ERDC TR-23-16

US Army Corps
of Engineers®
Engineer Research and
Development Center

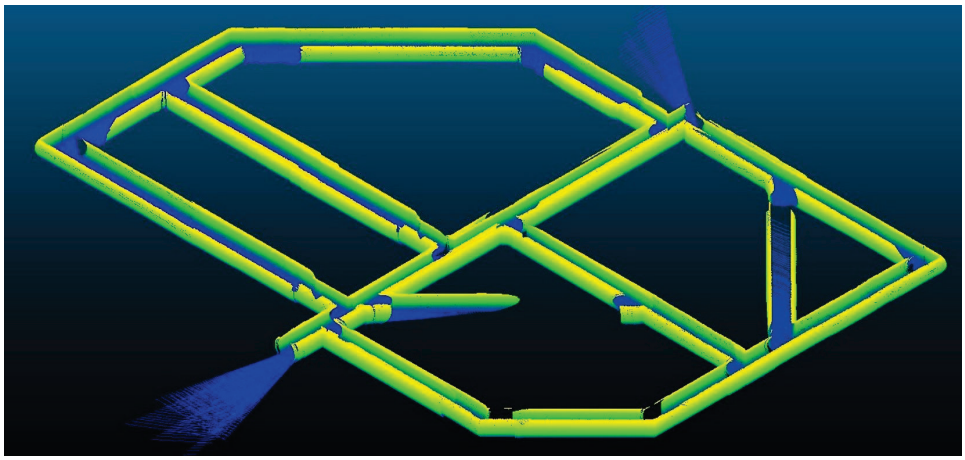
ERDC
ENGINEER RESEARCH & DEVELOPMENT CENTER

Sewer Network Interrogation Program (SNIP)

Mapping and Localization Within a Mock Sewer System

Brandon Dodd, Osama Ennasr, Amir Naser,
Chuck Ellison, Jason Ray, Garry Glaspell, and
Anton Netchaev

September 2023



Engineer Research and
Development Center

The US 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.erdc.usace.army.mil.

To search for other technical reports published by ERDC, visit the ERDC online library at https://erdclibrary.on.worldcat.org/discovery.

ERDC TR-23-16
September 2023

Mapping and Localization Within a Mock Sewer System

Brandon Dodd, Osama Ennasr, Amir Naser, and Garry Glaspell

US Army Engineer Research and Development Center (ERDC)
Geospatial Research Laboratory (GRL)
7701 Telegraph Road
Alexandria, VA 22315-3864

Chuck Ellison, Jason Ray, and Anton Netchaev

US Army Engineer Research and Development Center (ERDC)
Information Technology Laboratory (ITL)
3909 Halls Ferry Road
Vicksburg, MS 39180-6199

Final Technical Report (TR)

Prepared for Headquarters, US Army Corps of Engineers
Washington, DC 20314-1000

Under Program Element No. 0603463A, Project No. AR6

Abstract

Herein, we explored a robot's ability to localize and map, both in simulation and on a physical robot, within a mock sewer system. Mapping and localization techniques were first developed and tested in simulation and were then transitioned to the actual robot for additional physical testing. Several odometry and simultaneous localization and mapping (SLAM) techniques, including gmapping, SLAM toolbox, elevation mapping, and RTABMap, were evaluated for this particular environment. The results of the odometry and the various SLAM approaches are discussed in detail.

Contents

Figures and Tables

Figures

Tables

Preface

This study was conducted for Headquarters, US Army Corps of Engineers under Program Element No. 0603463A, Project No. AR6.

The work was performed by the Data Representation Branch of the Topography Imagery and Geospatial Research Division, US Army Engineer Research and Development Center, Geospatial Research Laboratory (ERDC-GRL). At the time of publication, Mr. Vineet Gupta was branch chief, Mr. Jeff Murphy was division chief, and Dr. Austin Davis was the technical director of the Geospatial Research Laboratory. The deputy director of ERDC-GRL was Ms. Valerie L. Carney, and the director was Mr. David R. Hibner.

The work was also performed by the Computational Science and Engineering Division of the ERDC Information Technology Laboratory (ERDC-ITL). At the time of publication, Dr. Jeffrey Hensley was division chief, and Dr. Rob Wallace and Mr. Ken Pathak were the technical directors of ITL. The deputy director of ERDC-ITL was Dr. Jacqueline Pettway, and the director was Dr. David Horner.

The authors would like to acknowledge Mr. Steven Bunkley and Mr. Mike Paquette for their contributions to this project.

COL Christian Patterson was commander of ERDC, and Dr. David W. Pittman was the director.

1 Introduction

1.1 Background

The Sewer Network Interrogation Program (SNIP) is a task within the greater Understanding the Environment as a Threat (UET) program under the Engineer Research and Development Center (ERDC) research and development area Installations and Operational Environments. While the UET effort is focused on developing tools and methods for identifying toxic industrial chemicals and materials (TIC/Ms) in subterranean (SubT) environments and reverse point sourcing their location, SNIP is responsible for the following specific tasks:

- Developing sensors for persistent surveillance of TIC/Ms
- Placing the sensors in key locations within the SubT environment
- Relaying the data from the sensors to users outside of the SubT environment

However, the ability to place sensors in key locations within the SubT environment is contingent on unmanned ground vehicles (UGVs) understanding their immediate environment. As a result, this report explores the various nuances of localization and mapping within a mock sewer system, both in simulation and with a physical robot.

