppa:george-edison55/cmake-3.x
```

```
sudo -E apt-get update
```

```
sudo apt-get install cmake
```

vvv

#you can now check your cmake version with `cmake --version`

#The following 2 installs are dependencies as math packages

```
cd ~/Downloads
```

```
git clone https://github.com/RainerKuemmerle/g2o.git
```

```
cd g2o
```

```
mkdir build && cd build
```

```
cmake ..
```

```
make && sudo make install
```

```
cd ~/Downloads
```

```
git clone https://bitbucket.org/gtborg/gtsam.git
```

```
cd gtsam
```

```
mkdir build && cd build
```

```
cmake ..
```

```
make && sudo make install
```

#this is a general camera device driver

```
cd ~/Downloads
```

```
git clone https://github.com/openkinect/libfreenect2.git
```

```
cd libfreenect2/
```

```
cd depends/
```

```
cd ..
```

```
cd depends; ./download_debs_trusty.sh

sudo apt-get install build-essential cmake pkg-config

sudo dpkg -i debs/libusb*deb

sudo apt-get install libturbojpeg libjpeg-turbo8-dev

sudo dpkg -i debs/libglfw*deb; sudo apt-get install -f; sudo apt-get install
libgl1-mesa-dri-lts-vivid

sudo apt-get install libopengl2-dev

cd ..

mkdir build

cd build

cmake .. -DCMAKE_INSTALL_PREFIX=$HOME/freenect2

make

sudo make install


#download Eigen library from online repository, this is also another math
dependency

cd ~/Downloads

#download Eigen from eigen.tuxfamily.org/

#The version used here was Eigen 3.3.4

#once downloaded, enter the folder and install

cd eigen-eigen-5a0156e40feb/
```

```
./INSTALL
```

```
cd Eigen/
```

```
mkdir build
```

```
cd build
```

```
cmake ..
```

```
make
```

```
sudo make install
```

#Pangolin is a lightweight portable rapid development library for managing OpenGL display / interaction and abstracting video input.

```
cd Downloads/
```

```
git clone https://github.com/stevenlovegrove/Pangolin.git
```

```
cd Pangolin/
```

```
mkdir build
```

```
cd build
```

```
sudo apt-get install Glew
```

```
sudo apt-get install libglew-dev
```

```
sudo apt-get install libglew2.0
```

```
cmake ..
```

```
cmake --build .
```

```
#for usb devices
```

```
sudo apt-get install libudev
```

```
#Now that all the dependencies are built
```

```
#We can install the main items
```

```
cd ~/catkin_ws/src
```

```
git clone https://github.com/orbbec/ros_astra_launch.git
```

```
git clone https://github.com/orbbec/ros_astra_camera.git
```

```
git clone https://github.com/ros-drivers/rgbd_launch.git -b indigo-devel  
#dependency of astra launch
```

```
cd ~/Downloads
```

```
git clone https://github.com/raulmur/ORB_SLAM2.git ORB_SLAM2
```

```
#now add these to your .bashrc so that ROS can find your packages
```

```
gedit ~/.bashrc #to open the file
```

```
#Make sure the bottom of the file looks like this
```

```
#source /opt/ros/indigo/setup.bash
```

```
#source ~/catkin_ws/devel/setup.bash
```

```
#source
```

```
~/Downloads/ORB_SLAM2/Examples/ROS/ORB_SLAM2/build/devel/setu  
p.bash
```

```
#export
LD_LIBRARY_PATH=/home/*USER*/catkin_ws/devel/lib:/opt/ros/indigo/lib
:/opt/ros/kinetic/lib/x86_64
```

```
#export
ROS_PACKAGE_PATH=${ROS_PACKAGE_PATH}:/Downloads/ORB_
SLAM2/Examples/ROS
```

```
#setup our astra
```

```
cd ~/catkin_ws
```

```
catkin_make
```

```
roscd astra_camera && ./scripts/create_udev_rules
```

```
cd ~
```

```
sudo gedit 56-orbbec.rules
```

```
#put the following into the rules file
```

```
#SUBSYSTEM=="usb", ATTR{idProduct}=="0401",
ATTR{idVendor}=="2bc5", MODE:="0666", OWNER:="root",
GROUP:="video"
```

```
#SUBSYSTEM=="usb", ATTR{idProduct}=="0402",
ATTR{idVendor}=="2bc5", MODE:="0666", OWNER:="root",
GROUP:="video"
```

```
#SUBSYSTEM=="usb", ATTR{idProduct}=="0403",
ATTR{idVendor}=="2bc5", MODE:="0666", OWNER:="root",
GROUP:="video"
```

```
#SUBSYSTEM=="usb", ATTR{idProduct}=="0404",
ATTR{idVendor}=="2bc5", MODE:="0666", OWNER:="root",
GROUP:="video"
```

```
#SUBSYSTEM=="usb", ATTR{idProduct}=="0405",  
ATTR{idVendor}=="2bc5", MODE:="0666", OWNER:="root",  
GROUP:="video"
```

```
#the (idProduct) can be extended from 401-405 to 401-410
```

```
sudo cp 56-orbbec.rules /etc/udev/rules.d/
```

```
sudo udevadm control --reload-rules
```

```
sudo service udev reload
```

```
sudo service udev restart
```

```
#We can test the camera by running
```

```
roscore
```

```
roslaunch astra_launch astra.launch #this launches the astra service, from  
now, the camera is on and publishing topics
```

```
roslaunch image_view image_view image:=/camera/rgb/image_raw #this will  
show us the raw image
```

```
#we can now build orbslam
```

```
cd ~/Downloads/ORB_SLAM2/
```

```
chmod +x build_ros.sh
```

```
./build_ros.sh
```

```
#to run ORBSLAM in ros mode
```

```
roslaunch ORB_SLAM2 RGBD
~/Downloads/ORB_SLAM2/Vocabulary/ORBvoc.txt
~/Downloads/ORB_SLAM2/Examples/RGB-D/ASTRA.yaml #RGBD is the
mode, the vocabulary file is provided, the ASTRA.yaml file is a provided
RGBD.yaml copied from TUM1.yaml
```

#to edit the topics that ORBSLAM listens to

```
gedit
~/Downloads/ORB_SLAM2/Examples/ROS/ORB_SLAM2/src/ros_rgbd.cc
```

#in this file, edit the following lines

```
#message_filters::Subscriber<sensor_msgs::Image> rgb_sub(nh, "TOPIC
FOR RGB", 1);
```

```
#message_filters::Subscriber<sensor_msgs::Image> depth_sub(nh,
"TOPIC FOR DEPTH", 1);
```

#The ones that were chosen were "/camera/rgb/image_rect_color" and
"/camera/depth/image_rect_raw" for our testing

#ORBSLAM provides real time fast updating sparse pointclouds which
makes it very useful for creating

#a digital environment where fast localization can be achieved as well as
fast mapping to create such environment

DSO_SLAM

#To set up the machine from scratch, we need to configure where we can
get packages from

```
sudo add-apt-repository "deb http://archive.ubuntu.com/ubuntu
$(lsb_release -sc) main universe restricted multiverse"
```

```
sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu trusty main" >
/etc/apt/sources.list.d/ros-latest.list'
```

```
wget https://raw.githubusercontent.com/ros/rosdistro/master/ros.key -O - |
sudo apt-key add -
```

```
sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb_release -
sc) main" > /etc/apt/sources.list.d/ros-latest.list'
```

```
sudo apt-key adv --keyserver hkp://ha.pool.sks-keyservers.net:80 --recv-
key 421C365BD9FF1F717815A3895523BAEEB01FA116
```

```
sudo apt-get update
```

```
sudo apt-get install libsqlite3-dev libpcl-1.7-all libopencv-dev
```

```
sudo apt-get install ros-indigo-desktop-full
```

```
sudo rosdep init
```

```
rosdep update
```

```
echo "source /opt/ros/indigo/setup.bash" >> ~/.bashrc
```

```
source ~/.bashrc
```

```
sudo apt-get install python-rosinstall
```

```
mkdir -p ~/catkin_ws/src
```

```
cd ~/catkin_ws
```

```
catkin_make
```

```
source devel/setup.bash
```

```
cd ~/Downloads
```

```
#DSO
```

```
git clone https://github.com/JakobEngel/dso.git
```

```
#suitesparse and eigen3
```

```
sudo apt-get install libsuitesparse-dev libeigen3-dev libboost-all-dev
```

```
#pangolin
```

```
git clone --depth=1 https://github.com/stevenlovegrove/Pangolin.git
```

```
cd Pangolin
```

```
git pull
```

```
mkdir -p build
```

```
cd build
```

```
cmake -DAVFORMAT_INCLUDE_DIR="" -DCPP11_NO_BOOST=ON ..
```

```
cd ..
```

```
#build dso
```

```
cd dso
```

```
mkdir build
```

```
cd build
```

```
cmake ..
```

```
make -j
```

```
#dso_ros
```

```
export DSO_PATH=~/.Downloads/ dso
```

```
rosmake
```

#To run, make sure image_raw is the correct raw image stream in ROS,
the other files must be generated by hand or a tool

```
roslaunch dso_ros dso_live image:=image_raw \
```

```
    calib=XXXXXX/camera.txt \
```

```
    gamma=XXXXXX/pcalib.txt \
```

```
    vignette=XXXXXX/vignette.png \
```

ElasticFusion

#This only works on Ubuntu 14.04, 15.04, or 16.04

#Building from scratch

```
cd ~/.Downloads
```

```
git clone https://github.com/mp3guy/ElasticFusion.git
```

```
cd ElasticFusion
```

```
#!/bin/bash
```

```
mkdir deps &> /dev/null
```

```
cd deps
```

```
#Add necessary extra repos

version=$(lsb_release -a 2>&1)

if [[ $version == *"14.04"* ]] ; then

    wget
    http://developer.download.nvidia.com/compute/cuda/repos/ubuntu1404/x8
    6_64/cuda-repo-ubuntu1404_7.5-18_amd64.deb

    sudo dpkg -i cuda-repo-ubuntu1404_7.5-18_amd64.deb

    rm cuda-repo-ubuntu1404_7.5-18_amd64.deb

    sudo apt-get update

    sudo apt-get install cuda-7-5

elif [[ $version == *"15.04"* ]] ; then

    wget
    http://developer.download.nvidia.com/compute/cuda/repos/ubuntu1504/x8
    6_64/cuda-repo-ubuntu1504_7.5-18_amd64.deb

    sudo dpkg -i cuda-repo-ubuntu1504_7.5-18_amd64.deb

    rm cuda-repo-ubuntu1504_7.5-18_amd64.deb

    sudo apt-get update

    sudo apt-get install cuda-7-5

elif [[ $version == *"16.04"* ]] ; then

    wget
    http://developer.download.nvidia.com/compute/cuda/repos/ubuntu1604/x8
    6_64/cuda-repo-ubuntu1604_8.0.44-1_amd64.deb
```

```
sudo dpkg -i cuda-repo-ubuntu1604_8.0.44-1_amd64.deb
```

```
rm cuda-repo-ubuntu1604_8.0.44-1_amd64.deb
```

```
sudo add-apt-repository ppa:openjdk-r/ppa
```

```
sudo apt-get update
```

```
sudo apt-get install cuda-8-0
```

```
else
```

```
echo "Don't use this on anything except 14.04, 15.04, or 16.04"
```

```
exit
```

```
fi
```

```
sudo apt-get install -y cmake-qt-gui git build-essential libusb-1.0-0-dev  
libudev-dev openjdk-7-jdk freeglut3-dev libglew-dev libsuitesparse-dev  
libeigen3-dev zlib1g-dev libjpeg-dev libpng-dev
```

```
#Installing Pangolin, make sure you rename glDrawFrustrum() to  
glDrawFrusrtum() in any files that have the error (it might change as the  
owner #updates)
```

```
git clone https://github.com/stevenlovegrove/Pangolin.git
```

```
cd Pangolin
```

```
mkdir build
```

```
cd build
```

```
cmake ../ -DAVFORMAT_INCLUDE_DIR="" -DCPP11_NO_BOOST=ON
```

```
make -j8
```

```
cd ../../
```

```
#Up to date OpenNI2
```

```
git clone https://github.com/occipital/OpenNI2.git
```

```
cd OpenNI2
```

```
make -j8
```

```
cd ..
```

```
#Actually build ElasticFusion
```

```
cd ../Core
```

```
mkdir build
```

```
cd build
```

```
cmake ../src
```

```
make -j8
```

```
cd ../../GPUtest
```

```
mkdir build
```

```
cd build
```

```
cmake ../src
```

```
make -j8
```

```
cd ../../GUI
```

```
mkdir build
```

```
cd build
```

```
cmake ../src
```

```
make -j8
```

```
#first calibrate
```

```
cd ~/Downloads/ElasticFusion/GPUTest/build/
```

```
#navigate into the next folder and then run
```

```
./GPUTest ..
```

```
#follow the instructions it tells you, and pastes the info into the file
```

```
~/Downloads/ElasticFusion/Core/src/Utils/GPUConfig.h
```

```
#now rebuild the program
```

```
#to run, connect a Kinect
```

```
~/Downloads/ElasticFusion/GUI/build/ElasticFusion
```

```
#if you want to run with Astra
```

```
#Put the driver files from the Astra SDK located on the website into
```

```
#~/Downloads/ElasticFusion/deps/OpenNI2/build/OpenNI2/.../drivers
```

Co-Fusion

#For co-fusion most of the process is the same

#Here is the process from scratch

#Similarly, only works on 14.04, 15.04 or 16.04

#!/bin/bash -e

#

This is a build script for Co-Fusion.

#

To install all dependencies as well, invoke:

./build.sh --fresh-install

#

which will create:

- ./deps/densecrf

- ./deps/gSLICr

- ./deps/OpenNI2

- ./deps/Pangolin

- ./deps/opencv-3.1.0

- ./deps/opencv_contrib

- ./deps/boost (unless env BOOST_ROOT is defined)

```
cd $(dirname `realpath $0`)/..

mkdir -p deps

cd deps

if [ "${1:-}" = "--fresh-install" ]; then

# add cuda and other necessary upgrades

sudo apt-get install -y wget software-properties-common

source /etc/lsb-release # fetch DISTRIB_CODENAME

if [[ $DISTRIB_CODENAME == *"trusty"* ]]; then

# CUDA

wget
http://developer.download.nvidia.com/compute/cuda/repos/ubuntu1404/x8
6_64/cuda-repo-ubuntu1404_7.5-18_amd64.deb

sudo dpkg -i cuda-repo-ubuntu1404_7.5-18_amd64.deb

rm cuda-repo-ubuntu1404_7.5-18_amd64.deb

# g++ 4.9.4

sudo add-apt-repository -y ppa:ubuntu-toolchain-r/test

# cmake 3.2.2

sudo add-apt-repository -y ppa:george-edison55/cmake-3.x

# openjdk 8
```

```
sudo add-apt-repository -y ppa:openjdk-r/ppa

sudo apt-get update > /dev/null

sudo apt-get install -y cuda-7-5

elif [[ $DISTRIB_CODENAME == *"vivid"* ]]; then

# CUDA

wget
http://developer.download.nvidia.com/compute/cuda/repos/ubuntu1504/x8
6_64/cuda-repo-ubuntu1504_7.5-18_amd64.deb

sudo dpkg -i cuda-repo-ubuntu1504_7.5-18_amd64.deb

rm cuda-repo-ubuntu1504_7.5-18_amd64.deb

sudo apt-get update > /dev/null

sudo apt-get install cuda-7-5

elif [[ $DISTRIB_CODENAME == *"xenial"* ]]; then

wget
http://developer.download.nvidia.com/compute/cuda/repos/ubuntu1604/x8
6_64/cuda-repo-ubuntu1604_8.0.44-1_amd64.deb

sudo dpkg -i cuda-repo-ubuntu1604_8.0.44-1_amd64.deb

rm cuda-repo-ubuntu1604_8.0.44-1_amd64.deb

sudo apt-get update > /dev/null

sudo apt-get install -y cuda-8-0

else

echo "$DISTRIB_CODENAME is not yet supported"
```

```
exit 1
```

```
fi
```

```
sudo apt-get install -y \
```

```
build-essential \
```

```
cmake \
```

```
cmake-qt-gui \
```

```
freeglut3-dev \
```

```
g++-4.9 \
```

```
gcc-4.9 \
```

```
git \
```

```
zeroc-ice35 \
```

```
libeigen3-dev \
```

```
libglew-dev \
```

```
libjpeg-dev \
```

```
libsuitesparse-dev \
```

```
libudev-dev \
```

```
libusb-1.0-0-dev \
```

```
openjdk-8-jdk \
```

```
unzip \
```

```
zlib1g-dev \
```

```
libpng-dev
```

```
if [[ $DISTRIB_CODENAME == *"trusty"* ]] ; then
```

```
# switch to g++-4.9
```

```
sudo update-alternatives --install /usr/bin/gcc gcc /usr/bin/gcc-4.9 60 --  
slave /usr/bin/g++ g++ /usr/bin/g++-4.9
```

```
# switch to java-1.8.0
```

```
sudo update-java-alternatives -s java-1.8.0-openjdk-amd64
```

```
fi
```

```
git clone --depth=1 https://github.com/stevenlovegrove/Pangolin.git
```

```
git clone --depth=1 https://github.com/occipital/OpenNI2.git
```

```
git clone --depth=1 https://github.com/martinruenz/densecrf.git
```

```
git clone --depth=1 https://github.com/carlren/gSLICr.git
```

```
if [ ! -d opencv-3.1.0/install ]; then
```

```
wget https://github.com/Itseez/opencv/archive/3.1.0.zip
```

```
unzip 3.1.0.zip
```

```
rm 3.1.0.zip
```

```
cd opencv-3.1.0
```

```
mkdir -p build

cd build

cmake \

-DCMAKE_BUILD_TYPE=Release \

-DCMAKE_INSTALL_PREFIX=""`pwd`/../install" \

\

`# OpenCV: (building is not possible when
DBUILD_opencv_video/_videoio is OFF?)` \

-DBUILD_opencv_flann=ON \

-DWITH_CUDA=OFF \

-DBUILD_DOCS=OFF \

-DBUILD_TESTS=OFF \

-DBUILD_PERF_TESTS=OFF \

-DBUILD_opencv_java=OFF \

-DBUILD_opencv_python2=OFF \

-DBUILD_opencv_python3=OFF \

-DBUILD_opencv_features2d=ON \

-DBUILD_opencv_calib3d=ON \

-DBUILD_opencv_objdetect=ON \

-DBUILD_opencv_stitching=OFF \

-DBUILD_opencv_superres=OFF \
```

```
-DBUILD_opencv_shape=OFF \  
  
-DWITH_1394=OFF \  
  
-DWITH_GSTREAMER=OFF \  
  
-DWITH_GPHOTO2=OFF \  
  
-DWITH_MATLAB=OFF \  
  
-DWITH_TIFF=OFF \  
  
-DWITH_VTK=OFF \  
  
\  
  
`# OpenCV-Contrib:` \  
  
-DBUILD_opencv_surface_matching=ON \  
  
-DBUILD_opencv_aruco=OFF \  
  
-DBUILD_opencv_bgsegm=OFF \  
  
-DBUILD_opencv_bioinspired=OFF \  
  
-DBUILD_opencv_ccalib=OFF \  
  
-DBUILD_opencv_contrib_world=OFF \  
  
-DBUILD_opencv_datasets=OFF \  
  
-DBUILD_opencv_dnn=OFF \  
  
-DBUILD_opencv_dpm=OFF \  
  
-DBUILD_opencv_face=OFF \  
  
-DBUILD_opencv_fuzzy=OFF \  

```

```
-DBUILD_opencv_line_descriptor=OFF \  
-DBUILD_opencv_matlab=OFF \  
-DBUILD_opencv_optflow=OFF \  
-DBUILD_opencv_plot=OFF \  
-DBUILD_opencv_reg=OFF \  
-DBUILD_opencv_rgbd=OFF \  
-DBUILD_opencv_saliency=OFF \  
-DBUILD_opencv_stereo=OFF \  
-DBUILD_opencv_structured_light=OFF \  
-DBUILD_opencv_text=OFF \  
-DBUILD_opencv_tracking=OFF \  
-DBUILD_opencv_xfeatures2d=OFF \  
-DBUILD_opencv_ximgproc=OFF \  
-DBUILD_opencv_xobjdetect=OFF \  
-DBUILD_opencv_xphoto=OFF \  
  
..  
  
make -j8  
  
make install > /dev/null  
  
cd ../build  
  
OpenCV_DIR=$(pwd)
```

```
cd ../../
```

```
fi
```

```
if [ -z "${BOOST_ROOT}" -a ! -d boost ]; then
```

```
  wget -O boost_1_62_0.tar.bz2
```

```
  https://sourceforge.net/projects/boost/files/boost/1.62.0/boost_1_62_0.tar.  
  bz2/download
```

```
  tar -xjvf boost_1_62_0.tar.bz2 > /dev/null
```

```
  rm boost_1_62_0.tar.bz2
```

```
  cd boost_1_62_0
```

```
  mkdir -p ../boost
```

```
  ./bootstrap.sh --prefix=../boost
```

```
  ./b2 --prefix=../boost --with-filesystem install > /dev/null
```

```
  cd ..
```

```
  rm -r boost_1_62_0
```

```
  BOOST_ROOT=$(pwd)/boost
```

```
fi
```

```
fi
```

```
if [ -z "${BOOST_ROOT}" -a -d boost ]; then
```

```
  BOOST_ROOT=$(pwd)/boost
```

fi

build pangolin make sure you also have the typo fixed

cd Pangolin

git pull

mkdir -p build

cd build

cmake -DAVFORMAT_INCLUDE_DIR="" -DCPP11_NO_BOOST=ON ..

make -j8

Pangolin_DIR=\$(pwd)

cd ../../

build OpenNI2

cd OpenNI2

git pull

make -j8

cd ..

build DenseCRF, see: <http://graphics.stanford.edu/projects/drf/>

cd densecrf

```
git pull
```

```
mkdir -p build
```

```
cd build
```

```
cmake \
```

```
-DCMAKE_BUILD_TYPE=Release \
```

```
-DCMAKE_CXX_FLAGS="${CMAKE_CXX_FLAGS} -fPIC" \
```

```
..
```

```
make -j8
```

```
cd ../../
```

```
# build gSLICr, see: http://www.robots.ox.ac.uk/~victor/gslcr/
```

```
cd gSLICr
```

```
git pull
```

```
mkdir -p build
```

```
cd build
```

```
cmake \
```

```
-DOpenCV_DIR="${OpenCV_DIR}" \
```

```
-DCMAKE_BUILD_TYPE=Release \
```

```
-DCUDA_HOST_COMPILER=/usr/bin/gcc-4.9 \
```

```
-DCMAKE_CXX_FLAGS="${CMAKE_CXX_FLAGS} -  
D_FORCE_INLINES" \
```

```
..
```

```
make -j8
```

```
cd ../../
```

```
# build Co-Fusion
```

```
cd ..
```

```
mkdir -p build
```

```
cd build
```

```
cmake \
```

```
-DBOOST_ROOT="${BOOST_ROOT}" \
```

```
-DOpenCV_DIR="${OpenCV_DIR}" \
```

```
-DPangolin_DIR="${Pangolin_DIR}" \
```

```
..
```

```
make -j8
```

```
cd ..
```

```
#To build, do GPUPTest, and to run is the same as ElasticFusion
```

```
Rtabmap
```

```
#Rtabmap resources can be found here: http://introlab.github.io/rtabmap/  
as well as https://github.com/introlab/rtabmap/wiki
```

#To install from scratch on a 14.04 machine

```
sudo add-apt-repository "deb http://archive.ubuntu.com/ubuntu  
$(lsb_release -sc) main universe restricted multiverse"
```

```
sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu trusty main" >  
/etc/apt/sources.list.d/ros-latest.list'
```

```
wget https://raw.githubusercontent.com/ros/rosdistro/master/ros.key -O - |  
sudo apt-key add -
```

```
sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb_release -  
sc) main" > /etc/apt/sources.list.d/ros-latest.list'
```

```
sudo apt-key adv --keyserver hkp://ha.pool.sks-keyservers.net:80 --recv-  
key 421C365BD9FF1F717815A3895523BAEEB01FA116
```

```
sudo apt-get update
```

```
sudo apt-get install libsqlite3-dev libpcl-1.7-all libopencv-dev
```

```
sudo apt-get install ros-indigo-desktop-full
```

```
sudo rosdep init
```

```
rosdep update
```

```
echo "source /opt/ros/indigo/setup.bash" >> ~/.bashrc
```

```
source ~/.bashrc
```

```
sudo apt-get install python-rosinstall
```

```
mkdir -p ~/catkin_ws/src
```

```
cd ~/catkin_ws
```

```
catkin_make
```

```
source devel/setup.bash
```

```
echo "export  
LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/opt/ros/kinetic/lib/x86_64-  
linux-gnu" >> ~/.bashrc
```

```
sudo apt-get install ros-indigo-rtabmap ros-indigo-rtabmap-ros
```

```
sudo apt-get remove ros-indigo-rtabmap ros-indigo-rtabmap-ros
```

```
#put these lines in ~/.bashrc as well
```

```
source /opt/ros/kinetic/setup.bash
```

```
source ~/catkin_ws/devel/setup.bash
```

```
#install g2o and gtsam
```

```
sudo apt-get install libsuitesparse-dev ros-kinetic-libg2o
```

```
cd ~/Downloads
```

```
git clone https://bitbucket.org/gtborg/gtsam.git
```

```
cd gtsam
```

```
mkdir build
```

```
cd build
```

```
cmake ..
```

```
make -j
```

```
sudo make install
```

```
#install rtabmap
```

```
cd ~
```

```
git clone https://github.com/introlab/rtabmap.git rtabmap
```

```
cd rtabmap/build cmake .. [<---double dots included]
```

```
$ make
```

```
sudo make install
```

```
#install the ros integration
```

```
cd ~/catkin_ws
```

```
$git clone https://github.com/introlab/rtabmap_ros.git src/rtabmap_ros
```

```
catkin_make -j1
```

```
#To run, you may do roslaunch rtabmap xxx.launch
```

```
#xxx.launch is the name of a launch file that you create
```

```
#an example can be found here
```

```
https://github.com/introlab/rtabmap\_ros/blob/master/launch/rtabmap.launch
```

```
#The most important fields would be the image input and depth input topics
```

#here is the file below

<launch>

<!-- Convenience launch file to launch odometry, rtabmap and rtabmapviz nodes at once -->

<!-- For stereo:=false

Your RGB-D sensor should be already started with "depth_registration:=true".

Examples:

```
$ roslaunch freenect_launch freenect.launch depth_registration:=true
```

```
$ roslaunch openni2_launch openni2.launch depth_registration:=true -->
```

<!-- For stereo:=true

Your camera should be calibrated and publishing rectified left and right

images + corresponding camera_info msgs. You can use stereo_image_proc for image rectification.

Example:

```
$ roslaunch rtabmap_ros bumblebee.launch -->
```

<!-- Choose between RGB-D and stereo -->

<arg name="stereo" default="false"/>

```
<!-- Choose visualization -->
```

```
<arg name="rtabmapviz" default="true" />
```

```
<arg name="rviz" default="false" />
```

```
<!-- Localization-only mode -->
```

```
<arg name="localization" default="false"/>
```

```
<!-- sim time for convenience, if playing a rosbag -->
```

```
<arg name="use_sim_time" default="false"/>
```

```
<param if="$(arg use_sim_time)" name="use_sim_time" value="true"/>
```

```
<!-- Corresponding config files -->
```

```
<arg name="cfg" default="" /> <!-- To change RTAB-Map's parameters,  
set the path of config file (*.ini) generated by the standalone app -->
```

```
<arg name="gui_cfg" default="~/ros/rtabmap_gui.ini" />
```

```
<arg name="rviz_cfg" default="$(find  
rtabmap_ros)/launch/config/rgbd.rviz" />
```

```
<arg name="frame_id" default="camera_link"/> <!-- Fixed frame id, you  
may set "base_link" or "base_footprint" if they are published -->
```

```
<arg name="odom_frame_id" default="" /> <!-- If set, TF is used to get  
odometry instead of the topic -->
```

```
<arg name="map_frame_id" default="map"/>
```

```
<arg name="ground_truth_frame_id" default=""/> <!-- e.g., "world" -->
```

```
<arg name="ground_truth_base_frame_id" default=""/> <!-- e.g.,  
"tracker", a fake frame matching the frame "frame_id" (but on different TF  
tree) -->
```

```
<arg name="namespace" default="rtabmap"/>
```

```
<arg name="database_path" default="~/ros/rtabmap.db"/>
```

```
<arg name="queue_size" default="10"/>
```

```
<arg name="wait_for_transform" default="0.2"/>
```

```
<arg name="args" default=""/> <!-- delete_db_on_start, udebug -->
```

```
<arg name="rtabmap_args" default="$(arg args)"/> <!-- deprecated, use  
"args" argument -->
```

```
<arg name="launch_prefix" default=""/> <!-- for debugging purpose, it fills  
launch-prefix tag of the nodes -->
```

```
<arg name="output" default="screen"/> <!-- Control node output (screen  
or log) -->
```

```
<!-- if timestamps of the input topics are synchronized using approximate  
or exact time policy-->
```

```
<arg if="$(arg stereo)" name="approx_sync" default="false"/>
```

```
<arg unless="$(arg stereo)" name="approx_sync" default="true"/>
```

```
<!-- RGB-D related topics -->
```

```
<arg name="rgb_topic" default="/camera/rgb/image_rect_color" />

<arg name="depth_topic" default="/camera/depth_registered/image_raw"
/>

<arg name="camera_info_topic" default="/camera/rgb/camera_info" />

<arg name="depth_camera_info_topic" default="$(arg
camera_info_topic)" />


<!-- stereo related topics -->

<arg name="stereo_namespace" default="/stereo_camera"/>

<arg name="left_image_topic" default="$(arg
stereo_namespace)/left/image_rect_color" />

<arg name="right_image_topic" default="$(arg
stereo_namespace)/right/image_rect" /> <!-- using grayscale image for
efficiency -->

<arg name="left_camera_info_topic" default="$(arg
stereo_namespace)/left/camera_info" />

<arg name="right_camera_info_topic" default="$(arg
stereo_namespace)/right/camera_info" />


<!-- Already synchronized RGB-D related topic, with
rtabmap_ros/rgbd_sync nodelet -->

<arg name="rgbd_sync" default="false"/> <!-- pre-sync rgb_topic,
depth_topic, camera_info_topic -->

<arg name="approx_rgbd_sync" default="true"/> <!-- false=exact
synchronization -->
```

```
<arg name="subscribe_rgbd" default="$(arg rgbd_sync)"/>
```

```
<arg name="rgb_topic" default="rgb_image" />
```

```
<arg name="depth_scale" default="1.0" />
```

```
<arg name="compressed" default="false"/> <!-- If you want to subscribe  
to compressed image topics -->
```

```
<arg name="rgb_image_transport" default="compressed"/> <!-- Common  
types: compressed, theora (see "roslaunch image_transport list_transports") -  
-->
```

```
<arg name="depth_image_transport" default="compressedDepth"/> <!--  
Depth compatible types: compressedDepth (see "roslaunch image_transport  
list_transports") -->
```

```
<arg name="subscribe_scan" default="false"/>
```

```
<arg name="scan_topic" default="/scan"/>
```

```
<arg name="subscribe_scan_cloud" default="false"/>
```

```
<arg name="scan_cloud_topic" default="/scan_cloud"/>
```

```
<arg name="scan_normal_k" default="0"/>
```

```
<arg name="visual_odometry" default="true"/> <!-- Launch rtabmap visual  
odometry node -->
```

```
<arg name="icp_odometry" default="false"/> <!-- Launch rtabmap icp  
odometry node -->
```

```
<arg name="odom_topic" default="odom"/> <!-- Odometry topic name -->
```

```
<arg name="vo_frame_id" default="$(arg odom_topic)"/> <!-- Visual/lcp  
odometry frame ID for TF -->
```

```
<arg name="odom_tf_angular_variance" default="1"/> <!-- If TF is used to  
get odometry, this is the default angular variance -->
```

```
<arg name="odom_tf_linear_variance" default="1"/> <!-- If TF is used to  
get odometry, this is the default linear variance -->
```

```
<arg name="odom_args" default=""/> <!-- More arguments for odometry  
(overwrite same parameters in rtabmap_args) -->
```

```
<arg name="odom_sensor_sync" default="false"/>
```

```
<arg name="odom_guess_frame_id" default=""/>
```

```
<arg name="odom_guess_min_translation" default="0"/>
```

```
<arg name="odom_guess_min_rotation" default="0"/>
```

```
<arg name="imu_topic" default="/imu"/> <!-- only used with VIO  
approaches -->
```

```
<arg name="subscribe_user_data" default="false"/> <!-- user data  
synchronized subscription -->
```

```
<arg name="user_data_topic" default="/user_data"/>
```

```
<arg name="user_data_async_topic" default="/user_data_async" /> <!--  
user data async subscription (rate should be lower than map update rate)  
-->
```

```
<!-- These arguments should not be modified directly, see referred topics  
without "_relay" suffix above -->
```

```
<arg if="$(arg compressed)" name="rgb_topic_relay" default="$(arg
rgb_topic)_relay"/>
```

```
<arg unless="$(arg compressed)" name="rgb_topic_relay" default="$(arg
rgb_topic)"/>
```

```
<arg if="$(arg compressed)" name="depth_topic_relay" default="$(arg
depth_topic)_relay"/>
```

```
<arg unless="$(arg compressed)" name="depth_topic_relay"
default="$(arg depth_topic)"/>
```

```
<arg if="$(arg compressed)" name="left_image_topic_relay"
default="$(arg left_image_topic)_relay"/>
```

```
<arg unless="$(arg compressed)" name="left_image_topic_relay"
default="$(arg left_image_topic)"/>
```

```
<arg if="$(arg compressed)" name="right_image_topic_relay"
default="$(arg right_image_topic)_relay"/>
```

```
<arg unless="$(arg compressed)" name="right_image_topic_relay"
default="$(arg right_image_topic)"/>
```

```
<!-- Nodes -->
```

```
<group ns="$(arg namespace)">
```

```
<!-- relays -->
```

```
<group unless="$(arg stereo)">
```

```
<group unless="$(arg subscribe_rgbd)">
```

```
<node if="$(arg compressed)" name="republish_rgb" type="republish"
pkg="image_transport" args="$(arg rgb_image_transport) in:=$(arg
rgb_topic) raw out:=$(arg rgb_topic_relay)" />
```

```
<node if="$(arg compressed)" name="republish_depth" type="republish"
pkg="image_transport" args="$(arg depth_image_transport) in:=$(arg
depth_topic) raw out:=$(arg depth_topic_relay)" />
```

```
</group>
```

```
<group if="$(arg rgbd_sync)">
```

```
<node if="$(arg compressed)" name="republish_rgb" type="republish"
pkg="image_transport" args="$(arg rgb_image_transport) in:=$(arg
rgb_topic) raw out:=$(arg rgb_topic_relay)" />
```

```
<node if="$(arg compressed)" name="republish_depth" type="republish"
pkg="image_transport" args="$(arg depth_image_transport) in:=$(arg
depth_topic) raw out:=$(arg depth_topic_relay)" />
```

```
<node pkg="nodelet" type="nodelet" name="rgbd_sync"
args="standalone rtabmap_ros/rgbd_sync" output="$(arg output)">
```

```
<remap from="rgb/image" to="$(arg rgb_topic_relay)"/>
```

```
<remap from="depth/image" to="$(arg depth_topic_relay)"/>
```

```
<remap from="rgb/camera_info" to="$(arg camera_info_topic)"/>
```

```
<remap from="rgbd_image" to="$(arg rgbd_topic)"/>
```

```
<param name="approx_sync" type="bool" value="$(arg
approx_rgbd_sync)"/>
```

```
<param name="queue_size" type="int" value="$(arg queue_size)"/>
```

```
<param name="depth_scale" type="double" value="$(arg depth_scale)"/>
```

```
</node>
```

```
</group>
```

```
</group>
```

```
<group if="$(arg stereo)">
```

```
<group unless="$(arg subscribe_rgbd)">
```

```
<node if="$(arg compressed)" name="republish_left" type="republish"  
pkg="image_transport" args="$(arg rgb_image_transport) in:=$(arg  
left_image_topic) raw out:=$(arg left_image_topic_relay)" />
```

```
<node if="$(arg compressed)" name="republish_right" type="republish"  
pkg="image_transport" args="$(arg rgb_image_transport) in:=$(arg  
right_image_topic) raw out:=$(arg right_image_topic_relay)" />
```

```
</group>
```

```
<group if="$(arg rgbd_sync)">
```

```
<node if="$(arg compressed)" name="republish_left" type="republish"  
pkg="image_transport" args="$(arg rgb_image_transport) in:=$(arg  
left_image_topic) raw out:=$(arg left_image_topic_relay)" />
```

```
<node if="$(arg compressed)" name="republish_right" type="republish"  
pkg="image_transport" args="$(arg rgb_image_transport) in:=$(arg  
right_image_topic) raw out:=$(arg right_image_topic_relay)" />
```

```
<node pkg="nodelet" type="nodelet" name="rgbd_sync"  
args="standalone rtabmap_ros/stereo_sync" output="$(arg output)">
```

```
<remap from="left/image_rect" to="$(arg left_image_topic_relay)"/>
```

```
<remap from="right/image_rect" to="$(arg right_image_topic_relay)"/>
```

```
<remap from="left/camera_info" to="$(arg left_camera_info_topic)"/>
```

```
<remap from="right/camera_info" to="$(arg right_camera_info_topic)"/>
```

```
<remap from="rgbd_image" to="$(arg rgbd_topic)"/>
```

```
<param name="approx_sync" type="bool" value="$(arg
approx_rgbd_sync)"/>

<param name="queue_size" type="int" value="$(arg queue_size)"/>

</node>

</group>

</group>

<group unless="$(arg rgbd_sync)">

  <node if="$(arg subscribe_rgbd)" name="republish_rgbd_image"
type="relay" pkg="topic_tools" args="$(arg rgbd_topic) $(arg
rgb_topic)_relay" />

</group>

<!-- Visual odometry -->

<group unless="$(arg icp_odometry)">

  <group if="$(arg visual_odometry)">

    <!-- RGB-D Odometry -->

    <node unless="$(arg stereo)" pkg="rtabmap_ros" type="rgb_d_odometry"
name="rgb_d_odometry" output="$(arg output)" args="$(arg rtabmap_args)
$(arg odom_args)" launch-prefix="$(arg launch_prefix)">

    <remap from="rgb/image" to="$(arg rgb_topic_relay)"/>

    <remap from="depth/image" to="$(arg depth_topic_relay)"/>
```

```
<remap from="rgb/camera_info" to="$(arg camera_info_topic)"/>
```

```
<remap if="$(arg rgb_sync)" from="rgb_image" to="$(arg  
rgb_topic)"/>
```

```
<remap unless="$(arg rgb_sync)" from="rgb_image" to="$(arg  
rgb_topic)_relay"/>
```

```
<remap from="odom" to="$(arg odom_topic)"/>
```

```
<param name="frame_id" type="string" value="$(arg frame_id)"/>
```

```
<param name="odom_frame_id" type="string" value="$(arg  
vo_frame_id)"/>
```

```
<param name="ground_truth_frame_id" type="string" value="$(arg  
ground_truth_frame_id)"/>
```

```
<param name="ground_truth_base_frame_id" type="string" value="$(arg  
ground_truth_base_frame_id)"/>
```

```
<param name="wait_for_transform_duration" type="double" value="$(arg  
wait_for_transform)"/>
```

```
<param name="approx_sync" type="bool" value="$(arg approx_sync)"/>
```

```
<param name="config_path" type="string" value="$(arg cfg)"/>
```

```
<param name="queue_size" type="int" value="$(arg queue_size)"/>
```

```
<param name="subscribe_rgb" type="bool" value="$(arg  
subscribe_rgb)"/>
```

```
<param name="guess_frame_id" type="string" value="$(arg  
odom_guess_frame_id)"/>
```

```
<param name="guess_min_translation" type="double" value="$(arg  
odom_guess_min_translation)"/>
```

```
<param name="guess_min_rotation" type="double" value="$(arg  
odom_guess_min_rotation)"/>
```

```
</node>
```

```
<!-- Stereo Odometry -->
```

```
<node if="$(arg stereo)" pkg="rtabmap_ros" type="stereo_odometry"  
name="stereo_odometry" output="$(arg output)" args="$(arg  
rtabmap_args) $(arg odom_args)" launch-prefix="$(arg launch_prefix)">
```

```
<remap from="left/image_rect" to="$(arg left_image_topic_relay)"/>
```

```
<remap from="right/image_rect" to="$(arg right_image_topic_relay)"/>
```

```
<remap from="left/camera_info" to="$(arg left_camera_info_topic)"/>
```

```
<remap from="right/camera_info" to="$(arg right_camera_info_topic)"/>
```

```
<remap if="$(arg rgb_sync)" from="rgb_image" to="$(arg  
rgb_topic)"/>
```

```
<remap unless="$(arg rgb_sync)" from="rgb_image" to="$(arg  
rgb_topic)_relay"/>
```

```
<remap from="odom" to="$(arg odom_topic)"/>
```

```
<remap from="imu" to="$(arg imu_topic)"/>
```

```
<param name="frame_id" type="string" value="$(arg frame_id)"/>
```

```
<param name="odom_frame_id" type="string" value="$(arg  
vo_frame_id)"/>
```

```
<param name="ground_truth_frame_id" type="string" value="$(arg  
ground_truth_frame_id)"/>
```