1.2 Objective

This report addresses the focus areas established in the *Army Multi-Domain Intelligence: FY21–22 S&T Focus Areas* (Office of the Deputy Chief of Staff 2020). Specifically, this work addresses the statement, "Wars will be fought at hyper speed and scale, dominated by technologies such as robotics and autonomous systems (RAS), machine learning (ML), and AI [artificial intelligence] capabilities, which are widely available, packaged, and ready for use" (5). While the overall mission of UET addresses this focus area by establishing a methodology to reverse point source or obtain an immediate warning of threats in a SubT environment using unmanned robotic payloads, the first step, which is the primary objective of this report, is for the robot to localize and map within the SubT environment.

1.3 Approach

To test the robot's ability to simultaneously localize and map in a relevant SubT environment, the SNIP team constructed a mock sewer system. The mock sewer system covers an area of approximately 2,700 sq ft* and is composed of 95 ft of 30 in. diameter tubes and 182.5 ft of 24 in. diameter tubes. The longest straight section measures 40 ft. Figure 1 shows the completed mock sewer system. While initial tests, and this report, focus on localizing and mapping, subsequent tests will involve dropping sensor nodes from the UGV platform.

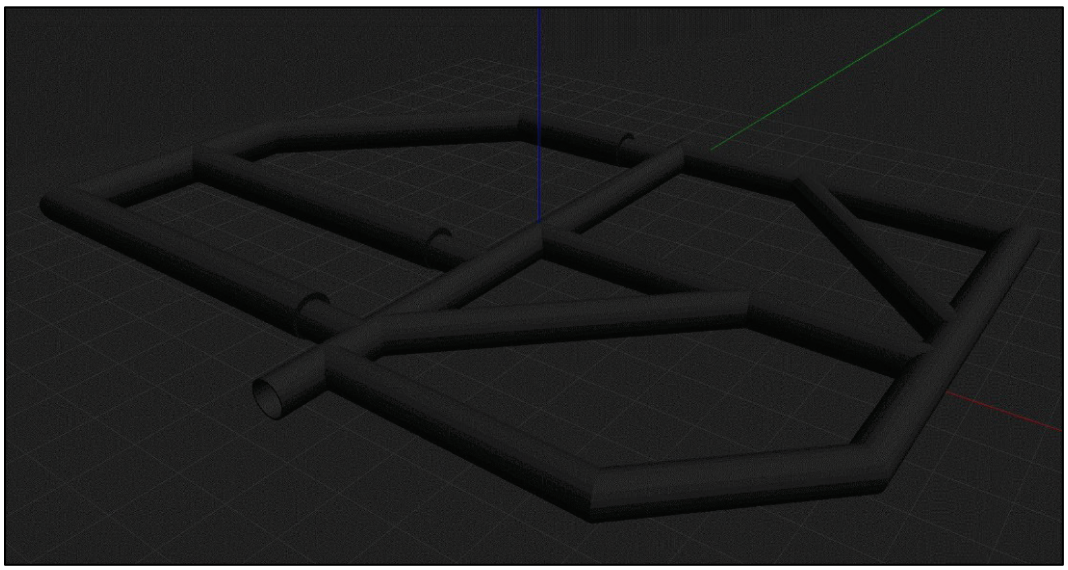
Figure 1. Mock sewer system.

To rapidly test various robotic hardware and software options, we developed a simulated environment based on the mock sewer system shown in Figure 1. The simulated world is shown in Figure 2. The simulated mock sewer system has the same dimensions as its physical counterpart. We anticipated that this would be an extremely tough environment for the robot

* For a full list of the spelled-out forms of the units of measure used in this document, please refer to *US Government Publishing Office Style Manual*, 31st ed. (Washington, DC: US Government Publishing Office, 2016), 248–252, https://www.govinfo.gov/content/pkg/GPO-STYLEMANUAL-2016/pdf/GPO-STYLEMANUAL-2016.pdf.

to localize and map. The lack of visual cues and expected difficulty turning in such a narrow corridor ruled out the use of visual odometry and wheel encoders for localization. However, the simulated environment allowed us to quickly and efficiently test a variety of methodologies for localization.

Figure 2. Simulated mock sewer system.



We planned to use the QinetiQ SPUR, which was named the winner of the Common Robotic System Individual Program of Record (QinetiQ, n.d.), as our main robotic platform. However, the SPUR uses the Interoperability Profile (IOP), and our navigation stack leverages the Robot Operating System (ROS), specifically Noetic Ninjemys. An IOP-to-ROS bridge is being developed for the SPUR. While we wait on its development, we plan to develop on the Rover Robotics Flipper Rover Pro, which is ROS-capable as received from the vendor. Once the bridge is complete, we will transfer both the hardware and software to the SPUR. Unfortunately, neither the SPUR nor the Flipper Rover Pro have simulated versions we can use with our virtual mock sewer system. Thus, for the simulation, we used the Clearpath Jackal, which has a footprint that is similar to those of the SPUR and the Flipper Rover Pro. The dimensions for all three robots are provided in Table 1.

Table 1. Robot footprints.