```
<param name="ground_truth_base_frame_id" type="string" value="$(arg
ground_truth_base_frame_id)"/>
```

```
<param name="wait_for_transform_duration" type="double" value="$(arg
wait_for_transform)"/>
```

```
<param name="approx_sync" type="bool" value="$(arg approx_sync)"/>
```

```
<param name="config_path" type="string" value="$(arg cfg)"/>
```

```
<param name="queue_size" type="int" value="$(arg queue_size)"/>
```

```
<param name="subscribe_rgb" type="bool" value="$(arg
subscribe_rgb)"/>
```

```
<param name="guess_frame_id" type="string" value="$(arg
odom_guess_frame_id)"/>
```

```
<param name="guess_min_translation" type="double" value="$(arg
odom_guess_min_translation)"/>
```

```
<param name="guess_min_rotation" type="double" value="$(arg
odom_guess_min_rotation)"/>
```

```
</node>
```

```
</group>
```

```
</group>
```

```
<!-- ICP Odometry -->
```

```
<node if="$(arg icp_odometry)" pkg="rtabmap_ros" type="icp_odometry"
name="icp_odometry" output="$(arg output)" args="$(arg rtabmap_args)
$(arg odom_args)" launch-prefix="$(arg launch_prefix)">
```

```
<remap from="scan" to="$(arg scan_topic)"/>
```

```
<remap from="scan_cloud" to="$(arg scan_cloud_topic)"/>
```

```
<remap from="odom" to="$(arg odom_topic)"/>
```

```
<param name="frame_id" type="string" value="$(arg frame_id)"/>
```

```
<param name="odom_frame_id" type="string" value="$(arg  
vo_frame_id)"/>
```

```
<param name="ground_truth_frame_id" type="string" value="$(arg  
ground_truth_frame_id)"/>
```

```
<param name="ground_truth_base_frame_id" type="string" value="$(arg  
ground_truth_base_frame_id)"/>
```

```
<param name="wait_for_transform_duration" type="double" value="$(arg  
wait_for_transform)"/>
```

```
<param name="config_path" type="string" value="$(arg cfg)"/>
```

```
<param name="queue_size" type="int" value="$(arg queue_size)"/>
```

```
<param name="guess_frame_id" type="string" value="$(arg  
odom_guess_frame_id)"/>
```

```
<param name="guess_min_translation" type="double" value="$(arg  
odom_guess_min_translation)"/>
```

```
<param name="guess_min_rotation" type="double" value="$(arg  
odom_guess_min_rotation)"/>
```

```
</node>
```

```
<!-- Visual SLAM (robot side) -->
```

```
<!-- args: "delete_db_on_start" and "udebug" -->
```

```
<node name="rtabmap" pkg="rtabmap_ros" type="rtabmap"
output="$(arg output)" args="$(arg rtabmap_args)" launch-prefix="">
```

```
<param if="$(arg stereo)" name="subscribe_depth" type="bool"
value="false"/>
```

```
<param unless="$(arg stereo)" name="subscribe_depth" type="bool"
value="true"/>
```

```
<param name="subscribe_rgb" type="bool" value="$(arg
subscribe_rgb)"/>
```

```
<param name="subscribe_stereo" type="bool" value="$(arg stereo)"/>
```

```
<param name="subscribe_scan" type="bool" value="$(arg
subscribe_scan)"/>
```

```
<param name="subscribe_scan_cloud" type="bool" value="$(arg
subscribe_scan_cloud)"/>
```

```
<param name="subscribe_user_data" type="bool" value="$(arg
subscribe_user_data)"/>
```

```
<param if="$(arg visual_odometry)" name="subscribe_odom_info"
type="bool" value="true"/>
```

```
<param if="$(arg icp_odometry)" name="subscribe_odom_info"
type="bool" value="true"/>
```

```
<param name="frame_id" type="string" value="$(arg frame_id)"/>
```

```
<param name="map_frame_id" type="string" value="$(arg
map_frame_id)"/>
```

```
<param name="odom_frame_id" type="string" value="$(arg
odom_frame_id)"/>
```

```
<param name="ground_truth_frame_id" type="string" value="$(arg
ground_truth_frame_id)"/>
```

```
<param name="ground_truth_base_frame_id" type="string" value="$(arg  
ground_truth_base_frame_id)"/>
```

```
<param name="odom_tf_angular_variance" type="double" value="$(arg  
odom_tf_angular_variance)"/>
```

```
<param name="odom_tf_linear_variance" type="double" value="$(arg  
odom_tf_linear_variance)"/>
```

```
<param name="odom_sensor_sync" type="bool" value="$(arg  
odom_sensor_sync)"/>
```

```
<param name="wait_for_transform_duration" type="double" value="$(arg  
wait_for_transform)"/>
```

```
<param name="database_path" type="string" value="$(arg  
database_path)"/>
```

```
<param name="approx_sync" type="bool" value="$(arg approx_sync)"/>
```

```
<param name="config_path" type="string" value="$(arg cfg)"/>
```

```
<param name="queue_size" type="int" value="$(arg queue_size)"/>
```

```
<param name="scan_normal_k" type="int" value="$(arg  
scan_normal_k)"/>
```

```
<remap from="rgb/image" to="$(arg rgb_topic_relay)"/>
```

```
<remap from="depth/image" to="$(arg depth_topic_relay)"/>
```

```
<remap from="rgb/camera_info" to="$(arg camera_info_topic)"/>
```

```
<remap if="$(arg rgbd_sync)" from="rgbd_image" to="$(arg  
rgbd_topic)"/>
```

```
<remap unless="$(arg rgb_sync)" from="rgb_image" to="$(arg
rgb_topic)_relay"/>
```

```
<remap from="left/image_rect" to="$(arg left_image_topic_relay)"/>
```

```
<remap from="right/image_rect" to="$(arg right_image_topic_relay)"/>
```

```
<remap from="left/camera_info" to="$(arg left_camera_info_topic)"/>
```

```
<remap from="right/camera_info" to="$(arg right_camera_info_topic)"/>
```

```
<remap from="scan" to="$(arg scan_topic)"/>
```

```
<remap from="scan_cloud" to="$(arg scan_cloud_topic)"/>
```

```
<remap from="user_data" to="$(arg user_data_topic)"/>
```

```
<remap from="user_data_async" to="$(arg user_data_async_topic)"/>
```

```
<remap from="odom" to="$(arg odom_topic)"/>
```

```
<!-- localization mode -->
```

```
    <param if="$(arg localization)" name="Mem/IncrementalMemory"
type="string" value="false"/>
```

```
    <param unless="$(arg localization)"
name="Mem/IncrementalMemory" type="string" value="true"/>
```

```
    <param name="Mem/InitWMWithAllNodes" type="string"
value="$(arg localization)"/>
```

```
</node>
```

```
<!-- Visualisation RTAB-Map -->
```

```
<node if="$(arg rtabmapviz)" pkg="rtabmap_ros" type="rtabmapviz"
name="rtabmapviz" args="-d $(arg gui_cfg)" output="$(arg output)"
launch-prefix="$(arg launch_prefix)">
```

```
<param if="$(arg stereo)" name="subscribe_depth" type="bool"
value="false"/>
```

```
<param unless="$(arg stereo)" name="subscribe_depth" type="bool"
value="true"/>
```

```
<param name="subscribe_rgb" type="bool" value="$(arg
subscribe_rgb)"/>
```

```
<param name="subscribe_stereo" type="bool" value="$(arg stereo)"/>
```

```
<param name="subscribe_scan" type="bool" value="$(arg
subscribe_scan)"/>
```

```
<param name="subscribe_scan_cloud" type="bool" value="$(arg
subscribe_scan_cloud)"/>
```

```
<param if="$(arg visual_odometry)" name="subscribe_odom_info"
type="bool" value="true"/>
```

```
<param if="$(arg icp_odometry)" name="subscribe_odom_info"
type="bool" value="true"/>
```

```
<param name="frame_id" type="string" value="$(arg frame_id)"/>
```

```
<param name="odom_frame_id" type="string" value="$(arg
odom_frame_id)"/>
```

```
<param name="wait_for_transform_duration" type="double" value="$(arg
wait_for_transform)"/>
```

```
<param name="queue_size" type="int" value="$(arg queue_size)"/>
```

```
<param name="approx_sync" type="bool" value="$(arg approx_sync)"/>
```

```
<remap from="rgb/image" to="$(arg rgb_topic_relay)"/>
```

```
<remap from="depth/image" to="$(arg depth_topic_relay)"/>
```

```
<remap from="rgb/camera_info" to="$(arg camera_info_topic)"/>
```

```
<remap if="$(arg rgbd_sync)" from="rgbd_image" to="$(arg  
rgbd_topic)"/>
```

```
<remap unless="$(arg rgbd_sync)" from="rgbd_image" to="$(arg  
rgbd_topic)_relay"/>
```

```
<remap from="left/image_rect" to="$(arg left_image_topic_relay)"/>
```

```
<remap from="right/image_rect" to="$(arg right_image_topic_relay)"/>
```

```
<remap from="left/camera_info" to="$(arg left_camera_info_topic)"/>
```

```
<remap from="right/camera_info" to="$(arg right_camera_info_topic)"/>
```

```
<remap from="scan" to="$(arg scan_topic)"/>
```

```
<remap from="scan_cloud" to="$(arg scan_cloud_topic)"/>
```

```
<remap from="odom" to="$(arg odom_topic)"/>
```

```
</node>
```

```
</group>
```

```
<!-- Visualization RVIZ -->
```

```
<node if="$(arg rviz)" pkg="rviz" type="rviz" name="rviz" args="-d $(arg  
viz_cfg)"/>
```

```
<node if="$(arg rviz)" pkg="nodelet" type="nodelet"  
name="points_xyzrgb" args="standalone  
rtabmap_ros/point_cloud_xyzrgb" output="$(arg output)">
```

```
<remap if="$(arg stereo)" from="left/image" to="$(arg  
left_image_topic_relay)"/>
```

```
<remap if="$(arg stereo)" from="right/image" to="$(arg  
right_image_topic_relay)"/>
```

```
<remap if="$(arg stereo)" from="left/camera_info" to="$(arg  
left_camera_info_topic)"/>
```

```
<remap if="$(arg stereo)" from="right/camera_info" to="$(arg  
right_camera_info_topic)"/>
```

```
<remap unless="$(arg subscribe_rgbd)" from="rgb/image" to="$(arg  
rgb_topic_relay)"/>
```

```
<remap unless="$(arg subscribe_rgbd)" from="depth/image" to="$(arg  
depth_topic_relay)"/>
```

```
<remap unless="$(arg subscribe_rgbd)" from="rgb/camera_info"  
to="$(arg camera_info_topic)"/>
```

```
<remap if="$(arg rgbd_sync)" from="rgbd_image" to="$(arg  
rgbd_topic)"/>
```

```
<remap unless="$(arg rgbd_sync)" from="rgbd_image" to="$(arg  
rgbd_topic_relay)"/>
```

```
<remap from="cloud" to="voxel_cloud" />
```

```
<param name="decimation" type="double" value="4"/>
```

```
<param name="voxel_size" type="double" value="0.0"/>
```

```
<param name="approx_sync" type="bool" value="$(arg approx_sync)"/>
```

```
</node>
```

```
</launch>
```

Appendix D: RTABMAP Launch files

- wei_Lidar_Astra.launch:

<launch>

<arg name="gui_cfg" default="/.ros/rtabmap_gui.ini" />

<arg name="launch_prefix" default=""/>

<arg name="output" default="screen"/>

<!-- Odometry from laser scans -->

<!-- We use Hector mapping to generate odometry for us -->

<node if="true" pkg="hector_mapping" type="hector_mapping"
name="hector_mapping" output="screen">

<!-- Frame names -->

<param name="map_frame" value="hector_map" />

<param name="base_frame" value="base_link" />

<param name="odom_frame" value="/scanmatcher_frame" />

<!-- Tf use -->

```
<param name="pub_map_odom_transform" value="false"/>
```

```
<param name="pub_map_scanmatch_transform" value="true"/>
```

```
<param name="pub_odometry" value="true"/>
```

```
<!-- Map size / start point -->
```

```
<param name="map_resolution" value="0.050"/>
```

```
<param name="map_size" value="2048"/>
```

```
<param name="map_multi_res_levels" value="2" />
```

```
<!-- Map update parameters -->
```

```
<param name="map_update_angle_thresh" value="0.06" />
```

```
<param name="map_pub_period" value="2" />
```

```
<!-- Advertising config -->
```

```
<param name="scan_topic" value="/scan"/>
```

```
</node>
```

```
<group ns="rtabmap">

  <node name="rtabmap" pkg="rtabmap_ros" type="rtabmap"
output="screen" args="--delete_db_on_start">

    <param name="frame_id" type="string" value="base_link"/>

    <param name="subscribe_depth" type="bool" value="true"/>

    <param name="subscribe_scan" type="bool" value="true"/>

    <remap from="odom" to="/scanmatch_odom"/>

    <remap from="scan" to="/scan"/>

    <remap from="rgb/image"
to="/camera/rgb/image_rect_color"/>

    <remap from="depth/image"
to="/camera/depth_registered/image_raw"/>

    <remap from="rgb/camera_info" to="/camera/rgb/camera_info"/>

    <param name="queue_size" type="int" value="10"/>

    <!-- RTAB-Map's parameters -->

    <param name="RGBD/NeighborLinkRefining" type="string"
value="true"/>

    <param name="RGBD/ProximityBySpace" type="string"
value="true"/>
```

```
<param name="RGBD/AngularUpdate" type="string"
value="0.01"/>

<param name="RGBD/LinearUpdate" type="string"
value="0.01"/>

<param name="RGBD/OptimizeFromGraphEnd" type="string"
value="false"/>

<param name="Optimizer/Slam2D" type="string"
value="true"/>

<param name="Reg/Strategy" type="string" value="1"/> <!--
1=ICP -->

<param name="Reg/Force3DoF" type="string" value="true"/>

<param name="Vis/MinInliers" type="string" value="5"/>

<param name="Vis/InlierDistance" type="string" value="0.1"/>

<param name="Rtabmap/TimeThr" type="string"
value="700"/>

<param name="Mem/RehearsalSimilarity" type="string"
value="0.45"/>

<param name="Grid/NoiseFilteringRadius" type="string"
value="0.05"/>

<param name="Grid/NoiseFilteringMinNeighbors" type="string"
value="8"/>

<param name="Grid/FromDepth" type="string" value="false"/>

<param name="RGBD/ProximityPathMaxNeighbors" type="string"
value="10"/>
```

```
</node>
```

```
<!-- Visualisation RTAB-Map -->
```

```
<node if="true" pkg="rtabmap_ros" type="rtabmapviz"
name="rtabmapviz" args="-d $(arg gui_cfg)" output="$(arg output)"
launch-prefix="$(arg launch_prefix)">
```

```
<param name="subscribe_depth" type="bool" value="true"/>
```

```
<param name="subscribe_rgbd" type="bool" value="false"/>
```

```
<param name="subscribe_scan" type="bool" value="true"/>
```

```
<param name="subscribe_scan_cloud" type="bool" value="false"/>
```

```
<param name="frame_id" type="string" value="base_link"/>
```

```
<param name="odom_frame_id" type="string" value=""/>
```

```
<param name="wait_for_transform_duration" type="double"
value="0.2"/>
```

```
<param name="queue_size" type="int" value="10"/>
```

```
<param name="approx_sync" type="bool" value="true"/>
```

```
<remap from="rgb/image"
to="/camera/rgb/image_rect_color"/>
```

```
<remap from="depth/image"
to="/camera/depth_registered/image_raw"/>
```

```
<remap from="rgb/camera_info" to="/camera/rgb/camera_info"/>
```

```
<remap from="scan"      to="/scan"/>
```

```
<remap from="odom"      to="/scanmatch_odom"/>
```

```
</node>
```

```
<node if="false" pkg="rviz" type="rviz" name="rviz" args="-d $(find  
rtabmap_ros)/launch/config/demo_robot_mapping.rviz" output="screen"/>
```

```
<node if="false" pkg="nodelet" type="nodelet" name="points_xyzrgb"  
args="standalone rtabmap_ros/point_cloud_xyzrgb" output="$(arg  
output)">
```

```
<remap from="rgb/camera_info" to="/camera/rgb/camera_info"/>
```

```
<remap from="rgb/image" to="/camera/rgb/image_rect_color"/>
```

```
<remap from="depth/image"  
to="/camera/depth_registered/image_raw"/>
```

```
<remap from="cloud"      to="voxel_cloud" />
```

```
<param name="voxel_size" type="double" value="0.01"/>

</node>

<node pkg="tf" type="static_transform_publisher" name="base_to_cam"

      args="0.23 -0.12 0.0 0.0 0.0 0.0 /base_link /camera_link 100" />

<node if="true" pkg="tf" type="static_transform_publisher"
name="odom_to_scanmatchfram"

      args="0.0 0.0 0.0 0.0 0.0 0.0 /odom /scanmatcher_frame 100" />

<node pkg="tf" type="static_transform_publisher" name="base_to_laser"

      args="0.18 0.0 0.145 0 0 0 /base_link /laser 100" />

<node pkg="tf" type="static_transform_publisher"
name="scanmatcher_to_base_link"

      args="0.0 0.0 0.0 0.0 0.0 0.0 /scanmatcher_frame /base_link 100"
/>
```

```
</group>
```

```
</launch>
```

- wei_Lidar_Realsense.launch:

```
<launch>
```

```
<arg name="gui_cfg"      default="~/ros/rtabmap_gui.ini" />
```

```
<arg name="launch_prefix"  default=""/>
```

```
<arg name="output"        default="screen"/>
```

```
<!-- Odometry from laser scans -->
```

```
<!-- We use Hector mapping to generate odometry for us -->
```

```
<node if="true" pkg="hector_mapping" type="hector_mapping"
name="hector_mapping" output="screen">
```

```
<!-- Frame names -->
```

```
<param name="map_frame" value="hector_map" />
```

```
<param name="base_frame" value="base_link" />

<param name="odom_frame" value="/scanmatcher_frame" />

<!-- Tf use -->

<param name="pub_map_odom_transform" value="false"/>

<param name="pub_map_scanmatch_transform" value="true"/>

<param name="pub_odometry" value="true"/>

<!-- Map size / start point -->

<param name="map_resolution" value="0.050"/>

<param name="map_size" value="2048"/>

<param name="map_multi_res_levels" value="2" />

<!-- Map update parameters -->

<param name="map_update_angle_thresh" value="0.06" />

<param name="map_pub_period" value="2" />

<!-- Advertising config -->

<param name="scan_topic" value="/scan"/>

</node>
```

```
<group ns="rtabmap">

  <node name="rtabmap" pkg="rtabmap_ros" type="rtabmap"
output="screen" args="--delete_db_on_start">

    <param name="frame_id" type="string" value="base_link"/>

    <param name="subscribe_depth" type="bool" value="true"/>

    <param name="subscribe_scan" type="bool" value="true"/>

    <remap from="odom" to="/scanmatch_odom"/>

    <remap from="scan" to="/scan"/>

    <remap from="rgb/image" to="/camera/color/image_raw"/>

    <remap from="depth/image"
to="/camera/aligned_depth_to_color/image_raw"/>

    <remap from="rgb/camera_info" to="/camera/color/camera_info"/>

    <param name="queue_size" type="int" value="10"/>

    <!-- RTAB-Map's parameters -->

    <param name="RGBD/NeighborLinkRefining" type="string"
value="true"/>
```

```
<param name="RGBD/ProximityBySpace" type="string"
value="true"/>
```

```
<param name="RGBD/AngularUpdate" type="string"
value="0.01"/>
```

```
<param name="RGBD/LinearUpdate" type="string"
value="0.01"/>
```

```
<param name="RGBD/OptimizeFromGraphEnd" type="string"
value="false"/>
```

```
<param name="Optimizer/Slam2D" type="string"
value="true"/>
```

```
<param name="Reg/Strategy" type="string" value="1"/> <!--
1=ICP -->
```

```
<param name="Reg/Force3DoF" type="string" value="true"/>
```

```
<param name="Vis/MinInliers" type="string" value="5"/>
```

```
<param name="Vis/InlierDistance" type="string" value="0.1"/>
```

```
<param name="Rtabmap/TimeThr" type="string"
value="700"/>
```

```
<param name="Mem/RehearsalSimilarity" type="string"
value="0.45"/>
```

```
<param name="Grid/NoiseFilteringRadius" type="string"
value="0.05"/>
```

```
<param name="Grid/NoiseFilteringMinNeighbors" type="string"
value="8"/>
```

```
<param name="Grid/FromDepth" type="string" value="false"/>
```

```
<param name="RGBD/ProximityPathMaxNeighbors" type="string"
value="10"/>
```

```
</node>
```

```
<!-- Visualisation RTAB-Map -->
```

```
<node if="true" pkg="rtabmap_ros" type="rtabmapviz"
name="rtabmapviz" args="-d $(arg gui_cfg)" output="$(arg output)"
launch-prefix="$(arg launch_prefix)">
```

```
<param name="subscribe_depth" type="bool" value="true"/>
```

```
<param name="subscribe_rgbd" type="bool" value="false"/>
```

```
<param name="subscribe_scan" type="bool" value="true"/>
```

```
<param name="subscribe_scan_cloud" type="bool" value="false"/>
```

```
<param name="frame_id" type="string" value="base_link"/>
```

```
<param name="odom_frame_id" type="string" value=""/>
```

```
<param name="wait_for_transform_duration" type="double"
value="0.2"/>
```

```
<param name="queue_size" type="int" value="10"/>
```

```
<param name="approx_sync" type="bool" value="true"/>
```

```
<remap from="rgb/image" to="/camera/color/image_raw"/>
```

```
<remap from="depth/image"
to="/camera/aligned_depth_to_color/image_raw"/>
```

```
<remap from="rgb/camera_info" to="/camera/color/camera_info"/>
```

```
<remap from="scan"      to="/scan"/>
```

```
<remap from="odom"      to="/scanmatch_odom"/>
```

```
</node>
```

```
<node if="false" pkg="rviz" type="rviz" name="rviz" args="-d $(find  
rtabmap_ros)/launch/config/demo_robot_mapping.rviz" output="screen"/>
```

```
<node if="false" pkg="nodelet" type="nodelet" name="points_xyzrgb"  
args="standalone rtabmap_ros/point_cloud_xyzrgb" output="$(arg  
output)">
```

```
<remap from="rgb/camera_info" to="/camera/rgb/camera_info"/>
```

```
<remap from="rgb/image" to="/camera/rgb/image_rect_color"/>
```

```
<remap from="depth/image"  
to="/camera/depth_registered/image_raw"/>
```

```
<remap from="cloud"      to="voxel_cloud" />
```

```

    <param name="voxel_size" type="double" value="0.01"/>

</node>

<node pkg="tf" type="static_transform_publisher" name="base_to_cam"
      args="0.275 0.0 0.27 0.0 0.0 0.0 /base_link /camera_link 100" />

<node pkg="tf" type="static_transform_publisher" name="base_to_laser"
      args="0.295 0.0 0.04 0 0 0 /base_link /laser 100" />

<node if="true" pkg="tf" type="static_transform_publisher"
name="odom_to_scanmatchfram"

      args="0.0 0.0 0.0 0.0 0.0 0.0 /odom /scanmatcher_frame 100" />

</group>

</launch>

```

- wei_LAW.launch:

```

<launch>

  <arg name="gui_cfg"                default="~/ros/rtabmap_gui.ini" />

  <arg name="launch_prefix"          default=""/>

```

```
<arg name="output"                default="screen"/>
```

```
<group ns="rtabmap">
```

```
<node name="rtabmap" pkg="rtabmap_ros" type="rtabmap"  
output="screen" args="--delete_db_on_start">
```

```
<param name="frame_id" type="string" value="base_link"/>
```

```
<param name="subscribe_depth" type="bool"  
value="true"/>
```

```
<param name="subscribe_scan" type="bool" value="true"/>
```

```
<remap from="odom" to="/wheel_odom"/>
```

```
<remap from="scan" to="/scan"/>
```

```
<remap from="rgb/image" to="/camera/rgb/image_rect_color"/>
```

```
<remap from="depth/image"  
to="/camera/depth_registered/image_raw"/>
```

```
<remap from="rgb/camera_info" to="/camera/rgb/camera_info"/>
```

```
<param name="queue_size" type="int" value="10"/>
```

```
<!-- RTAB-Map's parameters -->

<param name="RGBD/NeighborLinkRefining" type="string"
value="true"/>

<param name="RGBD/ProximityBySpace" type="string"
value="true"/>

<param name="RGBD/AngularUpdate" type="string"
value="0.01"/>

<param name="RGBD/LinearUpdate" type="string"
value="0.01"/>

<param name="RGBD/OptimizeFromGraphEnd"
type="string" value="false"/>

<param name="Reg/Force3DoF" type="string"
value="true"/> <!--same as optimizer/slam2D -->

<param name="Reg/Strategy" type="string"
value="1"/> <!-- 1=ICP -->

<param name="Reg/Force3DoF" type="string"
value="true"/>

<param name="Vis/MinInliers" type="string"
value="5"/>

<param name="Vis/InlierDistance" type="string"
value="0.1"/>

<param name="Rtabmap/TimeThr" type="string"
value="700"/>

<param name="Mem/RehearsalSimilarity" type="string"
value="0.45"/>

<param name="Grid/CellSize" type="string" value="0.05"/>
<!-- 5cm voxel default-->
```

```

        <param name="Kp/MaxFeatures" type="string"
value="400"/>

        <param name="Optimizer/Strategy" type="string" value="0"/> <!-- 0
is TORO, Default G20-->

        <param name="RGBD/OptimizeMaxError" type="string"
value="0"/> <!--When using TORO, seto to 0,otherwise set to 1-->

        <param name="Kp/DetectorStrategy" type="string" value="6"/> <!--
0=SURF 1=SIFT 2=ORB 3=FAST/FREAK 4=FAST/BRIEF
5=GFTT/FREAK 6=GFTT/BRIEF(default) 7=BRISK 8=GFTT/ORB
9=KAZE-->

        <param name="RGBD/LoopClosureReextractFeatures"
type="string" value="true"/> <!--extract features even if there are some
already in the node-->

        <param name="Grid/FromDepth" type="string" value="false"/> <!--
suppress warning (subscribe_scan=true)-->

        <param name="RGBD/ProximityPathMaxNeighbors" type="string"
value="10"/> <!--suppress warning (subscribe_scan = true and
Reg/Strategy=1-->

        <param name="Vis/FeatureType" type="string" value="6"/> <!--
suppress warning,want to be same as KP/detectorstrategy -->

</node>

<!-- Visualisation RTAB-Map -->

<node pkg="rtabmap_ros" type="rtabmapviz" name="rtabmapviz" args="-
d $(arg gui_cfg)" output="$(arg output)" launch-prefix="$(arg
launch_prefix)">

```

```
<param name="subscribe_rgbd" type="bool" value="false"/>
<param name="subscribe_scan" type="bool" value="true"/>
<param name="subscribe_scan_cloud" type="bool" value="false"/>
<param name="frame_id" type="string" value="base_link"/>
<param name="odom_frame_id" type="string" value=""/>
<param name="wait_for_transform_duration" type="double" value="0.2"/>
<param name="queue_size" type="int" value="10"/>
<param name="approx_sync" type="bool" value="true"/>
```

```
    <remap from="rgb/image" to="/camera/rgb/image_rect_color"/>
    <remap from="depth/image"
to="/camera/depth_registered/image_raw"/>
    <remap from="rgb/camera_info" to="/camera/rgb/camera_info"/>
<remap from="scan" to="/scan"/>
```

```
<remap from="odom" to="/wheel_odom"/>
</node>
```

```
<node pkg="tf" type="static_transform_publisher" name="base_to_laser"
args="0.098 0.0 0.0 0 0 0 /base_link /laser 100" />
```

```
<node pkg="tf" type="static_transform_publisher" name="base_to_cam"
      args="0.198 0.0 0.0 0.0 0 0 /base_link /camera_link 100" />
```

```
</group>
```

```
</launch>
```

- wei_LRW.launch:

```
<launch>
```

```
<arg name="gui_cfg"          default="~/ros/rtabmap_gui.ini" />
```

```
<arg name="launch_prefix"    default=""/>
```

```
<arg name="output"           default="screen"/>
```

```
<group ns="rtabmap">
```

```
<node name="rtabmap" pkg="rtabmap_ros" type="rtabmap"
output="screen" args="--delete_db_on_start">
```

```
<param name="frame_id" type="string" value="base_link"/>
```

```
<param name="subscribe_depth" type="bool"
value="true"/>
```

```
<param name="subscribe_scan" type="bool" value="true"/>
```

```
<remap from="odom" to="/wheel_odom"/>
```

```
<remap from="scan" to="/scan"/>
```

```
<remap from="rgb/image" to="/camera/color/image_raw"/>
```

```
<remap from="depth/image"
to="/camera/aligned_depth_to_color/image_raw"/>
```

```
<remap from="rgb/camera_info" to="/camera/color/camera_info"/>
```

```
<param name="queue_size" type="int" value="10"/>
```

```
<!-- RTAB-Map's parameters -->
```

```
<param name="RGBD/NeighborLinkRefining" type="string"
value="true"/>
```

```
<param name="RGBD/ProximityBySpace" type="string"
value="true"/>
```

```
<param name="RGBD/AngularUpdate" type="string"
value="0.01"/>
```

```
<param name="RGBD/LinearUpdate" type="string"
value="0.01"/>
```

```

        <param name="RGBD/OptimizeFromGraphEnd"
type="string" value="false"/>

        <param name="Reg/Force3DoF"                type="string"
value="true"/> <!--same as optimizer/slam2D -->

        <param name="Reg/Strategy"                type="string"
value="1"/> <!-- 1=ICP -->

        <param name="Reg/Force3DoF"                type="string"
value="true"/>

        <param name="Vis/MinInliers"                type="string"
value="5"/>

        <param name="Vis/InlierDistance"            type="string"
value="0.1"/>

        <param name="Rtabmap/TimeThr"                type="string"
value="700"/>

        <param name="Mem/RehearsalSimilarity" type="string"
value="0.45"/>

        <param name="Grid/CellSize" type="string" value="0.05"/>
<!-- 5cm voxel default-->

        <param name="Kp/MaxFeatures" type="string"
value="400"/>

        <param name="Optimizer/Strategy" type="string" value="0"/> <!-- 0
is TORO, Default G20-->

        <param name="RGBD/OptimizeMaxError" type="string"
value="0"/> <!--When using TORO, seto to 0,otherwise set to 1-->

        <param name="Kp/DetectorStrategy" type="string" value="6"/> <!--
0=SURF 1=SIFT 2=ORB 3=FAST/FREAK 4=FAST/BRIEF
5=GFTT/FREAK 6=GFTT/BRIEF(default) 7=BRISK 8=GFTT/ORB
9=KAZE-->

```

```
<param name="RGBD/LoopClosureReextractFeatures"
type="string" value="true"/> <!--extract features even if there are some
already in the node-->
```

```
<param name="Grid/FromDepth" type="string" value="false"/> <!--
suppress warning (subscribe_scan=true)-->
```

```
<param name="RGBD/ProximityPathMaxNeighbors" type="string"
value="10"/> <!--suppress warning (subscribe_scan = true and
Reg/Strategy=1-->
```

```
<param name="Vis/FeatureType" type="string" value="6"/> <!--
suppress warning,want to be same as KP/detectorstrategy -->
```

```
</node>
```

```
<!-- Visualisation RTAB-Map -->
```

```
<node pkg="rtabmap_ros" type="rtabmapviz" name="rtabmapviz" args="-
d $(arg gui_cfg)" output="$(arg output)" launch-prefix="$(arg
launch_prefix)">
```

```
<param name="subscribe_rgbd" type="bool" value="false"/>
```

```
<param name="subscribe_scan" type="bool" value="true"/>
```

```
<param name="subscribe_scan_cloud" type="bool" value="false"/>
```

```
<param name="frame_id" type="string" value="base_link"/>
```

```
<param name="odom_frame_id" type="string" value=""/>
```

```
<param name="wait_for_transform_duration" type="double" value="0.2"/>
```

```
<param name="queue_size" type="int" value="10"/>
```

```
<param name="approx_sync"          type="bool" value="true"/>
```

```
    <remap from="rgb/image" to="/camera/color/image_raw"/>
```

```
    <remap from="depth/image"
to="/camera/aligned_depth_to_color/image_raw"/>
```

```
    <remap from="rgb/camera_info" to="/camera/color/camera_info"/>
```

```
<remap from="scan"          to="/scan"/>
```

```
<remap from="odom"          to="/wheel_odom"/>
```

```
</node>
```

```
<node pkg="tf" type="static_transform_publisher" name="base_to_laser"
```

```
  args="0.098 0.0 0.0 0 0 0 /base_link /laser 100" />
```

```
<node pkg="tf" type="static_transform_publisher" name="base_to_cam"
```

```
  args="0.198 0.0 0.0 0.0 0 0 /base_link /camera_link 100" />
```

```
</group>
```

```
</launch>
```

- wei_Pozyx_full_Astra.launch:

```
<launch>
```

```
<arg name="gui_cfg"      default="~/ros/rtabmap_gui.ini" />
```

```
<arg name="launch_prefix"  default=""/>
```

```
<arg name="output"        default="screen"/>
```

```
<!-- Odometry from laser scans -->
```

```
<!-- We use Hector mapping to generate odometry for us -->
```

```
<node if="true" pkg="hector_mapping" type="hector_mapping"
name="hector_mapping" output="screen">
```

```
<!-- Frame names -->
```

```
<param name="map_frame" value="hector_map" />
```

```
<param name="base_frame" value="base_link" />
```

```
<param name="odom_frame" value="/scanmatcher_frame" />
```

```
<!-- Tf use -->
```

```
<param name="pub_map_odom_transform" value="false"/>
```

```
<param name="pub_map_scanmatch_transform" value="true"/>
```

```
<param name="pub_odometry" value="true"/>
```

```
<!-- Map size / start point -->

<param name="map_resolution" value="0.050"/>

<param name="map_size" value="2048"/>

<param name="map_multi_res_levels" value="2" />


<!-- Map update parameters -->

<param name="map_update_angle_thresh" value="0.06" />

<param name="map_pub_period" value="2" />


<!-- Advertising config -->

<param name="scan_topic" value="/scan"/>

</node>


<group ns="rtabmap">

  <node name="rtabmap" pkg="rtabmap_ros" type="rtabmap"
output="screen" args="--delete_db_on_start">

    <param name="frame_id" type="string" value="base_link"/>


    <param name="subscribe_depth" type="bool" value="true"/>

    <param name="subscribe_scan" type="bool" value="true"/>

  </node>

</group>
```

```
<remap from="odom" to="/odometry/filtered"/>

<remap from="scan" to="/scan"/>

<remap from="rgb/image" to="/camera/rgb/image_rect_color"/>

<remap from="depth/image"
to="/camera/depth_registered/image_raw"/>

<remap from="rgb/camera_info" to="/camera/rgb/camera_info"/>


<param name="queue_size" type="int" value="10"/>


<!-- RTAB-Map's parameters -->

<param name="RGBD/NeighborLinkRefining" type="string"
value="true"/>

<param name="RGBD/ProximityBySpace" type="string"
value="true"/>

<param name="RGBD/AngularUpdate" type="string"
value="0.01"/>

<param name="RGBD/LinearUpdate" type="string"
value="0.01"/>

<param name="RGBD/OptimizeFromGraphEnd" type="string"
value="false"/>

<param name="Optimizer/Slam2D" type="string"
value="true"/>

<param name="Reg/Strategy" type="string" value="1"/> <!--
1=ICP -->
```

```
<param name="Vis/MinInliers" type="string" value="5"/>

<param name="Vis/InlierDistance" type="string" value="0.1"/>

<param name="Rtabmap/TimeThr" type="string"
value="700"/>

<param name="Mem/RehearsalSimilarity" type="string"
value="0.45"/>

<param name="Grid/NoiseFilteringRadius" type="string"
value="0.05"/>

<param name="Grid/NoiseFilteringMinNeighbors" type="string"
value="8"/>

<param name="Grid/FromDepth" type="string" value="false"/>

<param name="RGBD/ProximityPathMaxNeighbors" type="string"
value="10"/>

</node>

<!-- Visualisation RTAB-Map -->

<node if="true" pkg="rtabmap_ros" type="rtabmapviz"
name="rtabmapviz" args="-d $(arg gui_cfg)" output="$(arg output)"
launch-prefix="$(arg launch_prefix)">

<param name="subscribe_depth" type="bool" value="true"/>

<param name="subscribe_rgb" type="bool" value="false"/>

<param name="subscribe_scan" type="bool" value="true"/>
```

```
<param name="subscribe_scan_cloud" type="bool" value="false"/>

<param name="frame_id" type="string" value="base_link"/>

<param name="odom_frame_id" type="string" value=""/>

<param name="wait_for_transform_duration" type="double"
value="0.2"/>

<param name="queue_size" type="int" value="10"/>

<param name="approx_sync" type="bool" value="true"/>


<remap from="rgb/image" to="/camera/rgb/image_rect_color"/>

<remap from="depth/image"
to="/camera/depth_registered/image_raw"/>

<remap from="rgb/camera_info" to="/camera/rgb/camera_info"/>


<remap from="scan" to="/scan"/>


<remap from="odom" to="/odometry/filtered"/>

</node>


<node pkg="tf" type="static_transform_publisher" name="base_to_cam"

args="0.18 0.0 0.0 0.0 0.0 0.0 /base_link /camera_link 100" />
```

```
<node if="true" pkg="tf" type="static_transform_publisher"
name="odom_to_scanmatchframe"

    args="0.0 0.0 0.0 0.0 0.0 0.0 /odom /scanmatcher_frame 100" />


<node pkg="tf" type="static_transform_publisher" name="base_to_laser"

    args="0.1 0.0 0.0 0 0 0 /base_link /laser 100" />


</group>

</launch>
```

- wei_Pozyx_full_rs.launch:

```
<launch>

<arg name="gui_cfg"      default="/.ros/rtabmap_gui.ini" />

<arg name="launch_prefix"    default=""/>

<arg name="output"      default="screen"/>
```

```
<!-- Odometry from laser scans -->
```

```
<!-- We use Hector mapping to generate odometry for us -->
```

```
<node if="true" pkg="hector_mapping" type="hector_mapping"
name="hector_mapping" output="screen">
```

```
<!-- Frame names -->
```

```
<param name="map_frame" value="hector_map" />
```

```
<param name="base_frame" value="base_link" />
```

```
<param name="odom_frame" value="/scanmatcher_frame" />
```

```
<!-- Tf use -->
```

```
<param name="pub_map_odom_transform" value="false"/>
```

```
<param name="pub_map_scanmatch_transform" value="true"/>
```

```
<param name="pub_odometry" value="true"/>
```

```
<!-- Map size / start point -->
```

```
<param name="map_resolution" value="0.050"/>
```

```
<param name="map_size" value="2048"/>
```

```
<param name="map_multi_res_levels" value="2" />
```

```
<!-- Map update parameters -->
```

```
<param name="map_update_angle_thresh" value="0.06" />
```

```
<param name="map_pub_period" value="2" />
```

```
<!-- Advertising config -->

<param name="scan_topic" value="/scan"/>

</node>

<group ns="rtabmap">

  <node name="rtabmap" pkg="rtabmap_ros" type="rtabmap"
output="screen" args="--delete_db_on_start">

    <param name="frame_id" type="string" value="base_link"/>

    <param name="subscribe_depth" type="bool" value="true"/>

    <param name="subscribe_scan" type="bool" value="true"/>

    <remap from="odom" to="/odometry/filtered"/>

    <remap from="scan" to="/scan"/>

    <remap from="rgb/image" to="/camera/color/image_raw"/>

    <remap from="depth/image"
to="/camera/aligned_depth_to_color/image_raw"/>

    <remap from="rgb/camera_info" to="/camera/color/camera_info"/>

    <param name="queue_size" type="int" value="10"/>
```

```
<!-- RTAB-Map's parameters -->

<param name="RGBD/NeighborLinkRefining" type="string"
value="true"/>

<param name="RGBD/ProximityBySpace" type="string"
value="true"/>

<param name="RGBD/AngularUpdate" type="string"
value="0.01"/>

<param name="RGBD/LinearUpdate" type="string"
value="0.01"/>

<param name="RGBD/OptimizeFromGraphEnd" type="string"
value="false"/>

<param name="Optimizer/Slam2D" type="string"
value="true"/>

<param name="Reg/Strategy" type="string" value="1"/> <!--
1=ICP -->

<param name="Vis/MinInliers" type="string" value="5"/>

<param name="Vis/InlierDistance" type="string" value="0.1"/>

<param name="Rtabmap/TimeThr" type="string"
value="700"/>

<param name="Mem/RehearsalSimilarity" type="string"
value="0.45"/>

<param name="Grid/NoiseFilteringRadius" type="string"
value="0.05"/>
```

```
<param name="Grid/NoiseFilteringMinNeighbors" type="string"
value="8"/>

<param name="Grid/FromDepth" type="string" value="false"/>

<param name="RGBD/ProximityPathMaxNeighbors" type="string"
value="10"/>

</node>

<!-- Visualisation RTAB-Map -->

<node if="true" pkg="rtabmap_ros" type="rtabmapviz"
name="rtabmapviz" args="-d $(arg gui_cfg)" output="$(arg output)"
launch-prefix="$(arg launch_prefix)">

  <param name="subscribe_depth" type="bool" value="true"/>

  <param name="subscribe_rgb" type="bool" value="false"/>

  <param name="subscribe_scan" type="bool" value="true"/>

  <param name="subscribe_scan_cloud" type="bool" value="false"/>

  <param name="frame_id" type="string" value="base_link"/>

  <param name="odom_frame_id" type="string" value=""/>

  <param name="wait_for_transform_duration" type="double"
value="0.2"/>

  <param name="queue_size" type="int" value="10"/>

  <param name="approx_sync" type="bool" value="true"/>
```

```
<remap from="rgb/image" to="/camera/color/image_raw"/>
```

```
<remap from="depth/image"  
to="/camera/aligned_depth_to_color/image_raw"/>
```

```
<remap from="rgb/camera_info" to="/camera/color/camera_info"/>
```

```
<remap from="scan"      to="/scan"/>
```

```
<remap from="odom"      to="/odometry/filtered"/>
```

```
</node>
```

```
<node pkg="tf" type="static_transform_publisher" name="base_to_cam"  
  
args="0.28 0.0 0.02 0.0 0.0 0.0 /base_link /camera_link 100" />
```

```
<node if="true" pkg="tf" type="static_transform_publisher"  
name="odom_to_scanmatchframe"  
  
args="0.0 0.0 0.0 0.0 0.0 0.0 /odom /scanmatcher_frame 100" />
```

```
<node pkg="tf" type="static_transform_publisher" name="base_to_laser"  
  
args="0.145 0.0 0.246 0 0 0 /base_link /laser 100" />
```

</group>

</launch>

Appendix E: POZYX

This section contains the code used for controlling the Pozyx sensor, and the code used for fusing different setups of sensors with robot_localization.

Pozyx Code

imu.ino – This file is the Arduino code that publishes the IMU data from the Pozyx to the topic “Pozyx_imu”.