Robot	SPUR	Rover PRO	Jackal
Length (in.)	33	24	20
Width (in.)	18	15	17

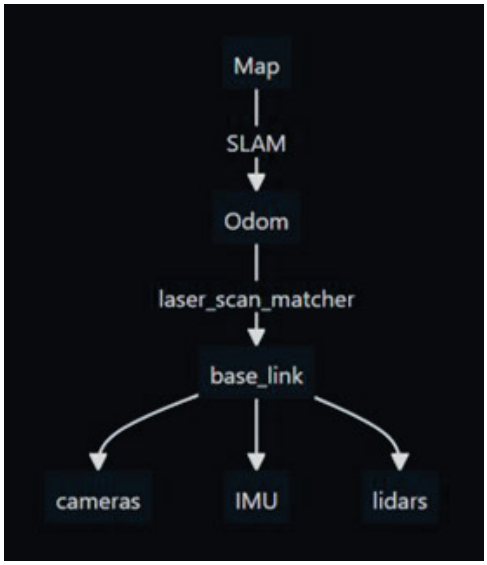
In robotics, the ability to effectively transform between the coordinate systems used by various applications and devices is critical. In ROS, the transform (tf) tree fills this role by tracking multiple frames (i.e., coordinate systems) over time and provides methods for transforming from one to another. Each entry in the tree has a single parent but can have multiple children. This tree structure achieves high flexibility but ensures that only one path exists between any two frames. By searching through this tree and summing the transforms, ROS is able to provide a transform from any point in the model to any other point in that model.

By convention, there is generally a master frame, designated *base link,* on the robot that serves as the top node of a tree and represents all of the critical locations and local coordinate systems used by the various sensors and applications. Odometry consists of tracking the movement of base link relative to an assumed stationary frame of reference, commonly designated the *odom* frame, which serves as the parent frame for base link. While it is possible to stop the tree here, it is generally convenient to create an additional frame, commonly designated *map*, as the parent of odom. Odom and map are similar in that they are static (i.e., ideally stationary in the real world). While it may seem strange to maintain two static frames, it is convenient because separate algorithms can maintain each frame, with each focused on its own unique use case.

The odom frame is generally tuned to be fast, smooth, and accurate over short time windows, providing the highest possible accuracy in terms of relative motion. In contrast, the map frame is allowed to make the sudden corrections that are common with simultaneous localization and mapping (SLAM) techniques, allowing it to provide the current best guess at the absolute position of the robot, regardless of how that deviates from the last update. Stated simply, map frame is focused on the best absolute positional accuracy, regardless of how unstable the frame seems thorough time; odom is smooth and consistent, even if it results in long-term drift. A representative tf tree for the simulated robot is shown in Figure 3. Each sensor has its own frame in the tree. The simulated robot is equipped with a front-facing red-green-blue and depth (RGB-D) camera, an inertial

measurement unit (IMU), and a Velodyne VLP 16 beam lidar. The physical robot is equipped with front- and rear-facing FLIR Boson 640 thermal cameras, a LORD Microstrain 3DM-GX5-25 IMU, and an Ouster OS-1 64 beam lidar.

Figure 3. A transform (tf) tree.