```
#include <Pozyx.h>
#include <Pozyx_definitions.h>
#include <Wire.h>
#include <ros.h>
#include <geometry_msgs/PoseWithCovarianceStamped.h>
#include <sensor_msgs/Imu.h>
#define TWO_PI 6.283185307179586476925286766559

////////////////////////////////////
////////////////// PARAMETERS //////////////////////////////////
////////////////////////////////////

uint16_t remote_id = 0x6e75;
// set this to the ID of the remote device
bool remote = false; // set this to true to use the remote ID

//boolean use_processing = false; // set this to true to output data for the processing sketch

const uint8_t algorithm = POZYX_POS_ALG_TRACKING; // positioning algorithm to use. try
POZYX_POS_ALG_TRACKING for fast moving objects.
const uint8_t dimension = POZYX_3D; // positioning dimension
const int32_t height = 1000; // height of device, required in 2.5D positioning

ros::NodeHandle nh;

sensor_msgs::Imu imu_pub;

ros::Publisher pub2("Pozyx_imu", &imu_pub);

////////////////////////////////////

void setup(){
  Serial.begin(57600);
```

```
if(Pozyx.begin(false, MODE_INTERRUPT, POZYX_INT_MASK_IMU) ==
POZYX_FAILURE){
    abort();
}

nh.initNode();
nh.advertise(pub2);

if(!remote){
    remote_id = NULL;
}

// clear all previous devices in the device list
Pozyx.clearDevices(remote_id);
// sets the positioning algorithm
Pozyx.setPositionAlgorithm(algorithm, dimension, remote_id);

delay(100);
}

void loop(){
    quaternion_t quaternion;
    angular_vel_t angVel;
    linear_acceleration_t acceleration;

    memset(&quaternion, 0, sizeof(quaternion_t));
    memset(&angVel, 0, sizeof(angVel));
    memset(&acceleration, 0, sizeof(acceleration));

    if(remote){
        Pozyx.getQuaternion(&quaternion, remote_id);
        Pozyx.getAngularVelocity_dps(&angVel, remote_id);
        Pozyx.getLinearAcceleration_mg(&acceleration, remote_id);

    } else {
        Pozyx.getQuaternion(&quaternion);
        Pozyx.getAngularVelocity_dps(&angVel);
        Pozyx.getLinearAcceleration_mg(&acceleration);
    }
    printImu(quaternion, angVel, acceleration);
}

// exports pose to ROS
void printImu(quaternion_t quat, angular_vel_t angVel, linear_acceleration_t accel){

    imu_pub.orientation.x = quat.x;
    imu_pub.orientation.y = quat.y;
```

```

imu_pub.orientation.z = quat.z;
imu_pub.orientation.w = quat.weight;
imu_pub.orientation_covariance[0] = 1.0;
imu_pub.orientation_covariance[4] = 1.0;
imu_pub.orientation_covariance[8] = 1.0;

imu_pub.angular_velocity.x = angVel.x / 360 * TWO_PI;
imu_pub.angular_velocity.y = angVel.y / 360 * TWO_PI;
imu_pub.angular_velocity.z = angVel.z / 360 * TWO_PI;
imu_pub.angular_velocity_covariance[0] = 1.0;
imu_pub.angular_velocity_covariance[4] = 1.0;
imu_pub.angular_velocity_covariance[8] = 1.0;

imu_pub.linear_acceleration.x = ((float) accel.x) / 1000 * 9.8;
imu_pub.linear_acceleration.y = ((float) accel.y) / 1000 * 9.8;
imu_pub.linear_acceleration.z = ((float) accel.z) / 1000 * 9.8;
imu_pub.linear_acceleration_covariance[0] = 1.0;
imu_pub.linear_acceleration_covariance[4] = 1.0;
imu_pub.linear_acceleration_covariance[8] = 1.0;

pub2.publish(&imu_pub);

nh.spinOnce();

}

```

pose_imu.ino – This file is the Arduino code that publishes the IMU data from the Pozyx to the topic “Pozyx_imu”, and UWB data from the Pozyx to topic “Pozyx_pose”.

```

// Please read the ready-to-localize tutorial together with this example.
// https://www.pozyx.io/Documentation/Tutorials/ready_to_localize
/**

```

The Pozyx ready to localize tutorial (c) Pozyx Labs

Please read the tutorial that accompanies this sketch:

https://www.pozyx.io/Documentation/Tutorials/ready_to_localize/Arduino

This tutorial requires at least the contents of the Pozyx Ready to Localize kit. It demonstrates the positioning capabilities

of the Pozyx device both locally and remotely. Follow the steps to correctly set up your environment in the link, change the

parameters and upload this sketch. Watch the coordinates change as you move your device around!

```

*/
#include <Pozyx.h>
#include <Pozyx_definitions.h>
#include <Wire.h>

```

```

#include <ros.h>
#include <geometry_msgs/PoseWithCovarianceStamped.h>
#include <sensor_msgs/Imu.h>
#define TWO_PI 6.283185307179586476925286766559

////////////////////////////////////
////////////////// PARAMETERS //////////////////
////////////////////////////////////

uint16_t remote_id = 0x6e75;
// set this to the ID of the remote device
bool remote = false; // set this to true to use the remote ID

//boolean use_processing = false; // set this to true to output data for the processing sketch

const uint8_t num_anchors = 4; // the number of anchors
const uint16_t anchors[num_anchors] = {0x6e2c, 0x6e2a, 0x6e09, 0x6e04}; // the network id of
the anchors: change these to the network ids of your anchors.
//const int32_t anchors_x[num_anchors] = {0, 7260, 6000, 250}; // anchor x-coordinates in mm
//const int32_t anchors_y[num_anchors] = {-1600, -5300, 0, -5300}; // anchor y-coordinates in
mm
//const int32_t heights[num_anchors] = {0, 1470, 770, 0}; // anchor z-coordinates in mm
const int32_t anchors_x[num_anchors] = {0, 800, 7450, 12030};
const int32_t anchors_y[num_anchors] = {-140, -3050, -4460, -230};
const int32_t heights[num_anchors] = {0, 1170, 0, 0};

const uint8_t algorithm = POZYX_POS_ALG_TRACKING; // positioning algorithm to use. try
POZYX_POS_ALG_TRACKING for fast moving objects.
const uint8_t dimension = POZYX_3D; // positioning dimension
const int32_t height = 1000; // height of device, required in 2.5D positioning

ros::NodeHandle nh;

geometry_msgs::PoseWithCovarianceStamped pose_pub;
sensor_msgs::Imu imu_pub;

ros::Publisher pub1("Pozyx_pose", &pose_pub);
ros::Publisher pub2("Pozyx_imu", &imu_pub);

////////////////////////////////////

void setup(){
  Serial.begin(57600);

  if(Pozyx.begin(false, MODE_INTERRUPT, POZYX_INT_MASK_IMU) ==
  POZYX_FAILURE){
    abort();
  }
}

```

```
}

nh.initNode();
nh.advertise(pub1);
nh.advertise(pub2);

if(!remote){
remote_id = NULL;
}

// clear all previous devices in the device list
Pozyx.clearDevices(remote_id);
// sets the anchor manually
setAnchorsManual();
// sets the positioning algorithm
Pozyx.setPositionAlgorithm(algorithm, dimension, remote_id);

delay(100);
}

void loop(){
coordinates_t position;
quaternion_t quaternion;
pos_error_t posError;
angular_vel_t angVel;
linear_acceleration_t acceleration;

memset(&position, 0, sizeof(coordinates_t));
memset(&quaternion, 0, sizeof(quaternion_t));
memset(&posError, 0, sizeof(pos_error_t));
memset(&angVel, 0, sizeof(angVel));
memset(&acceleration, 0, sizeof(acceleration));

if(remote){
Pozyx.doRemotePositioning(remote_id, &position, dimension, height, algorithm);
Pozyx.getQuaternion(&quaternion, remote_id);
Pozyx.getPositionError(&posError, remote_id);
Pozyx.getAngularVelocity_dps(&angVel, remote_id);
Pozyx.getLinearAcceleration_mg(&acceleration, remote_id);

} else {
Pozyx.doPositioning(&position, dimension, height, algorithm);
Pozyx.getQuaternion(&quaternion);
Pozyx.getPositionError(&posError);
Pozyx.getAngularVelocity_dps(&angVel);
Pozyx.getLinearAcceleration_mg(&acceleration);
}
printPoseImu(position, quaternion, posError, angVel, acceleration);
```

```
}

// exports pose to ROS
void printPoseImu(coordinates_t coor, quaternion_t quat, pos_error_t err, angular_vel_t angVel,
linear_acceleration_t accel){
    // Set coordinates to msg
    pose_pub.pose.pose.position.x = ((float) coor.x) / 1000;
    pose_pub.pose.pose.position.y = ((float) coor.y) / 1000;
    pose_pub.pose.pose.position.z = ((float) coor.z) / 1000;
    // Set orientation to msg
    pose_pub.pose.pose.orientation.x = quat.x;
    pose_pub.pose.pose.orientation.y = quat.y;
    pose_pub.pose.pose.orientation.z = quat.z;
    pose_pub.pose.pose.orientation.w = quat.weight;
    // Set covariance to msg
    pose_pub.pose.covariance[0] = err.x;
    pose_pub.pose.covariance[7] = err.y;
    pose_pub.pose.covariance[14] = err.z;
    pose_pub.pose.covariance[21] = err.yz;
    pose_pub.pose.covariance[28] = err.xz;
    pose_pub.pose.covariance[35] = err.xy;

    pub1.publish(&pose_pub);

    imu_pub.orientation.x = quat.x;
    imu_pub.orientation.y = quat.y;
    imu_pub.orientation.z = quat.z;
    imu_pub.orientation.w = quat.weight;
    imu_pub.orientation_covariance[0] = 1.0;
    imu_pub.orientation_covariance[4] = 1.0;
    imu_pub.orientation_covariance[8] = 1.0;

    imu_pub.angular_velocity.x = angVel.x / 360 * TWO_PI;
    imu_pub.angular_velocity.y = angVel.y / 360 * TWO_PI;
    imu_pub.angular_velocity.z = angVel.z / 360 * TWO_PI;
    imu_pub.angular_velocity_covariance[0] = 1.0;
    imu_pub.angular_velocity_covariance[4] = 1.0;
    imu_pub.angular_velocity_covariance[8] = 1.0;

    imu_pub.linear_acceleration.x = ((float) accel.x) / 1000 * 9.8;
    imu_pub.linear_acceleration.y = ((float) accel.y) / 1000 * 9.8;
    imu_pub.linear_acceleration.z = ((float) accel.z) / 1000 * 9.8;
    imu_pub.linear_acceleration_covariance[0] = 1.0;
    imu_pub.linear_acceleration_covariance[4] = 1.0;
    imu_pub.linear_acceleration_covariance[8] = 1.0;

    pub2.publish(&imu_pub);
```

```

nh.spinOnce();

}

// function to manually set the anchor coordinates
void setAnchorsManual() {
  for(int i = 0; i < num_anchors; i++){
    device_coordinates_t anchor;
    anchor.network_id = anchors[i];
    anchor.flag = 0x1;
    anchor.pos.x = anchors_x[i];
    anchor.pos.y = anchors_y[i];
    anchor.pos.z = heights[i];
    Pozyx.addDevice(anchor, remote_id);
  }
  if (num_anchors > 4){
    Pozyx.setSelectionOfAnchors(POZYX_ANCHOR_SEL_AUTO, num_anchors, remote_id);
  }
}

```

Robot_Localization Launch File – Replace the path of the .yaml file with the appropriate parameter file.

```

<launch>
  <node pkg="robot_localization" type="ekf_localization_node" name="ekf_se"
  clear_params="true">
    <rosparam command="load" file="$(find robot_localization)/params/ekf_template.yaml" />

    <!-- Placeholder for output topic remapping
    <remap from="odometry/filtered" to="" />
    <remap from="accel/filtered" to="" />
    -->

  </node>
</launch>

```

Robot_Localization Parameter Files – Use ctrl+f with the file name to navigate quickly, as the files are relatively long.

- ekf_template.yaml files is an example from the robot_localization package. Refer to this for more detailed definitions of each parameter.
- lidar_imu.yaml is for fusing the LiDAR odometry from Rtabmap and the Pozyx imu.
- pozyx_turtlebot.yaml fuses wheel encoders, Pozyx UWB, and Pozyx IMU, but publishes map frame data. It achieves this using two robot_localization nodes; one performs map→odom transform, and the other performs odom→base_link transform.

- vo_imu.yaml fuses visual odometry from Rtabmap and the Pozyx IMU.
- vo_wheel_imu.yaml fuses visual odometry from Rtabmap, wheel encoders, and the Pozyx IMU.
- wheel_imu.yaml fuses the wheel encoders and the Pozyx IMU.
- wheel_pozyx.yaml fuses the wheel encoders, the Pozyx IMU, and the Pozyx UWB, and publishes odometry frame data.

Any parameter in the .yaml files that are commented out with a "#" are just set to the default value.

More information on parameter files can be found on http://docs.ros.org/melodic/api/robot_localization/html/state_estimation_nodes.html (valid as of Aug 10, 2018).

ekf_template.yaml

```
# The frequency, in Hz, at which the filter will output a position estimate. Note that the filter will
not begin
# computation until it receives at least one message from one of the inputs. It will then run
continuously at the
# frequency specified here, regardless of whether it receives more measurements. Defaults to 30 if
unspecified.
frequency: 30

# The period, in seconds, after which we consider a sensor to have timed out. In this event, we
carry out a predict
# cycle on the EKF without correcting it. This parameter can be thought of as the minimum
frequency with which the
# filter will generate new output. Defaults to 1 / frequency if not specified.
sensor_timeout: 0.1

# ekf_localization_node and ukf_localization_node both use a 3D omnidirectional motion model.
If this parameter is
# set to true, no 3D information will be used in your state estimate. Use this if you are operating in
a planar
# environment and want to ignore the effect of small variations in the ground plane that might
otherwise be detected
# by, for example, an IMU. Defaults to false if unspecified.
two_d_mode: false

# Use this parameter to provide an offset to the transform generated by ekf_localization_node.
This can be used for
# future dating the transform, which is required for interaction with some other packages. Defaults
to 0.0 if
# unspecified.
transform_time_offset: 0.0
```

```
# Use this parameter to provide specify how long the tf listener should wait for a transform to
become available.
# Defaults to 0.0 if unspecified.
transform_timeout: 0.0

# If you're having trouble, try setting this to true, and then echo the /diagnostics_agg topic to see if
the node is
# unhappy with any settings or data.
print_diagnostics: true

# Debug settings. Not for the faint of heart. Outputs a ludicrous amount of information to the file
specified by
# debug_out_file. I hope you like matrices! Please note that setting this to true will have strongly
deleterious
# effects on the performance of the node. Defaults to false if unspecified.
debug: false

# Defaults to "robot_localization_debug.txt" if unspecified. Please specify the full path.
debug_out_file: /path/to/debug/file.txt

# Whether to broadcast the transformation over the /tf topic. Defaults to true if unspecified.
publish_tf: true

# Whether to publish the acceleration state. Defaults to false if unspecified.
publish_acceleration: false

# REP-105 (http://www.ros.org/reps/rep-0105.html) specifies four principal coordinate frames:
base_link, odom, map, and
# earth. base_link is the coordinate frame that is affixed to the robot. Both odom and map are
world-fixed frames.
# The robot's position in the odom frame will drift over time, but is accurate in the short term and
should be
# continuous. The odom frame is therefore the best frame for executing local motion plans. The
map frame, like the odom
# frame, is a world-fixed coordinate frame, and while it contains the most globally accurate
position estimate for your
# robot, it is subject to discrete jumps, e.g., due to the fusion of GPS data or a correction from a
map-based
# localization node. The earth frame is used to relate multiple map frames by giving them a
common reference frame.
# ekf_localization_node and ukf_localization_node are not concerned with the earth frame.
# Here is how to use the following settings:
# 1. Set the map_frame, odom_frame, and base_link frames to the appropriate frame names for
your system.
# 1a. If your system does not have a map_frame, just remove it, and make sure "world_frame" is
set to the value of
# odom_frame.
# 2. If you are fusing continuous position data such as wheel encoder odometry, visual odometry,
or IMU data, set
```

```

# "world_frame" to your odom_frame value. This is the default behavior for robot_localization's
state estimation nodes.
# 3. If you are fusing global absolute position data that is subject to discrete jumps (e.g., GPS or
position updates
# from landmark observations) then:
# 3a. Set your "world_frame" to your map_frame value
# 3b. MAKE SURE something else is generating the odom->base_link transform. Note that this
can even be another state
# estimation node from robot_localization! However, that instance should *not* fuse the global
data.
map_frame: map # Defaults to "map" if unspecified
odom_frame: odom # Defaults to "odom" if unspecified
base_link_frame: base_link # Defaults to "base_link" if unspecified
world_frame: odom # Defaults to the value of odom_frame if unspecified

# The filter accepts an arbitrary number of inputs from each input message type
(nav_msgs/Odometry,
# geometry_msgs/PoseWithCovarianceStamped, geometry_msgs/TwistWithCovarianceStamped,
# sensor_msgs/Imu). To add an input, simply append the next number in the sequence to its "base"
name, e.g., odom0,
# odom1, twist0, twist1, imu0, imu1, imu2, etc. The value should be the topic name. These
parameters obviously have no
# default values, and must be specified.
odom0: example/odom

# Each sensor reading updates some or all of the filter's state. These options give you greater
control over which
# values from each measurement are fed to the filter. For example, if you have an odometry
message as input, but only
# want to use its Z position value, then set the entire vector to false, except for the third entry. The
order of the
# values is x, y, z, roll, pitch, yaw, vx, vy, vz, vroll, vpitch, vyaw, ax, ay, az. Note that not some
message types
# do not provide some of the state variables estimated by the filter. For example, a
TwistWithCovarianceStamped message
# has no pose information, so the first six values would be meaningless in that case. Each vector
defaults to all false
# if unspecified, effectively making this parameter required for each sensor.
odom0_config: [true, true, false,
false, false, false,
false, false, false,
false, false, true,
false, false, false]

# If you have high-frequency data or are running with a low frequency parameter value, then you
may want to increase
# the size of the subscription queue so that more measurements are fused.
odom0_queue_size: 2

```

```
# [ADVANCED] Large messages in ROS can exhibit strange behavior when they arrive at a high
frequency. This is a result
# of Nagle's algorithm. This option tells the ROS subscriber to use the tcpNoDelay option, which
disables Nagle's
# algorithm.
odom0_nodelay: false

# [ADVANCED] When measuring one pose variable with two sensors, a situation can arise in
which both sensors under-
# report their covariances. This can lead to the filter rapidly jumping back and forth between each
measurement as they
# arrive. In these cases, it often makes sense to (a) correct the measurement covariances, or (b) if
velocity is also
# measured by one of the sensors, let one sensor measure pose, and the other velocity. However,
doing (a) or (b) isn't
# always feasible, and so we expose the differential parameter. When differential mode is enabled,
all absolute pose
# data is converted to velocity data by differentiating the absolute pose measurements. These
velocities are then
# integrated as usual. NOTE: this only applies to sensors that provide pose measurements; setting
differential to true
# for twist measurements has no effect.
odom0_differential: false

# [ADVANCED] When the node starts, if this parameter is true, then the first measurement is
treated as a "zero point"
# for all future measurements. While you can achieve the same effect with the differential
parameter, the key
# difference is that the relative parameter doesn't cause the measurement to be converted to a
velocity before
# integrating it. If you simply want your measurements to start at 0 for a given sensor, set this to
true.
odom0_relative: false

# [ADVANCED] If your data is subject to outliers, use these threshold settings, expressed as
Mahalanobis distances, to
# control how far away from the current vehicle state a sensor measurement is permitted to be.
Each defaults to
# numeric_limits<double>::max() if unspecified. It is strongly recommended that these parameters
be removed if not
# required. Data is specified at the level of pose and twist variables, rather than for each variable in
isolation.
# For messages that have both pose and twist data, the parameter specifies to which part of the
message we are applying
# the thresholds.
odom0_pose_rejection_threshold: 5
odom0_twist_rejection_threshold: 1

# Further input parameter examples
```

```
odom1: example/another_odom
odom1_config: [false, false, true,
false, false, false,
false, false, false,
false, false, true,
false, false, false]
odom1_differential: false
odom1_relative: true
odom1_queue_size: 2
odom1_pose_rejection_threshold: 2
odom1_twist_rejection_threshold: 0.2
odom1_nodelay: false

pose0: example/pose
pose0_config: [true, true, false,
false, false, false,
false, false, false,
false, false, false,
false, false, false]
pose0_differential: true
pose0_relative: false
pose0_queue_size: 5
pose0_rejection_threshold: 2 # Note the difference in parameter name
pose0_nodelay: false

twist0: example/twist
twist0_config: [false, false, false,
false, false, false,
true, true, true,
false, false, false,
false, false, false]
twist0_queue_size: 3
twist0_rejection_threshold: 2
twist0_nodelay: false

imu0: example/imu
imu0_config: [false, false, false,
true, true, true,
false, false, false,
true, true, true,
true, true, true]
imu0_nodelay: false
imu0_differential: false
imu0_relative: true
imu0_queue_size: 5
imu0_pose_rejection_threshold: 0.8 # Note the difference in parameter names
imu0_twist_rejection_threshold: 0.8 #
imu0_linear_acceleration_rejection_threshold: 0.8 #
```

```

# [ADVANCED] Some IMUs automatically remove acceleration due to gravity, and others don't.
# If yours doesn't, please set
# this to true, and *make sure* your data conforms to REP-103, specifically, that the data is in
# ENU frame.
imu0_remove_gravitational_acceleration: true

# [ADVANCED] The EKF and UKF models follow a standard predict/correct cycle. During
# prediction, if there is no
# acceleration reference, the velocity at time t+1 is simply predicted to be the same as the velocity
# at time t. During
# correction, this predicted value is fused with the measured value to produce the new velocity
# estimate. This can be
# problematic, as the final velocity will effectively be a weighted average of the old velocity and
# the new one. When
# this velocity is integrated into a new pose, the result can be sluggish convergence. This effect
# is especially
# noticeable with LIDAR data during rotations. To get around it, users can try inflating the
# process_noise_covariance
# for the velocity variable in question, or decrease the variance of the variable in question in the
# measurement
# itself. In addition, users can also take advantage of the control command being issued to the
# robot at the time we
# make the prediction. If control is used, it will get converted into an acceleration term, which will
# be used during
# prediction. Note that if an acceleration measurement for the variable in question is available
# from one of the
# inputs, the control term will be ignored.
# Whether or not we use the control input during prediction. Defaults to false.
use_control: true
# Whether the input (assumed to be cmd_vel) is a geometry_msgs/Twist or
# geometry_msgs/TwistStamped message. Defaults to
# false.
stamped_control: false
# The last issued control command will be used in prediction for this period. Defaults to 0.2.
control_timeout: 0.2
# Which velocities are being controlled. Order is vx, vy, vz, vroll, vpitch, vyaw.
control_config: [true, false, false, false, false, true]
# Places limits on how large the acceleration term will be. Should match your robot's kinematics.
acceleration_limits: [1.3, 0.0, 0.0, 0.0, 0.0, 3.4]
# Acceleration and deceleration limits are not always the same for robots.
deceleration_limits: [1.3, 0.0, 0.0, 0.0, 0.0, 4.5]
# If your robot cannot instantaneously reach its acceleration limit, the permitted change can be
# controlled with these
# gains
acceleration_gains: [0.8, 0.0, 0.0, 0.0, 0.0, 0.9]
# If your robot cannot instantaneously reach its deceleration limit, the permitted change can be
# controlled with these
# gains
deceleration_gains: [1.0, 0.0, 0.0, 0.0, 0.0, 1.0]

```



```
0, 0, 0, 0, 0, 0, 0, 0, 0, 1e-9, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 1e-9, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1e-9, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1e-9, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1e-9, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1e-9, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1e-9]
```

lidar_imu.yaml

```
frequency: 20
sensor_timeout: 0.1
two_d_mode: false
transform_time_offset: 0.0
transform_timeout: 0.0
print_diagnostics: true
debug: false
debug_out_file: /path/to/debug/file.txt
publish_tf: true
publish_acceleration: false

map_frame: map # Defaults to "map" if unspecified
odom_frame: odom # Defaults to "odom" if unspecified
base_link_frame: base_link # Defaults to "base_link" if unspecified
world_frame: odom # Defaults to the value of odom_frame if unspecified

odom0: /scanmatch_odom
odom0_config: [true, true, false,
false, false, true,
false, false, false,
false, false, false,
false, false, false]

odom0_queue_size: 20
odom0_nodelay: false
odom0_differential: true
odom0_relative: true
#odom0_pose_rejection_thresd: 5
#odom0_twist_rejection_threshold: 1

imu0: Pozyx_imu
imu0_config: [false, false, false,
true, true, true,
false, false, false,
true, true, true,
true, true, true]
imu0_nodelay: false
imu0_differential: false
imu0_relative: true
imu0_queue_size: 20
#imu0_pose_rejection_threshold: 0.8 # Note the difference in parameter names
```

```
#imu0_twist_rejection_threshold: 0.8 #
#imu0_linear_acceleration_rejection_threshold: 0.8 #

# [ADVANCED] Some IMUs automatically remove acceleration due to gravity, and others don't.
# If yours doesn't, please set
# this to true, and *make sure* your data conforms to REP-103, specifically, that the data is in
# ENU frame.
imu0_remove_gravitational_acceleration: false

# [ADVANCED] The EKF and UKF models follow a standard predict/correct cycle. During
# prediction, if there is no
# acceleration reference, the velocity at time t+1 is simply predicted to be the same as the velocity
# at time t. During
# correction, this predicted value is fused with the measured value to produce the new velocity
# estimate. This can be
# problematic, as the final velocity will effectively be a weighted average of the old velocity and
# the new one. When
# this velocity is integrated into a new pose, the result can be sluggish convergence. This effect
# is especially
# noticeable with LIDAR data during rotations. To get around it, users can try inflating the
# process_noise_covariance
# for the velocity variable in question, or decrease the variance of the variable in question in the
# measurement
# itself. In addition, users can also take advantage of the control command being issued to the
# robot at the time we
# make the prediction. If control is used, it will get converted into an acceleration term, which will
# be used during
# prediction. Note that if an acceleration measurement for the variable in question is available
# from one of the
# inputs, the control term will be ignored.
# Whether or not we use the control input during prediction. Defaults to false.
use_control: true
# Whether the input (assumed to be cmd_vel) is a geometry_msgs/Twist or
# geometry_msgs/TwistStamped message. Defaults to
# false.
stamped_control: false
# The last issued control command will be used in prediction for this period. Defaults to 0.2.
control_timeout: 0.2
# Which velocities are being controlled. Order is vx, vy, vz, vroll, vpitch, vyaw.
control_config: [true, false, false, false, false, true]
# Places limits on how large the acceleration term will be. Should match your robot's kinematics.
acceleration_limits: [1.3, 0.0, 0.0, 0.0, 0.0, 3.4]
# Acceleration and deceleration limits are not always the same for robots.
deceleration_limits: [1.3, 0.0, 0.0, 0.0, 0.0, 4.5]
# If your robot cannot instantaneously reach its acceleration limit, the permitted change can be
# controlled with these
# gains
acceleration_gains: [0.8, 0.0, 0.0, 0.0, 0.0, 0.9]
```

```

# If your robot cannot instantaneously reach its deceleration limit, the permitted change can be
controlled with these
# gains
deceleration_gains: [1.0, 0.0, 0.0, 0.0, 0.0, 1.0]

# [ADVANCED] The process noise covariance matrix can be difficult to tune, and can vary for
each application, so it is
# exposed as a configuration parameter. This matrix represents the noise we add to the total error
after each
# prediction step. The better the omnidirectional motion model matches your system, the smaller
these values can be.
# However, if users find that a given variable is slow to converge, one approach is to increase the
# process_noise_covariance diagonal value for the variable in question, which will cause the
filter's predicted error
# to be larger, which will cause the filter to trust the incoming measurement more during
correction. The values are
# ordered as x, y, z, roll, pitch, yaw, vx, vy, vz, vroll, vpitch, vyaw, ax, ay, az. Defaults to the
matrix below if
# unspecified.
process_noise_covariance: [0.05, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0.05, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0.06, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0.03, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0.03, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0.06, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0.025, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0.025, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0.04, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0.01, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0.01, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0.02, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0.01, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0.01, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0.015]

# [ADVANCED] This represents the initial value for the state estimate error covariance matrix.
Setting a diagonal
# value (variance) to a large value will result in rapid convergence for initial measurements of the
variable in
# question. Users should take care not to use large values for variables that will not be measured
directly. The values
# are ordered as x, y, z, roll, pitch, yaw, vx, vy, vz, vroll, vpitch, vyaw, ax, ay, az. Defaults to the
matrix below
#if unspecified.
initial_estimate_covariance: [1e-9, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 1e-9, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 1e-9, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 1e-9, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 1e-9, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

```

```

0, 0, 0, 0, 0, 1e-9, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 1e-9, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 1e-9, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 1e-9, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 1e-9, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1e-9, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1e-9, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1e-9, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1e-9, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1e-9]

```

pozyx_turtlebot.yaml

For parameter descriptions, please refer to the template parameter files for each node.

```

ekf_se_odom:
  frequency: 20
  sensor_timeout: 0.1
  two_d_mode: false
  transform_time_offset: 0.0
  transform_timeout: 0.0
  print_diagnostics: true
  debug: false

map_frame: map
odom_frame: odom
base_link_frame: base_link
world_frame: odom

odom0: wheel_odom
odom0_config: [true, true, false,
false, false, true,
true, true, false,
false, false, true,
false, false, false]
odom0_queue_size: 5
odom0_nodelay: true
odom0_differential: false
odom0_relative: true

imu0: Pozyx_imu
imu0_config: [false, false, false,
true, true, true,
false, false, false,
true, true, true,
true, true, true]
imu0_nodelay: true
imu0_differential: false
imu0_relative: true

```

```

imu0_queue_size: 10
imu0_remove_gravitational_acceleration: false

use_control: false

process_noise_covariance: [1e-3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 1e-3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 1e-3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0.3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0.3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0.01, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0.5, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0.5, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0.1, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0.3, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0.3, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0.3, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0.3, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0.3, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0.3]

initial_estimate_covariance: [1e-9, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 1e-9, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 1e-9, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 1.0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 1.0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 1e-9, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 1.0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 1.0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 1.0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 1.0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1.0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1.0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1.0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1.0]

ekf_se_map:
frequency: 20
sensor_timeout: 0.1
two_d_mode: false
transform_time_offset: 0.0
transform_timeout: 0.0
print_diagnostics: true
debug: false

map_frame: map
odom_frame: odom
base_link_frame: base_link

```

world_frame: map

odom0: wheel_odom
odom0_config: [true, true, false,
false, false, true,
true, true, false,
false, false, true,
false, false, false]
odom0_queue_size: 5
odom0_nodelay: true
odom0_differential: false
odom0_relative: true

odom1: Pozyx_pose
odom1_config: [true, true, true,
false, false, false,
false, false, false,
false, false, false,
false, false, false]
odom1_queue_size: 10
odom1_nodelay: true
odom1_differential: false
odom1_relative: true

imu0: Pozyx_imu
imu0_config: [false, false, false,
true, true, true,
false, false, false,
true, true, true,
true, true, true]
imu0_nodelay: true
imu0_differential: false
imu0_relative: true
imu0_queue_size: 10
imu0_remove_gravitational_acceleration: false

use_control: false

process_noise_covariance: [1.0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 1.0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 1e-3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0.3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0.3, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0.01, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0.5, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0.5, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0.1, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0.3, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0.3, 0, 0, 0, 0,

```
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0.3, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0.3, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0.3, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0.3]
```

```
initial_estimate_covariance: [1.0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 1.0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 1e-9, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 1.0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 1.0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 1e-9, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 1.0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 1.0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 1.0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 1.0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1.0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1.0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1.0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1.0]
```

vo_imu.yaml

frequency: 20

sensor_timeout: 0.1

two_d_mode: false

transform_time_offset: 0.0

transform_timeout: 0.0

print_diagnostics: true

debug: false

debug_out_file: /path/to/debug/file.txt

publish_tf: true

publish_acceleration: false

map_frame: map # Defaults to "map" if unspecified

odom_frame: odom # Defaults to "odom" if unspecified

base_link_frame: base_link # Defaults to "base_link" if unspecified

world_frame: odom # Defaults to the value of odom_frame if unspecified

odom0: /rtabmap/odom

odom0_config: [true, true, true,

true, true, true,

false, false, false,

```

false, false, false,
false, false, false]
odom0_queue_size: 5
odom0_nodelay: false
odom0_differential: false
odom0_relative: true
#odom0_pose_rejection_threshold: 5
#odom0_twist_rejection_threshold: 1

imu0: Pozyx_imu
imu0_config: [false, false, false,
true, true, true,
false, false, false,
true, true, true,
true, true, true]
imu0_nodelay: false
imu0_differential: false
imu0_relative: true
imu0_queue_size: 5
#imu0_pose_rejection_threshold: 0.8 # Note the difference in parameter names
#imu0_twist_rejection_threshold: 0.8 #
#imu0_linear_acceleration_rejection_threshold: 0.8 #

imu0_remove_gravitational_acceleration: false

# [ADVANCED] The EKF and UKF models follow a standard predict/correct cycle. During
prediction, if there is no
# acceleration reference, the velocity at time t+1 is simply predicted to be the same as the velocity
at time t. During
# correction, this predicted value is fused with the measured value to produce the new velocity
estimate. This can be
# problematic, as the final velocity will effectively be a weighted average of the old velocity and
the new one. When
# this velocity is integrated into a new pose, the result can be sluggish convergence. This effect
is especially
# noticeable with LIDAR data during rotations. To get around it, users can try inflating the
process_noise_covariance
# for the velocity variable in question, or decrease the variance of the variable in question in the
measurement
# itself. In addition, users can also take advantage of the control command being issued to the
robot at the time we
# make the prediction. If control is used, it will get converted into an acceleration term, which will
be used during
# prediction. Note that if an acceleration measurement for the variable in question is available
from one of the
# inputs, the control term will be ignored.
# Whether or not we use the control input during prediction. Defaults to false.
use_control: true

```

```

# Whether the input (assumed to be cmd_vel) is a geometry_msgs/Twist or
geometry_msgs/TwistStamped message. Defaults to
# false.
stamped_control: false
# The last issued control command will be used in prediction for this period. Defaults to 0.2.
control_timeout: 0.2
# Which velocities are being controlled. Order is vx, vy, vz, vroll, vpitch, vyaw.
control_config: [true, false, false, false, false, true]
# Places limits on how large the acceleration term will be. Should match your robot's kinematics.
acceleration_limits: [1.3, 0.0, 0.0, 0.0, 0.0, 3.4]
# Acceleration and deceleration limits are not always the same for robots.
deceleration_limits: [1.3, 0.0, 0.0, 0.0, 0.0, 4.5]
# If your robot cannot instantaneously reach its acceleration limit, the permitted change can be
controlled with these
# gains
acceleration_gains: [0.8, 0.0, 0.0, 0.0, 0.0, 0.9]
# If your robot cannot instantaneously reach its deceleration limit, the permitted change can be
controlled with these
# gains
deceleration_gains: [1.0, 0.0, 0.0, 0.0, 0.0, 1.0]

# [ADVANCED] The process noise covariance matrix can be difficult to tune, and can vary for
each application, so it is
# exposed as a configuration parameter. This matrix represents the noise we add to the total error
after each
# prediction step. The better the omnidirectional motion model matches your system, the smaller
these values can be.
# However, if users find that a given variable is slow to converge, one approach is to increase the
# process_noise_covariance diagonal value for the variable in question, which will cause the
filter's predicted error
# to be larger, which will cause the filter to trust the incoming measurement more during
correction. The values are
# ordered as x, y, z, roll, pitch, yaw, vx, vy, vz, vroll, vpitch, vyaw, ax, ay, az. Defaults to the
matrix below if
# unspecified.
process_noise_covariance: [0.05, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0.05, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0.06, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0.03, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0.03, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0.06, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0.025, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0.025, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0.04, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0.01, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0.01, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0.02, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0.01, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0.01, 0,

```

```
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0.015]
```

```
# [ADVANCED] This represents the initial value for the state estimate error covariance matrix.
Setting a diagonal
# value (variance) to a large value will result in rapid convergence for initial measurements of the
variable in
# question. Users should take care not to use large values for variables that will not be measured
directly. The values
# are ordered as x, y, z, roll, pitch, yaw, vx, vy, vz, vroll, vpitch, vyaw, ax, ay, az. Defaults to the
matrix below
#if unspecified.
initial_estimate_covariance: [1e-9, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 1e-9, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 1e-9, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 1e-9, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 1e-9, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 1e-9, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 1e-9, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 1e-9, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 1e-9, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 1e-9, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1e-9, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1e-9, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1e-9, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1e-9, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1e-9]
```

vo_wheel_imu.yaml

```
frequency: 20
```

```
sensor_timeout: 0.1
```

```
two_d_mode: false
```

```
transform_time_offset: 0.0
```

```
transform_timeout: 0.0
```

```
print_diagnostics: true
```

```
debug: false
```

```
debug_out_file: /path/to/debug/file.txt
```

```
publish_tf: true
```

```
publish_acceleration: false
```

```
map_frame: map # Defaults to "map" if unspecified
```

```
odom_frame: odom # Defaults to "odom" if unspecified
```

```
base_link_frame: base_link # Defaults to "base_link" if unspecified
world_frame: odom # Defaults to the value of odom_frame if unspecified
```

```
odom0: wheel_odom
odom0_config: [true, true, false,
               false, false, true,
               true, true, false,
               false, false, false,
               false, false, false]
```

```
odom0_queue_size: 4
odom0_nodelay: false
odom0_differential: false
odom0_relative: true
#odom0_pose_rejection_threshold: 5
#odom0_twist_rejection_threshold: 1
```

```
odom1: /rtabmap/odom
odom1_config: [true, true, true,
               true, true, true,
               false, false, false,
               false, false, false,
               false, false, false]
```

```
odom1_queue_size: 5
odom1_nodelay: false
odom1_differential: true
odom1_relative: true
#odom1_pose_rejection_threshold: 5
#odom1_twist_rejection_threshold: 1
```

```
imu0: Pozyx_imu
imu0_config: [false, false, false,
              true, true, true,
              false, false, false,
              true, true, true,
              true, true, true]
imu0_nodelay: false
imu0_differential: false
imu0_relative: true
imu0_queue_size: 5
#imu0_pose_rejection_threshold: 0.8 # Note the difference in parameter names
#imu0_twist_rejection_threshold: 0.8 #
#imu0_linear_acceleration_rejection_threshold: 0.8 #
```

```
# [ADVANCED] Some IMUs automatically remove acceleration due to gravity, and others don't.
If yours doesn't, please set
```

```
# this to true, and *make sure* your data conforms to REP-103, specifically, that the data is in
ENU frame.
imu0_remove_gravitational_acceleration: false

# [ADVANCED] The EKF and UKF models follow a standard predict/correct cycle. During
prediction, if there is no
# acceleration reference, the velocity at time t+1 is simply predicted to be the same as the velocity
at time t. During
# correction, this predicted value is fused with the measured value to produce the new velocity
estimate. This can be
# problematic, as the final velocity will effectively be a weighted average of the old velocity and
the new one. When
# this velocity is integrated into a new pose, the result can be sluggish convergence. This effect
is especially
# noticeable with LIDAR data during rotations. To get around it, users can try inflating the
process_noise_covariance
# for the velocity variable in question, or decrease the variance of the variable in question in the
measurement
# itself. In addition, users can also take advantage of the control command being issued to the
robot at the time we
# make the prediction. If control is used, it will get converted into an acceleration term, which will
be used during
# prediction. Note that if an acceleration measurement for the variable in question is available
from one of the
# inputs, the control term will be ignored.
# Whether or not we use the control input during prediction. Defaults to false.
use_control: true
# Whether the input (assumed to be cmd_vel) is a geometry_msgs/Twist or
geometry_msgs/TwistStamped message. Defaults to
# false.
stamped_control: false
# The last issued control command will be used in prediction for this period. Defaults to 0.2.
control_timeout: 0.2
# Which velocities are being controlled. Order is vx, vy, vz, vroll, vpitch, vyaw.
control_config: [true, false, false, false, false, true]
# Places limits on how large the acceleration term will be. Should match your robot's kinematics.
acceleration_limits: [1.3, 0.0, 0.0, 0.0, 0.0, 3.4]
# Acceleration and deceleration limits are not always the same for robots.
deceleration_limits: [1.3, 0.0, 0.0, 0.0, 0.0, 4.5]
# If your robot cannot instantaneously reach its acceleration limit, the permitted change can be
controlled with these
# gains
acceleration_gains: [0.8, 0.0, 0.0, 0.0, 0.0, 0.9]
# If your robot cannot instantaneously reach its deceleration limit, the permitted change can be
controlled with these
# gains
deceleration_gains: [1.0, 0.0, 0.0, 0.0, 0.0, 1.0]
```

[ADVANCED] The process noise covariance matrix can be difficult to tune, and can vary for each application, so it is
 # exposed as a configuration parameter. This matrix represents the noise we add to the total error after each
 # prediction step. The better the omnidirectional motion model matches your system, the smaller these values can be.
 # However, if users find that a given variable is slow to converge, one approach is to increase the
 # process_noise_covariance diagonal value for the variable in question, which will cause the filter's predicted error
 # to be larger, which will cause the filter to trust the incoming measurement more during correction. The values are
 # ordered as x, y, z, roll, pitch, yaw, vx, vy, vz, vroll, vpitch, vyaw, ax, ay, az. Defaults to the matrix below if
 # unspecified.