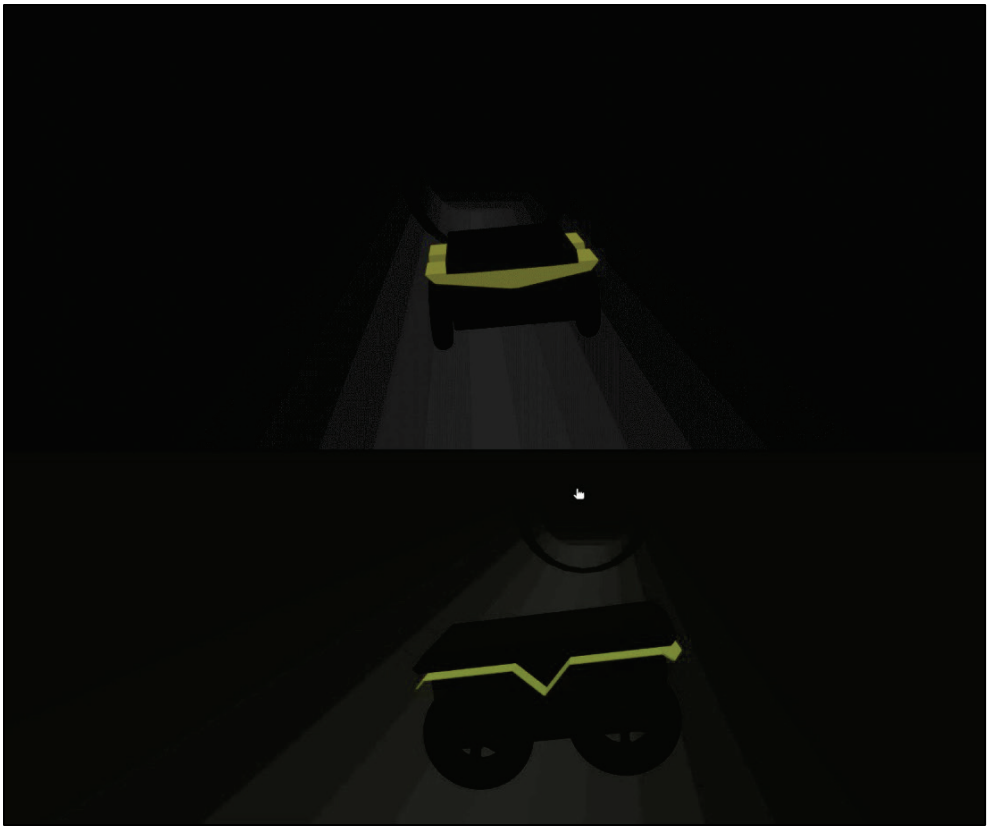
2 Research and Discussion

If we assume an unknown environment with no prior information, then the robot has to simultaneously localize and map. There are a number of sensors, including GPS, cameras, wheel encoders, IMUs, and lidars (both 2D and 3D), that can help the robot localize. Of these, GPS is the only one that provides absolution position; the rest provide velocity and acceleration or relative position. However, in this particular use case, we were operating in a GPS-denied environment, limiting us to relative-position calculations. Also, using cameras to perform visual odometry was not a viable approach due primarily to the lack of unique visual cues inside of a sewer system. Wheel encoders were also problematic because of the amount of wheel slippage expected in a potentially wet and slick environment. As a result, we focused on using lidar and an IMU to generate odometry, generally called *lidar inertial odometry*. The sections that follow demonstrate how we set up the robot's odometry and its application to various SLAM approaches and document the results in both simulation and on a physical robot.

2.1 Simulation

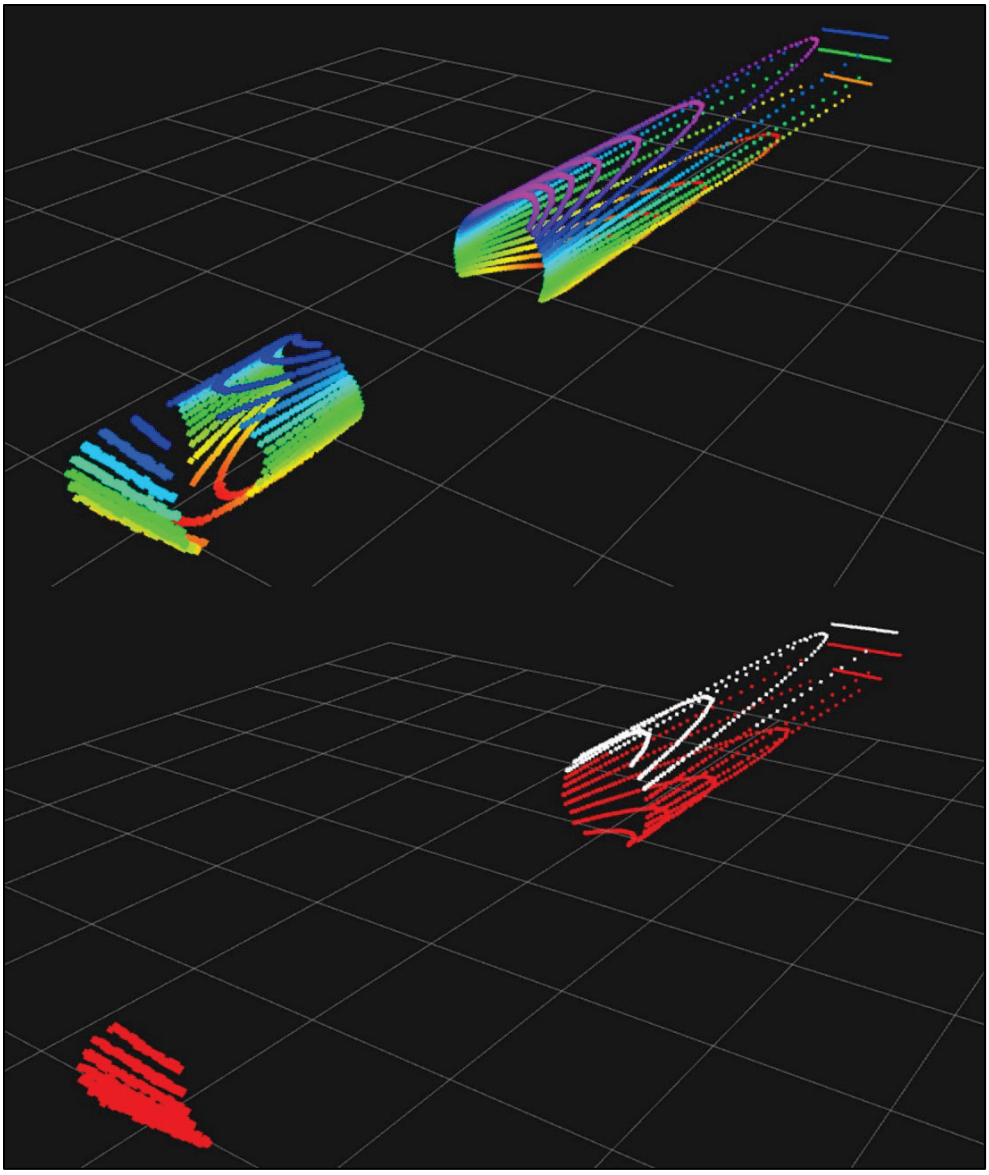
As mentioned previously, we used the Clearpath Jackal robot for testing in simulation. Figure 4 shows the Jackal inside the simulated mock sewer system. There were a number of things that stood out when navigating the Jackal in simulation. First, if the robot ended up orthogonal to the tunnel, shown in the bottom of Figure 4, it lacked the traction and torque to correct itself. Second, if not careful, the robot tended to drive up the side of the tube and flip when attempting to navigate from one tube to another. Third, we observed a fair amount of wheel slip, especially at junctions where the tubes connected.

Figure 4. Clearpath Jackal in simulated mock sewer system.