```
process_noise_covariance: [0.05, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0.05, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0.06, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0.03, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0.03, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0.06, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0.025, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0.025, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0.04, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0.01, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0.01, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0.02, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0.01, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0.01, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0.015]
```

[ADVANCED] This represents the initial value for the state estimate error covariance matrix. Setting a diagonal
 # value (variance) to a large value will result in rapid convergence for initial measurements of the variable in
 # question. Users should take care not to use large values for variables that will not be measured directly. The values
 # are ordered as x, y, z, roll, pitch, yaw, vx, vy, vz, vroll, vpitch, vyaw, ax, ay, az. Defaults to the matrix below
 # if unspecified.

```
initial_estimate_covariance: [1e-9, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 1e-9, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 1e-9, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 1e-9, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 1e-9, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 1e-9, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 1e-9, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 1e-9, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 1e-9, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 1e-9, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1e-9, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1e-9, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1e-9, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1e-9, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1e-9, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1e-9,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1e-9,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1e-9,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1e-9,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1e-9]
```

```
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1e-9, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1e-9, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1e-9, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1e-9, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1e-9]
```

wheel_imu.yaml

```
frequency: 20
```

```
sensor_timeout: 0.1
```

```
two_d_mode: false
```

```
transform_time_offset: 0.0
```

```
transform_timeout: 0.0
```

```
print_diagnostics: true
```

```
debug: false
```

```
debug_out_file: /path/to/debug/file.txt
```

```
publish_tf: true
```

```
publish_acceleration: false
```

```
map_frame: map # Defaults to "map" if unspecified
```

```
odom_frame: odom # Defaults to "odom" if unspecified
```

```
base_link_frame: base_link # Defaults to "base_link" if unspecified
```

```
world_frame: odom # Defaults to the value of odom_frame if unspecified
```

```
odom0: wheel_odom
```

```
odom0_config: [true, true, false,
```

```
  false, false, true,
```

```
  true, true, false,
```

```
  false, false, false,
```

```
  false, false, false]
```

```
odom0_queue_size: 4
```

```
odom0_nodelay: false
```

```
odom0_differential: false
```

```
odom0_relative: true
```

```
imu0: Pozyx_imu
```

```
imu0_config: [false, false, false,
```

```
  true, true, true,
```

```
  false, false, false,
```

```
  true, true, true,
```

```
  true, true, true]
```

```
imu0_nodelay: false
```

```

imu0_differential: false
imu0_relative: true
imu0_queue_size: 5
#imu0_pose_rejection_threshold: 0.8 # Note the difference in parameter names
#imu0_twist_rejection_threshold: 0.8 #
#imu0_linear_acceleration_rejection_threshold: 0.8 #

# [ADVANCED] Some IMUs automatically remove acceleration due to gravity, and others don't.
# If yours doesn't, please set
# this to true, and *make sure* your data conforms to REP-103, specifically, that the data is in
# ENU frame.
imu0_remove_gravitational_acceleration: false

# [ADVANCED] The EKF and UKF models follow a standard predict/correct cycle. During
# prediction, if there is no
# acceleration reference, the velocity at time t+1 is simply predicted to be the same as the velocity
# at time t. During
# correction, this predicted value is fused with the measured value to produce the new velocity
# estimate. This can be
# problematic, as the final velocity will effectively be a weighted average of the old velocity and
# the new one. When
# this velocity is integrated into a new pose, the result can be sluggish convergence. This effect
# is especially
# noticeable with LIDAR data during rotations. To get around it, users can try inflating the
# process_noise_covariance
# for the velocity variable in question, or decrease the variance of the variable in question in the
# measurement
# itself. In addition, users can also take advantage of the control command being issued to the
# robot at the time we
# make the prediction. If control is used, it will get converted into an acceleration term, which will
# be used during
# prediction. Note that if an acceleration measurement for the variable in question is available
# from one of the
# inputs, the control term will be ignored.
# Whether or not we use the control input during prediction. Defaults to false.
use_control: true
# Whether the input (assumed to be cmd_vel) is a geometry_msgs/Twist or
# geometry_msgs/TwistStamped message. Defaults to
# false.
stamped_control: false
# The last issued control command will be used in prediction for this period. Defaults to 0.2.
control_timeout: 0.2
# Which velocities are being controlled. Order is vx, vy, vz, vroll, vpitch, vyaw.
control_config: [true, false, false, false, false, true]
# Places limits on how large the acceleration term will be. Should match your robot's kinematics.
acceleration_limits: [1.3, 0.0, 0.0, 0.0, 0.0, 3.4]
# Acceleration and deceleration limits are not always the same for robots.
deceleration_limits: [1.3, 0.0, 0.0, 0.0, 0.0, 4.5]

```

```

# If your robot cannot instantaneously reach its acceleration limit, the permitted change can be
controlled with these
# gains
acceleration_gains: [0.8, 0.0, 0.0, 0.0, 0.0, 0.9]
# If your robot cannot instantaneously reach its deceleration limit, the permitted change can be
controlled with these
# gains
deceleration_gains: [1.0, 0.0, 0.0, 0.0, 0.0, 1.0]

# [ADVANCED] The process noise covariance matrix can be difficult to tune, and can vary for
each application, so it is
# exposed as a configuration parameter. This matrix represents the noise we add to the total error
after each
# prediction step. The better the omnidirectional motion model matches your system, the smaller
these values can be.
# However, if users find that a given variable is slow to converge, one approach is to increase the
# process_noise_covariance diagonal value for the variable in question, which will cause the
filter's predicted error
# to be larger, which will cause the filter to trust the incoming measurement more during
correction. The values are
# ordered as x, y, z, roll, pitch, yaw, vx, vy, vz, vroll, vpitch, vyaw, ax, ay, az. Defaults to the
matrix below if
# unspecified.
process_noise_covariance: [0.05, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0.05, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0.06, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0.03, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0.03, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0.06, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0.025, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0.025, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0.04, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0.01, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0.01, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0.02, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0.01, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0.01, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0.015]

# [ADVANCED] This represents the initial value for the state estimate error covariance matrix.
Setting a diagonal
# value (variance) to a large value will result in rapid convergence for initial measurements of the
variable in
# question. Users should take care not to use large values for variables that will not be measured
directly. The values
# are ordered as x, y, z, roll, pitch, yaw, vx, vy, vz, vroll, vpitch, vyaw, ax, ay, az. Defaults to the
matrix below
# if unspecified.
initial_estimate_covariance: [1e-9, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 1e-9, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 1e-9, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 1e-9, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 1e-9, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 1e-9, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 1e-9, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 1e-9, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 1e-9, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 1e-9, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1e-9, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1e-9, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1e-9, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1e-9, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1e-9]

```

```

0, 1e-9, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 1e-9, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 1e-9, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 1e-9, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 1e-9, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 1e-9, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 1e-9, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 1e-9, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 1e-9, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1e-9, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1e-9, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1e-9, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1e-9,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1e-9]

```

wheel_pozyx.yaml

```

frequency: 20
sensor_timeout: 0.1
two_d_mode: false
transform_time_offset: 0.0
transform_timeout: 0.0
print_diagnostics: true
debug: false
debug_out_file: /path/to/debug/file.txt
publish_tf: true

publish_acceleration: false

map_frame: map # Defaults to "map" if unspecified
odom_frame: odom # Defaults to "odom" if unspecified
base_link_frame: base_link # Defaults to "base_link" if unspecified
world_frame: odom # Defaults to the value of odom_frame if unspecified

odom0: wheel_odom
odom0_config: [true, true, false,
false, false, true,
true, true, false,
false, false, false,
false, false, false]

odom0_queue_size: 4

odom0_nodelay: false
odom0_differential: false
odom0_relative: true

pose0: Pozyx_pose
pose0_config: [true, true, true,
false, false, false,

```

```

false, false, false,
false, false, false,
false, false, false]
pose0_differential: true
pose0_relative: true
pose0_queue_size: 5
pose0_rejection_threshold: 2 # Note the difference in parameter name
pose0_nodelay: false

imu0: Pozyx_imu
imu0_config: [false, false, false,
true, true, true,
false, false, false,
true, true, true,
true, true, true]
imu0_nodelay: false
imu0_differential: false
imu0_relative: true
imu0_queue_size: 5
#imu0_pose_rejection_threshold: 0.8 # Note the difference in parameter names
#imu0_twist_rejection_threshold: 0.8 #
#imu0_linear_acceleration_rejection_threshold: 0.8 #
imu0_remove_gravitational_acceleration: false

# [ADVANCED] The EKF and UKF models follow a standard predict/correct cycle. During
prediction, if there is no
# acceleration reference, the velocity at time t+1 is simply predicted to be the same as the velocity
at time t. During
# correction, this predicted value is fused with the measured value to produce the new velocity
estimate. This can be
# problematic, as the final velocity will effectively be a weighted average of the old velocity and
the new one. When
# this velocity is integrated into a new pose, the result can be sluggish convergence. This effect
is especially
# noticeable with LIDAR data during rotations. To get around it, users can try inflating the
process_noise_covariance
# for the velocity variable in question, or decrease the variance of the variable in question in the
measurement
# itself. In addition, users can also take advantage of the control command being issued to the
robot at the time we
# make the prediction. If control is used, it will get converted into an acceleration term, which will
be used during
# prediction. Note that if an acceleration measurement for the variable in question is available
from one of the
# inputs, the control term will be ignored.
# Whether or not we use the control input during prediction. Defaults to false.
use_control: true
# Whether the input (assumed to be cmd_vel) is a geometry_msgs/Twist or
geometry_msgs/TwistStamped message. Defaults to

```

```

# false.
stamped_control: false
# The last issued control command will be used in prediction for this period. Defaults to 0.2.
control_timeout: 0.2
# Which velocities are being controlled. Order is vx, vy, vz, vroll, vpitch, vyaw.
control_config: [true, false, false, false, false, true]
# Places limits on how large the acceleration term will be. Should match your robot's kinematics.
acceleration_limits: [1.3, 0.0, 0.0, 0.0, 0.0, 3.4]
# Acceleration and deceleration limits are not always the same for robots.
deceleration_limits: [1.3, 0.0, 0.0, 0.0, 0.0, 4.5]
# If your robot cannot instantaneously reach its acceleration limit, the permitted change can be
controlled with these
# gains
acceleration_gains: [0.8, 0.0, 0.0, 0.0, 0.0, 0.9]
# If your robot cannot instantaneously reach its deceleration limit, the permitted change can be
controlled with these
# gains
deceleration_gains: [1.0, 0.0, 0.0, 0.0, 0.0, 1.0]

# [ADVANCED] The process noise covariance matrix can be difficult to tune, and can vary for
each application, so it is
# exposed as a configuration parameter. This matrix represents the noise we add to the total error
after each
# prediction step. The better the omnidirectional motion model matches your system, the smaller
these values can be.
# However, if users find that a given variable is slow to converge, one approach is to increase the
# process_noise_covariance diagonal value for the variable in question, which will cause the
filter's predicted error
# to be larger, which will cause the filter to trust the incoming measurement more during
correction. The values are
# ordered as x, y, z, roll, pitch, yaw, vx, vy, vz, vroll, vpitch, vyaw, ax, ay, az. Defaults to the
matrix below if
# unspecified.
process_noise_covariance: [0.05, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0.05, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0.06, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0.03, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0.03, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0.06, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0.025, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0.025, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0.04, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0.01, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0.01, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0.02, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0.01, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0.01, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0.015]

```

[ADVANCED] This represents the initial value for the state estimate error covariance matrix.
 Setting a diagonal
 # value (variance) to a large value will result in rapid convergence for initial measurements of the
 variable in
 # question. Users should take care not to use large values for variables that will not be measured
 directly. The values
 # are ordered as x, y, z, roll, pitch, yaw, vx, vy, vz, vroll, vpitch, vyaw, ax, ay, az. Defaults to the
 matrix below
 #if unspecified.
 initial_estimate_covariance: [1e-9, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 0, 1e-9, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 1e-9, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 1e-9, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 0, 1e-9, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 1e-9, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 1e-9, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 1e-9, 0, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 0, 1e-9, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 0, 0, 1e-9, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1e-9, 0, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1e-9, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1e-9, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1e-9]

Appendix F: Automation

Installation

This installation guide assumes that:

1. ROS Kinetic is installed.
2. A ROS workspace is properly configured.
3. RTAB-Map (and dependencies) are installed and are properly running.
4. Rviz is installed.
5. The robot publishes a topic ‘/cmd_vel’ that receives ‘geometry_msgs/Twist’ messages and is properly configured.

If all of these prerequisites are met, then installation can proceed. To install the necessary packages and dependencies, follow the proceeding steps, running commands exactly as they are written:

1. Installing Navigation Stack:

- a) `$ sudo apt install ros-kinetic-navigation`
- b) `$ cd ~/catkin_ws/src`
- c) `$ git clone --branch kinetic-devel https://github.com/ros-planning/navigation.git`
- d) `$ cd .. && catkin_make`

Navigation Stack should be installed properly, check this by running: `$ roscd move_base`

2. Installing explore_lite (and multi-robot-map-merge):

- a) `$ sudo apt install ros-kinetic-multirobot-map-merge ros-kinetic-explore-lite`
- b) `$ cd ~/catkin_ws/src`
- c) `$ git clone --branch kinetic-devel https://github.com/hrnr/m-explore.git`
- d) `$ cd .. && catkin_make`

Explore_lite should be installed properly, check this by running: `$ roscd explore_lite`

3. Installing frontier_exploration:

- a) `$ sudo apt install ros-kinetic-frontier-exploration`
- b) `$ cd ~/catkin_ws/src`
- c) `$ git clone --branch indigo-devel https://github.com/paulbovbel/frontier_exploration.git`
- d) `$ cd .. && catkin_make`

Frontier_exploration should be installed properly, check this by running: `$ roscd frontier_exploration`. Note that either frontier_exploration or explore_lite can be used, not both. Explore_lite is the favored package, but frontier_exploration may be experimented with, so it is advantageous to install.

4. Configuring move_base launch file:
 - a) `$ roscd move_base`
 - b) `$ mkdir launch/ && cd launch`
 - c) `$ nano move_base_rtabmap.launch`
 - d) Copy and paste move_base_rtabmap.launch and save (Appendix B)
 - e) Test to make sure that the file launches by running: `$ roslaunch move_base_rtabmap.launch`
 - The file should launch but return a warning since nothing else is running.
5. Configuring explore_lite launch file:
 - a) `$ roscd explore_lite/launch`
 - b) `$ nano explore_lite_rtabmap.launch`
 - c) Copy and paste explore_lite_rtabmap.launch and save (Appendix C)
 - d) Note: this file should be nearly identical to explore_costmap.launch in the same directory.
6. Configuring frontier_exploration launch file:
 - a) `$ roscd frontier_exploration/launch`
 - b) `$ nano frontier_exploration.launch`
 - c) Copy and paste frontier_exploration.launch and save (Appendix D)
 - d) Note: this file is similar to global_map.launch in frontier_exploration/launch
7. Gmapping applicable launch files for move_base and explore_lite:
 - a) If rtabmap is not working properly, but you still want autonomous exploration, these launch files have been proven to work with gmapping.

Execution

The execution order is as follows:

1. Turn on GVRbot and NUC
 - a) Verify that the NUC is connected to GVRbot WiFi
 - b) Verify that the GVRbot is broadcasting its odometry topic
 - c) Verify that the GVRbot's time is correct (`$ sudo hwclock - - show`)
2. Launch all sensors
 - a) This would include:
 - d435
 - t265
 - 3D LiDAR
3. Launch transforms file
 - a) `$ roslaunch ~/<keybaseRepo>/Transforms/transforms_ouster.launch`

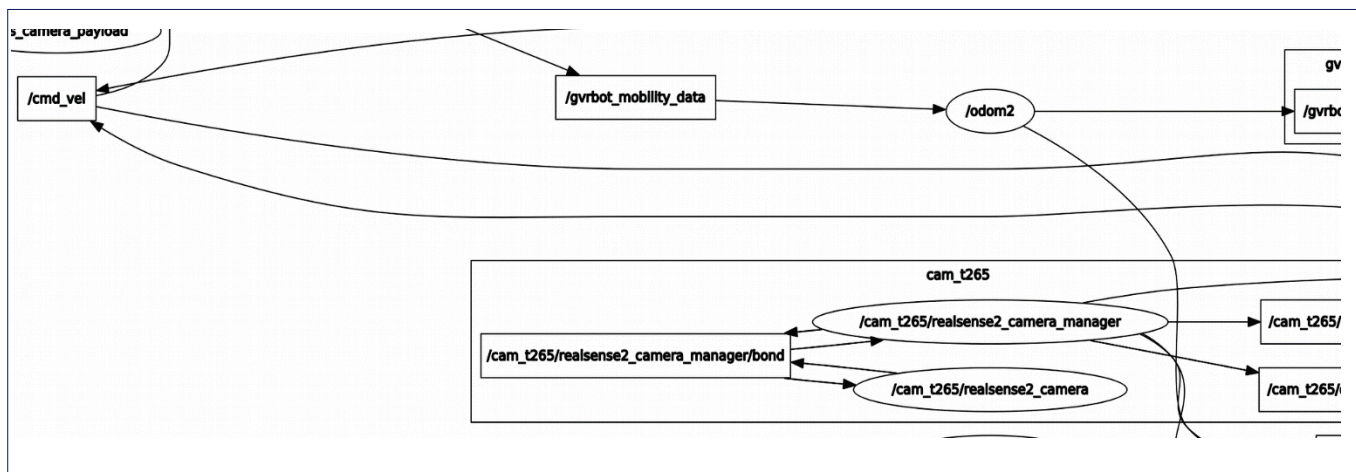
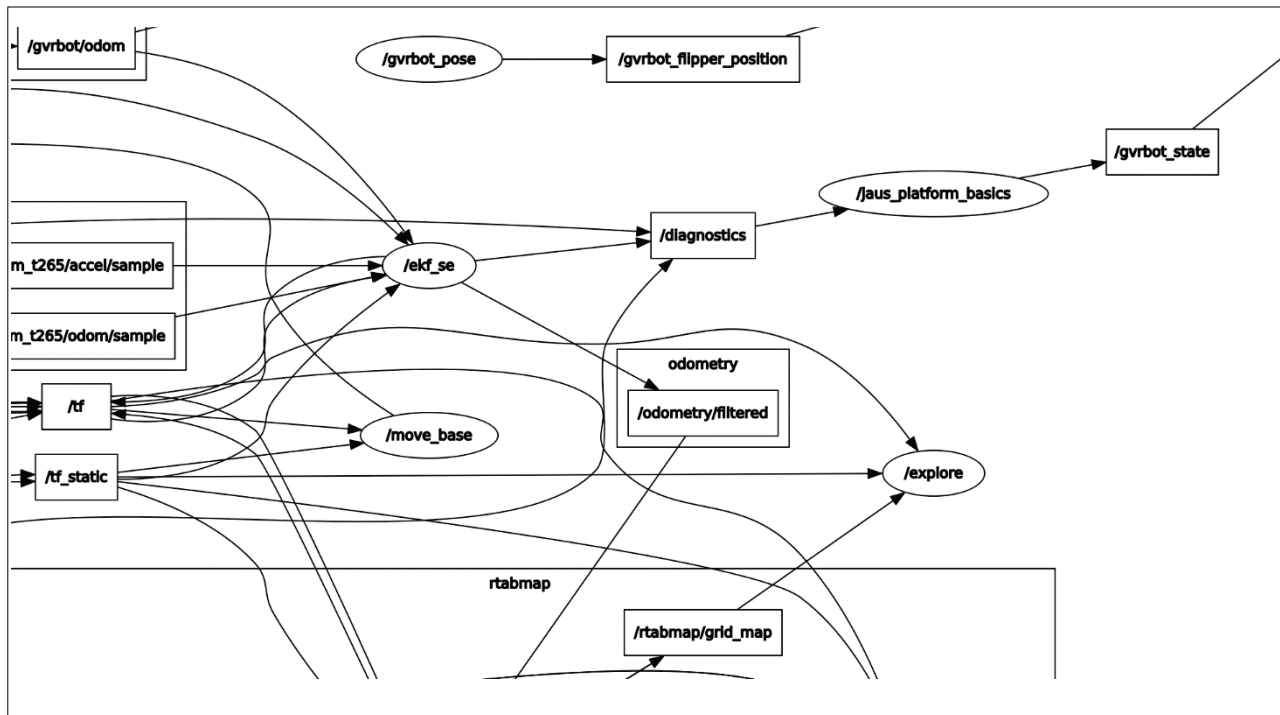
4. Launch robot_localization for sensor fusion
 - a) `$ roslaunch robot_localization ekf_gvrbot265.launch`
5. Launch rtabmap
 - a) `$ roslaunch rtabmap_ros turtlebot2_rtabmap.launch`
6. Launch move_base
 - a) `$ roslaunch ~/<keybaseRepo>/auto-explore/move_base_allconf.launch`
7. If using frontier_exploration:
 - a) `$ roslaunch ~/<keybaseRepo>/auto-explore/frontier_exploration.launch`
8. If using explore_lite:
 - a) `$ roslaunch ~/<keybaseRepo>/auto-explore/explore_lite_rtabmap.launch`

Note: If you are using gmapping instead of rtabmap, stop at step 4 and resume at the following:

9. Verify that gmapping is installed:
 - a) `$ rospack list` (And look for “gmapping” and “slam_gmapping”)
10. Then launch move_base and gmapping:
 - b) `$ roslaunch ~/<keybaseRepo>/auto-explore/move_base_gmapping.launch`
11. If you are using frontier_exploration:
 - c) `$ roslaunch ~/<keybaseRepo>/auto-explore/frontier_exploration.launch`
12. If you are using explore_lite:
 - d) `$ roslaunch ~/<keybaseRepo>/auto-explore/explore_lite_gmapping.launch`

Note: If you are using rviz, you can use Region_exploration.rviz for visualization.

A) Rqt_graph of all operations running with explore_lite for exploration. The left side of Figure 6 is connected to the right side of Figure 7. Although incomplete, this graph shows the connection between cmd_vel, move_base, rtabmap, and explore_lite.



B) move_base_rtabmap.launch (be careful that all ‘spaces’ are indeed spaces):

```
<launch>
<arg name="launch_prefix" default=""/>
<arg name="explore" default="true"/>
<arg name="output" default="screen"/>
```

```
<!-- The 2DOccupancyGrid should be produced by rtabmap/proj_map, or
/grid_map, or
/grid_prob_map, in that order, in accordance with desirability -->
```

```

<node pkg="move_base" type="move_base" name="move_base"
output="screen">

  <param name="controller_frequency" value="20.0"/>
  <param name="planner_patience" value="3.0"/>
  <param name="controller_patience" value="7.5"/>
  <rosparam> local_costmap:
    update_frequency: 5
    publish_frequency: 5
    transform_tolerance: 0.25 static_map: false rolling_window: true width: 5
    height: 5
    origin_x: 0
    origin_y: 0
    resolution: 0.08
    inflation_radius: 0.1
    obstacle_range: 4.0
    raytrace_range: 5.0
    footprint: [[-0.2667, 0.0762], [-0.2667, -0.5334], [0.2667, -0.5334],
    [0.2667, 0.0762]]
    map_topic: /rtabmap/grid_map subscribe_to_updates: true
  observation_sources: laser_scan_sensor
    laser_scan_sensor: {sensor_frame: laser, data_type: LaserScan, topic:
scan, marking: true, clearing: true}
    global_frame: map robot_base_frame: base_link
  always_send_full_costmap: true
</rosparam>

  <rosparam> global_costmap:
    update_frequency: 5
    publish_frequency: 5
    transform_tolerance: 0.5
    width: 20
    height: 20
    origin_x: 0
    origin_y: 0 static_map: false rolling_window: true inflation_radius: 0.1
    resolution: 0.08
    obstacle_range: 4.0
    raytrace_range: 5.0
    footprint: [[-0.2667, 0.0762], [-0.2667, -0.5334], [0.2667, -0.5334],
    [0.2667, 0.0762]]
    map_topic: /rtabmap/grid_map subscribe_to_updates: true
  observation_sources: laser_scan_sensor
    laser_scan_sensor: {sensor_frame: laser, data_type: LaserScan, topic:
scan, marking: true, clearing: true}
    global_frame: map robot_base_frame: base_link

```

```

always_send_full_costmap: true
</rosparam>

    <rosparam> TrajectoryPlannerROS:
    max_vel_x: 0.2
    min_vel_x: -0.2
    max_vel_theta: 0.35
    min_vel_theta: -0.35
    min_in_place_vel_theta: 0.25
    acc_lim_theta: 0.25
    acc_lim_x: 0.5
    acc_lim_Y: 0.5 holonomic_robot: false meter_scoring: true escape_vel: -0.15
    xy_goal_tolerance: 0.10
    yaw_goal_tolerance: 0.25
    </rosparam>

</node>

</launch>

C) explore_lite_rtabmap.launch:

<launch>
<node pkg="explore_lite" type="explore" respawn="false" name="explore"
output="screen">
    <param name="robot_base_frame" value="base_link"/>

    <!-- This version works with rtabmap, which publishes a map to move_base -->
    <param name="costmap_topic"
value="move_base/global_costmap/costmap"/>
    <param name="costmap_updates_topic"
value="move_base/global_costmap/costmap_updates"/>
    <!-- The previous topic may not exist, which is OK -->
    <!-- The following parameters have not been played with, but work well on the
turtlebot-->

    <param name="visualize" value="true"/>
    <param name="planner_frequency" value="0.33"/>
    <param name="progress_timeout" value="30.0"/>
    <param name="potential_scale" value="3.0"/>
    <param name="orientation_scale" value="0.0"/>
    <param name="gain_scale" value="1.0"/>
    <param name="transform_tolerance" value="0.3"/>
    <param name="min_frontier_size" value="0.75"/>

</node>

```

```
</launch>
```

D) frontier_exploration.launch:

```
<launch>
```

```
<!-- Set to your sensor's range -->
```

```
<arg name="sensor_range" default="1.0"/>
```

```
<node pkg="frontier_exploration" type="explore_client"
name="explore_client" output="screen"/>
```

```
<node pkg="frontier_exploration" type="explore_server"
name="explore_server" output="screen" >
```

```
<param name="frequency" type="double" value="2.0"/>
```

```
<param name="goal_aliasing" type="double" value="$(arg
sensor_range)"/>
```

```
#All standard costmap_2d parameters as in move_base, other than
BoundedExploreLayer
```

```
<rosparam ns="explore_costmap" subst_value="true">
  footprint: [[-0.2667, 0.0762], [-0.2667, -0.5334], [0.2667, -0.5334],
              [0.2667, 0.0762]]
```

```
transform_tolerance: 0.25
```

```
update_frequency: 5.0
```

```
publish_frequency: 5.0
```

```
#must match incoming static map global_frame: map robot_base_frame:
base_link resolution: 0.08
```

```
rolling_window: false track_unknown_space: true
```

```
plugins:
```

```
- {name: static,          type: "costmap_2d::StaticLayer"}
```

```
- {name: explore_boundary, type:
  "frontier_exploration::BoundedExploreLayer"}
```

```
#Can disable sensor layer if gmapping is fast enough to update scans
```

```
- {name: sensor,         type: "costmap_2d::ObstacleLayer"}
```

```
- {name: inflation,      type: "costmap_2d::InflationLayer"}
```

```
static:
```

```
#Can pull data from gmapping, map_server or a non-rolling costmap
```

```
#Uses a global map source, can also be used without. Just use
```

```

/move_base/global_costmap/costmap map_topic: /rtabmap/grid_map
subscribe_to_updates: true

explore_boundary: resize_to_boundary: false frontier_travel_point: middle
    #set to false for unknown environment
    exploration, true if reexploring
    explore_clear_space: false

    sensor:
        observation_sources: laser
        laser: {data_type: LaserScan, clearing: true, marking: true, topic:
scan, inf_is_valid: true, raytrace_range: $(arg sensor_range), obstacle_range:
$(arg sensor_range)}

inflation: inflation_radius: 0.1
</rosparam>
</node>

</launch>

```

E) Region_exploration.rviz

GNU nano 2.5.3
File: regioin_exploration.rviz

Panels:

```

Class: rviz/Displays Help Height: 78 Name: Displays Property Tree Widget: Expanded:
- /Global Options1
- /Status1
- /Marker1
/MarkerArray1 Splitter Ratio: 0.5
Tree Height: 778
Class: rviz/Selection Name: Selection
Class: rviz/ToolProperties Expanded:
- /2D Pose Estimate1
- /2D Nav Goal1
/Publish Point1 Name: Tool Properties
Splitter Ratio: 0.588679016
Class: rviz/Views Expanded:
/Current View1 Name: Views Splitter Ratio: 0.5
Class: rviz/Time Experimental: false Name: Time SyncMode: 0 SyncSource: ""
Toolbars: toolButtonStyle: 2 Visualization Manager: Class: ""
Displays:
- Alpha: 0.5
Cell Size: 1 Class: rviz/Grid
Color: 160; 160; 164
Enabled: true Line Style:

```

Line Width: 0.0299999993 Value: Lines

Name: Grid

Normal Cell Count: 0 Offset:

X: 0

Y: 0

Z: 0

Plane: XY

Plane Cell Count: 10

Reference Frame: <Fixed Frame> Value: true

- Class: rviz/TF Enabled: true Frame Timeout: 15 Frames:

All Enabled: true Marker Scale: 1 Name: TF

Show Arrows: true Show Axes: true Show Names: true Tree:

{}

Update Interval: 0 Value: true

- Alpha: 1

Autocompute Intensity Bounds: true Autocompute Value Bounds:

Max Value: 10

Min Value: -10 Value: true Axis: Z

Channel Name: intensity Class: rviz/LaserScan Color: 255; 255; 255

Color Transformer: Intensity Decay Time: 0

Enabled: true

Invert Rainbow: false Max Color: 255; 255; 255

Max Intensity: 0

Min Color: 0; 0; 0

Min Intensity: 0 Name: LaserScan

Position Transformer: XYZ Queue Size: 10

Selectable: true Size (Pixels): 3

Size (m): 0.00999999978

Style: Flat Squares Topic: /scan Unreliable: false

Use Fixed Frame: true Use rainbow: true Value: true

- Alpha: 0.699999988

Class: rviz/Map Color Scheme: map Draw Behind: false Enabled: true

Name: Map Topic: /map Unreliable: false

Use Timestamp: false Value: true

- Class: rviz/Image Enabled: false

Image Topic:

/camera/depth/im

age_raw Max

Value: 1

Median window: 5

Min Value: 0 Name: Image

Normalize Range: true Queue Size: 2 Transport Hint: raw Unreliable: false Value: false

- Class: rviz/Marker Enabled: true

Marker Topic:

/exploration_polygon_m

arker Name: Marker

Namespaces:

{}

```

Queue Size: 100 Value: true
- Alpha: 0.699999988
Class: rviz/Map Color Scheme: map Draw Behind: false Enabled: true Name: Map
Topic: /explore_server/explore_costmap/costmap Unreliable: false
Use Timestamp: false Value: true
- Alpha: 1
Autocompute
Intensity Bounds:
true Autocompute
Value Bounds:
  Max Value: 10
  Min Value: -10 Value: true Axis: Z
Channel Name: intensity Class: rviz/PointCloud2 Color: 255; 255; 255
Color Transformer: "" Decay Time: 0 Enabled: true
Invert Rainbow: false Max Color: 255; 255; 255
Max Intensity: 4096
Min Color: 0; 0; 0
Min Intensity: 0 Name: PointCloud2
Position Transformer: "" Queue Size: 10

Selectable: true Size (Pixels): 3
Size (m): 0.00999999978
Style: Flat Squares
Topic:
/explore_server/explore_costmap/explore_boundary
/frontiers Unreliable: false
Use Fixed Frame: true Use rainbow: true Value: true
- Class: rviz/MarkerArray Enabled: true
Marker Topic: /frontiers Name: MarkerArray Namespaces:
  {}
Queue Size: 100 Value: true
Enabled: true Global Options:
Background Color: 48; 48; 48 Default Light: true
Fixed Frame: map Frame Rate: 30 Name: root
Tools:
- Class: rviz/Interact
  Hide Inactive Objects: true
- Class: rviz/MoveCamera
- Class: rviz/Select
- Class: rviz/FocusCamera
- Class: rviz/Measure
Class: rviz/SetInitialPose Topic: /initialpose
- Class: rviz/SetGoal
  Topic: /move_base_simple/goal
Class: rviz/PublishPoint Single click: true
Topic: /clicked_point Value: true
Views:
Current:
Class: rviz/Orbit Distance: 15.2439346 Enable Stereo Rendering:
Stereo Eye Separation: 0.0599999987 Stereo Focal Distance: 1
Swap Stereo Eyes: false Value: false
Focal Point:
X: -0.127001464
Y: -0.259129792

```

Z: 0.273035973

Focal Shape Fixed Size: true Focal Shape Size: 0.0500000007 Invert Z Axis: false

Name: Current View

Near Clip Distance: 0.00999999978 Pitch: 0.210398525

Target Frame: <Fixed Frame> Value: Orbit (rviz)

Yaw: 1.97855985

Saved: ~ Window Geometry: Displays:

collapsed: false Height: 1056

Hide Left Dock: false Hide Right Dock: true Image:

collapsed: false QMainWindow State:

000000ff00000000fd0000000400000000000016a00000399fc0200000009fb000000120053006
5006c00650063007400690
06f006e00000001e10000009b0000006100ffffffb0000001e0054006f006f006c00200050007200
6f007000650072007400690
065007302000001ed000001df00000185000000a3fb00000012005600690065007700730020005
4006f006f02000001df0000
02110000018500000122fb0000000200054006f006f006c002000500072006f00700065007200740
0690065007300320300000
2880000011d000002210000017afb000000100044006900730070006c006100790073010000002
8000003990000000d700ffff
ffb0000002000730065006c0065006300740069006f006e0020006200750066006600650072020
0000138000000aa0000023a
00000294fb000000014005700690064006500530074006500720065006f02000000e6000000d20
00003ee0000030bfb0000000
c004b0069006e0065006300740200000186000001060000030c00000261fb0000000a0049006d
00610067006500000001dc0
00000de0000001600fffff000000010000010f00000292fc0200000003fb0000001e0054006f006
f006c002000500072006f007
0006500720074006900650073010000004100000078000000000000000fb0000000a005600690
06500770073000000000280
0000292000000ad00fffffb0000001200530065006c0065006300740069006f006e010000025a00
0000b20000000000000000
00000002000004900000000a9fc0100000001fb0000000a00560069006500770073030000004e000
00080000002e100000197
000000030000073f0000003bfc0100000002fb0000000800540069006d006501000000000000073
f0000030000fffffb00000
00800540069006d006501000000000000045000000000000000000005cf00000399000000040
0000004000000080000000
8fc0000000100000002000000010000000a0054006f006f006c00730100000000fffffb000000000
0000000

Selection: collapsed: false Time:

collapsed: false Tool Properties: collapsed: false Views: collapsed: true Width: 1855

X: 65

Y: 24

Unit Conversion Factors

Multiply	By	To Obtain
feet	0.3048	meters
inches	0.0254	meters
microns	1.0 E-06	meters
miles (nautical)	1,852	meters
miles (U.S. statute)	1,609.347	meters
yards	0.9144	meters

Acronyms

3-D	three-dimensional
6DoF	six degrees of freedom
COTS	commercial off the shelf
CUDA	Compute Unified Device Architecture
DUA	Dense Urban Areas
DWA	Dynamic Window Approach
EKF	extended Kalman filter
ERDC-GRL	Engineer Research and Development Center – Geospatial Research Laboratory
Geo ICUE	Geo-Intelligence and Complex Urban Environments
GPU	Graphics processing unit
GUI	Graphical user interface
GVR-BOT	Army Ground Vehicle Robot
ICP	Iterative Closet Point
IMU	Inertial Measurement Unit
IR	Infrared Radiation
LIDAR	LIght Detection And Ranging systems
MCoE	Manuever Center of Excellence
NUC	Next Unit of Computing
PdM	Program Manager

RAS	Robotic and Autonomous System
RGBD	Red, Green, Blue and Depth
ROS	Robot Operating System
RMS	root mean squared
RTAB-Map	Real-Time Appearance-Based Mapping
SBC	single board computer
SLAM	Simultaneous Localization and Mapping
SSGF	Standard Shareable Geospatial Foundation
SubT	Subterrain
SWaP-C	Size, Weight, Power and Cost
TF	Transforms
TOF	Time of Flight
UGV	Unmanned Ground Vehicle
UKF	unscented Kalman filter
URDF	Universal Robot Description Format
UWB	ultra-wideband
VGA	Video Graphics Array

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing this collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden to Department of Defense, Washington Headquarters Services, Directorate for Information Operations and Reports (0704-0188), 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to any penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number. **PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.**

1. REPORT DATE (DD-MM-YYYY) March 2020			2. REPORT TYPE Final		3. DATES COVERED (From - To)	
4. TITLE AND SUBTITLE Optimized Low Size, Weight, Power and Cost (SWaP-C) Payload for Mapping Interiors and Subterranean on an Unmanned Ground Vehicle					5a. CONTRACT NUMBER	
					5b. GRANT NUMBER	
					5c. PROGRAM ELEMENT NUMBER 62784	
6. AUTHOR(S) Garry Glaspell, Steven Lessard, Benjamin Christie, Kyle Jannak-Huang, Noah Wilde, Weiyu He, Osama Ennasr, Daniel Pham, Daniel Hasemann, Phil Devine, and John Kiene					5d. PROJECT NUMBER 855	
					5e. TASK NUMBER 23	
					5f. WORK UNIT NUMBER	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Geospatial Research Laboratory U.S. Army Engineer Research and Development Center 7701 Telegraph Road Alexadria, VA 22315-3864					8. PERFORMING ORGANIZATION REPORT NUMBER ERDC/GRL TR-20-6	
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) U.S. Army Corps of Engineers Washington, DC 20314-1000					10. SPONSOR/MONITOR'S ACRONYM(S)	
					11. SPONSOR/MONITOR'S REPORT NUMBER(S)	
12. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.						
13. SUPPLEMENTARY NOTES						
14. ABSTRACT Section 3 of the FY15 Force 2025 Maneuvers Annual Report indicates that in Dense Urban Areas (DUA), specifically in a subsurface, surface, or super-surface structure, the ability to identify threats will be diminished. Most commercially available Light Detection And Ranging (LIDAR) systems are specifically designed for high-resolution aerial imaging and mapping applications. As a result, they tend to be large, heavy, power-hungry, data bandwidth intensive, and expensive. They also employ lasers that are not typically eye-safe, which limits their overall effectiveness in subterranean and the interiors of subsurface or super-surface structures. However, due to recent advances in the automotive industry, there are new generations of Size, Weight, Power, and Cost (SWaP-C) sensors that are eye-safe, making them suitable for use indoors and in subterranean environments. While these tradeoffs limit their effective use to hundreds of meters (compared to kilometers for their more expensive counterparts), they are ideal candidates for use in subterranean and building interiors. While cameras fill this niche to some extent, the volumetric calculations provided by these sensors provide additional intelligence to shape the security of the environment and offer more precision when maneuvering troops. These sensors would provide the warfighter with situational understanding in previously inaccessible locations. Therefore, to aid in the Army's need to obtain and maintain situational understanding in DUAs, the authors propose utilizing low size, weight, power, and cost (SWaP-C) sensors, on a robot platform, for surveying and mapping underground structures and building interiors. Rapid/near real-time data processing is possible by utilizing open-source software and commercial off the shelf (COTS) components. Using the preferred sensor payload autonomously was also explored.						
15. SUBJECT TERMS Military robots Military surveillance Detectors			Military robots Cities and towns Underground construction – Military surveillance		Buildings – Military surveillance	
16. SECURITY CLASSIFICATION OF:				17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT Unclassified	b. ABSTRACT Unclassified	c. THIS PAGE Unclassified	SAR	218	19b. TELEPHONE NUMBER (include area code)	