Finally, we noticed that `segmenters lib` (AutoLidarPerception 2020), a package we typically use to separate the ground plane from potential obstacles, also did not work in this environment. Figure 5 shows the results of applying `segmenters lib` to the simulated mock sewer system. The top of the image shows the raw point cloud false colored as a function of the *z*-axis. The bottom image shows the results after applying `segmenters lib`. Points identified as ground are shown in red, and points identified as nonground are identified in white. It is unclear at this time why an entire section of the point cloud is missing and why only the back half points above the horizon are labeled as nonground after applying `segmenters lib`. In spite of these obstacles, the next section explains how we localized the robot.

Figure 5. Lidar segmentation in simulated mock sewer system.



2.1.1 Odometry

To generate odometry, we used the `laser_scan_matcher` (LSM) package (Censi 2008). Rather than using frame-to-frame matching, the LSM package compares laser scans to key frames to generate a pose topic. The use of key frames results in less drift, especially when the robot is stationary or making difficult maneuvers, such as transitioning to a different tube in the mock sewer system. The key frames generated are determined by the `kf_dist_linear` and `kf_dist_angular` parameters. The `kf_dist_linear` parameter is used to specify a minimum distance the robot travels before establishing a new keyframe. Similarly, the `kf_dist_angular` establishes a

new key frame after the robot turns a certain number of radians. The LSM package subscribes to either the LaserScan message `scan` or the PointCloud2 message `cloud` and publishes the Pose2D message `pose2D` (CCNY Robotics Lab 2022b). Alternatively, we can also pass either `imu/Data` or `odom` as a guess to speed up the scan registration process. When either an `imu/Data` or `odom` is passed, rather than calculating every possibility, LSM uses those data as an initial estimate and then optimizes the estimate using the laser scan data. In our particular use case, we only used the IMU as a guess. The node that follows was used to launch LSM.

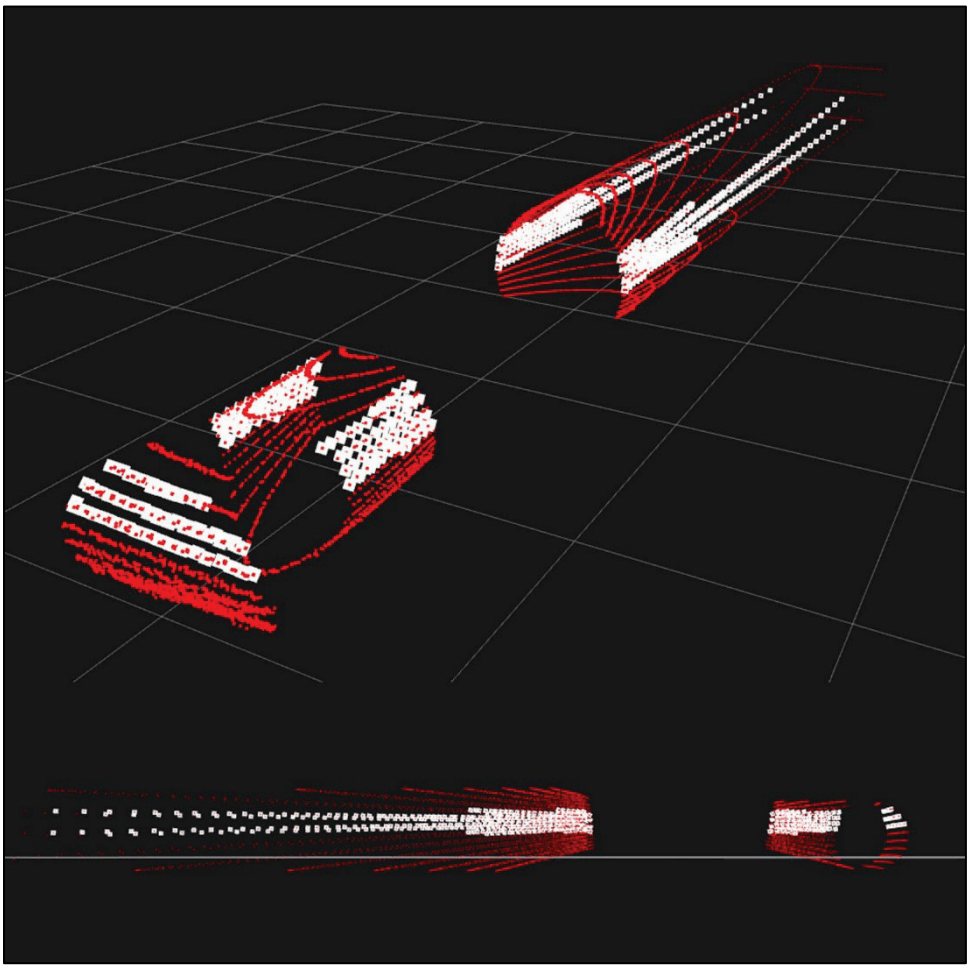
```
 <node pkg="laser_scan_matcher" type="
  laser_scan_matcher_node"
  name="laser_scan_matcher_node" output="screen">
  <param name="fixed_frame"       value="odom"/>
  <param name="max_iterations"    value="30"/>
  <param name="use_imu"           value="true"/>
  <param name="use_odom"          value="false"/>
  <param name="use_cloud_input"   value="true"/>
  <param name="publish_tf"        value="false"/>
  <param name="publish_pose"      value="false"/>
  <param name="publish_pose_stamped" value="false"/>
  <param name="kf_dist_linear"    value="0.30"/>
  <param name="kf_dist_angular"   value="0.5236"/>
  <param name="cloud_range_min"   value="0.3"/>
  <param name="cloud_range_max"   value="100.0"/>
  <remap from="scan"              to="/pc_2_ls"/>
  <remap from="cloud"             to="/cloud_ortho"/>
  <remap from="imu"               to="/imu/data"/>
  <remap from="odom"              to="/odom"/>
  <param name="do_compute_covariance" value="1"/>
  <param name="publish_pose_with_covariance" value="
 false"/>
  <param name="publish_pose_with_covariance_stamped"
 value="true"/>
 </node>
"/home/garry/Downloads/tb3/elevation_mapping/costmap_2.
  yaml" />"
  </node>
```

To generate the LaserScan topic from a 3D lidar PointCloud2 message, the package `pointcloud_to_laserscan` was used (ROS Perception 2022). Ultimately, this reduced the overall complexity, by converting a 3D point cloud to a 2D laser scan, and improved computational performance. The node was launched with the code that follows. The input topic `cloud_in` subscribes to the 3D lidar topic `velodyne_lidar`. The node publishes the LaserScan topic `pc_2_ls`. The parameters `min_height` and `max_height` throttle the 3D point cloud along the *z*-axis. While the red points are the

entire point cloud, the white points in Figure 6 are indicative of setting the `min_height` and `max_height` parameters to −0.10 and 0.10, relative to the lidar frame, inside the mock sewer system. The white points serve as the input and are projected onto a 2D plane. The resulting LaserScan topic, `pc_2_ls`, can be used in the `laser_scan_matcher` node if the parameter `use_cloud_input` is set to *false*. However, due to the nature of the mock sewer system, we also investigated an alternative approach.

```
<node pkg="pointcloud_to_laserscan" type="
 pointcloud_to_laserscan_node" name=" point-
 cloud_to_laserscan">
  <param name="min_height"          value="-0.10"/>
  <param name="max_height"          value="0.10"/>
  <param name="range_min"           value="0.3"/>
  <param name="range_max"           value="100.0"/>
  <param name="target_frame"        value="velodyne"/>
  <remap from="cloud_in" to="/velodyne_points"/>
  <remap from="scan" to="/pc_2_ls"/>
</node>
```

Figure 6. Filtering the point cloud along the *z*-axis.



Because the mock sewer system was tubular, we could not assume that the robot would not roll along the *x*-axis as it drove along the tunnel. An alternative approach was to use the package `laser_ortho_projector` to adjust for roll along the *x*-axis (CCNY Robotics Lab 2022a). The `laser_ortho_projector` node subscribes to the LaserScan topic generated in the `pointcloud_to_laserscan` node and outputs the PointCloud2 topic `/cloud_ortho`. The `laser_ortho_projector` uses the IMU to correct for roll and pitch and reprojects the corrected LaserScan topic as a PointCloud2 topic. The `/cloud_ortho` topic is passed to the `laser_scan_matcher` node and is used when `use_cloud_input` is set to *true*.

```
   <node pkg="laser_ortho_projector" type="
laser_ortho_projector_node" name="
laser_ortho_projector" output="screen">
 <param name="publish_tf"  value="false"/>
 <param name="fixed_frame" value="odom"/>
 <param name="base_frame"  value="base_link"/>
 <param name="use_imu"     value="true"/>
 <param name="use_pose"    value="false"/>
 <remap from="scan"        to="/pc_2_ls"/>
 <remap from="imu"         to="/imu/data"/>
</node>
```

Finally, in the `laser_scan_matcher` node, we set the parameter called `publish_pose_with_covariance_st` to *true*. This was because `laser_scan_matcher` outputs a Pose2D topic, and we needed an odom topic for SLAM. Fortunately, the package `robot_localization` can import PoseWithCovarianceStamped messages and output the odom message `odometry/filtered` (Charles River Analytics 2022). The node to launch `robot_localization` (Moore and Stouch 2014) was as follows; the corresponding YAML file is in Appendix A.

```
    <node pkg="robot_localization" type="
ekf_localization_node" name="ekf_se" clear_params="
true">
  <rosparam command="load" file="/home/garry/
Downloads/tb3/robot_localization/ekf_flipperbot.yaml " />
```

2.1.2 Simultaneous Localization and Mapping (SLAM)

The sections that follow investigate various SLAM approaches in the simulated environment using the odom topic generated from the `laser_scan_matcher` and `robot_localization` nodes. Each SLAM approach provides a unique capability that may be beneficial depending on the use case.

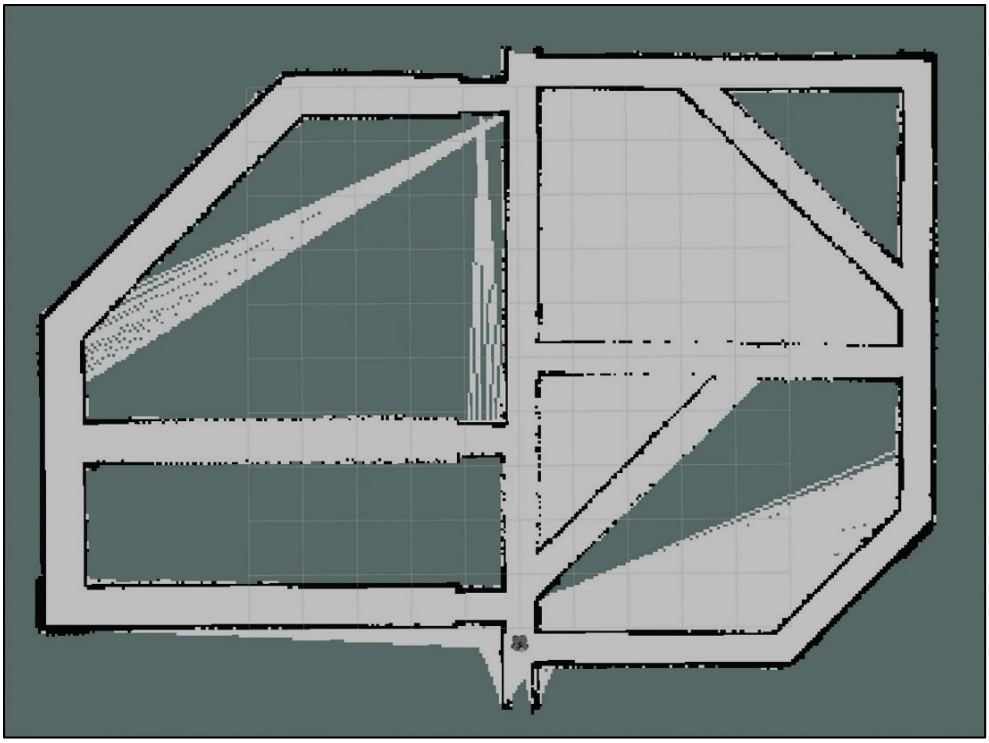
2.1.2.1 Gmapping

Gmapping is the de facto standard in regard to 2D SLAM approaches. The current repository has been maintained and improved upon since 2009 (ROS Perception 2020). As mentioned previously, gmapping is a 2D SLAM approach. Therefore, we passed the LaserScan topic `pc_2_ls`, generated from the `pointcloud_to_laserscan` node, as the input. The node to launch `slam_gmapping` follows, and the corresponding YAML file can be found in Appendix B.

```
<arg name="set_base_frame" default="base_link"/>
<arg name="set_odom_frame" default="odom"/>
<arg name="set_map_frame"  default="map"/>
<node pkg="gmapping" type="slam_gmapping" name="
 turtlebot3_slam_gmapping" output="screen">
  <remap from="/scan" to="/pc_2_ls"/>
  <param name="base_frame" value="$(arg␣set_base_frame)"/>
  <param name="odom_frame" value="$(arg␣set_odom_frame)"/>
  <param name="map_frame"  value="$(arg␣set_map_frame
)"/>
  <rosparam command="load" file="/home/garry/
 Downloads/tb3/gmapping/gmapping_params.yaml" />
</node>
```

The primary output of the `slam_gmapping` node is the OccupancyGrid topic *map*. This topic is updated as the robot moves through its environment. Figure 7 is the result of navigating through the mock sewer system. Typically, occupancy grids are trinary in nature and have three possible cell designations: obstacles, known space, and unknown space. As shown in Figure 7, obstacles are shown in black and thus avoided. Known space, shown in light gray, is equally free to traverse. Dark gray areas are unknown space. Due to working in a simulated environment, the lidar was able to see through walls in some regions, into regions that the robot did not traverse, and these were still labeled as free space. The resulting occupancy grid can be saved and used to localize the robot if there is a need to revisit the environment. Localization using previously generated occupancy grids will be discussed in Section 2.1.3.

Figure 7. Occupancy grid generated by gmapping.

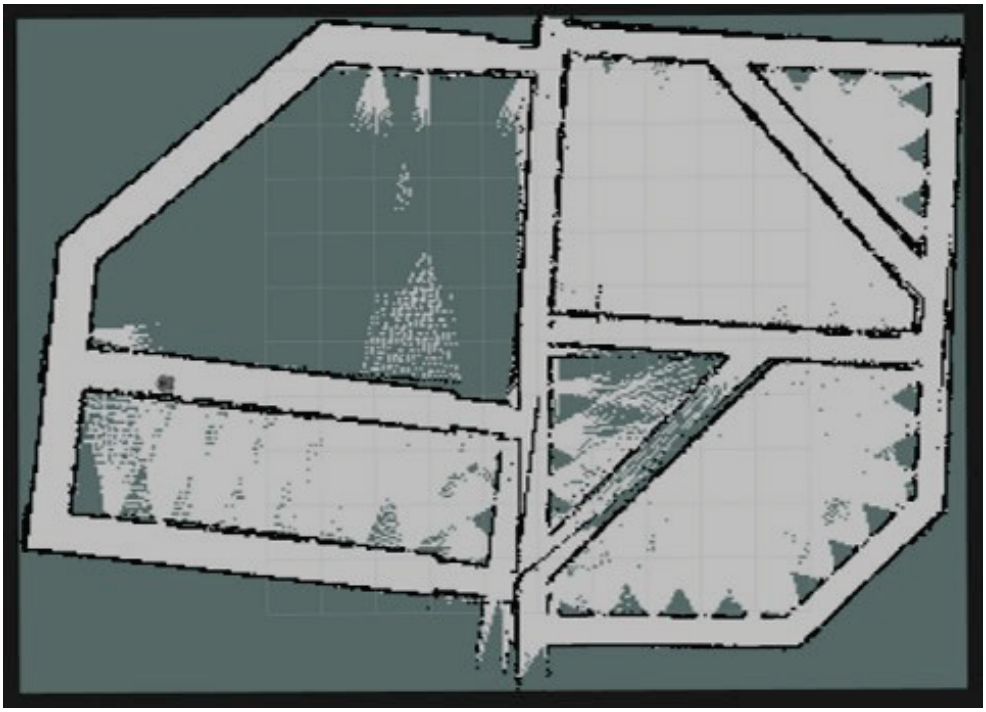


2.1.2.2 SLAM Toolbox

The package `slam_toolbox` is a more recent 2D SLAM approach (Macenski and Jambrecic 2021). Similar to gmapping, SLAM Toolbox subscribes to a LaserScan topic (Macenski 2023), and we passed the same LaserScan topic `pc_2_ls` as the input. The node also creates an occupancy grid called `map`. The node to launch SLAM Toolbox follows, and the corresponding YAML file is found in Appendix C. The resulting occupancy grid is shown in Figure 8.

```
    <node pkg="slam_toolbox" type=" sync_slam_toolbox_node"
name="slam_toolbox" output=" screen">
  <rosparam command="load" file="/home/garry/
Downloads/tb3/slam_toolbox/mapper_params_online_sync.yaml" />
</node>
```

Figure 8. Occupancy grid generated by SLAM Toolbox.



When comparing the occupancy grids between gmapping and SLAM Toolbox, the default parameters for SLAM Toolbox resulted in some drift and needed to be tuned for this particular environment. While optimizing the parameters is outside the scope of this report, the main advantage of SLAM Toolbox is the *lifelong mapping* concept. Specifically, lifelong mapping is used to update an existing occupancy grid. This is pertinent to environments that tend to change over time. Thus, if the concept of operations (CONOP) required routine surveillance or if the environment changed dynamically over time, then the lifelong mapping concept would incrementally adjust to those changes within the environment. For instance, if a region was inaccessible due to high water, we could leverage lifelong mapping to complete the map at a later time.

2.1.2.3 Elevation Mapping

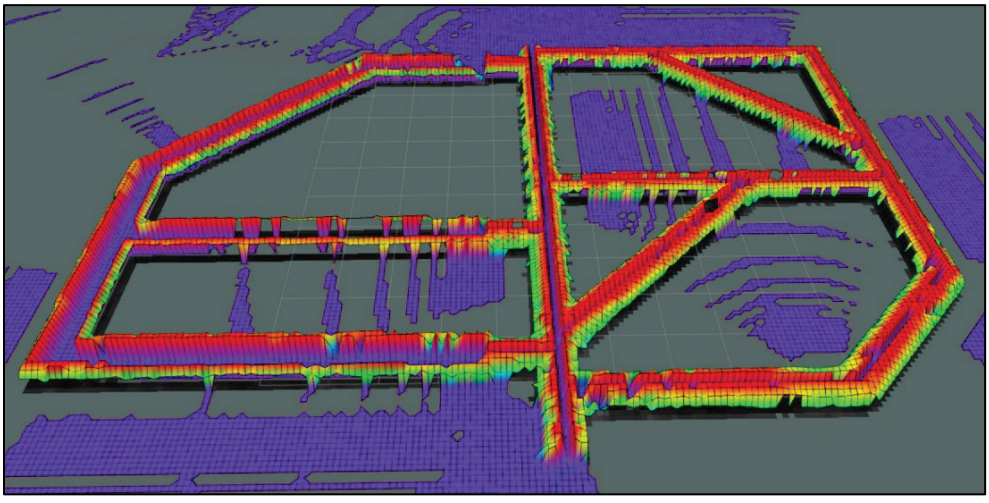
While gmapping and SLAM Toolbox are 2D approaches, elevation mapping is a 2.5D approach. In short, this means that within a 2D grid of points, each point has a height value. It is not full 3D because the model can only represent a single elevation value per grid point (generally the detected point closest to the robot), but it is still a significant improvement over simple 2D maps in complex environments (with little additional computational overhead). A 2.5D map can easily represent slope changes and

other surface features that are generally lost in 2D mapping approaches. However, it does not have the ability to represent ceilings or complex overhanging structures like a 3D map.

With elevation mapping, instead of receiving a 2D LaserScan topic, we pass it a full 3D PointCloud2 topic, `velodyne_points`, that it uses to generate a 2.5D grid map (Figure 9). Cooler colors represent lower elevation, and warmer colors represent higher elevation. One advantage of using elevation-based grid maps is that we can set upper and lower boundaries. In this particular CONOP, we could have the robot explore, and eventually stop, when approaching the lower limit. This could be useful if we assume that standing water will likely be present at a particular depth. The script to launch an elevation mapping node follows, and the corresponding YAML files are located in Appendix D through Appendix F.

```
<node pkg="elevation_mapping" type="elevation_mapping "
 name="elevation_mapping" output="screen">
   <rosparam command="load" file="/home/garry/
 Downloads/tb3/elevation_mapping/tb3.yaml" />
   <rosparam command="load" file="$(find␣
 elevation_mapping)/config/sensor_processors/
 velodyne_HDL-32E.yaml" />
   <rosparam command="load" file="/home/garry/
 Downloads/tb3/elevation_mapping/ postprocessor_pipeline.yaml" />
 </node>
```

Figure 9. Grid map generated by elevation mapping.

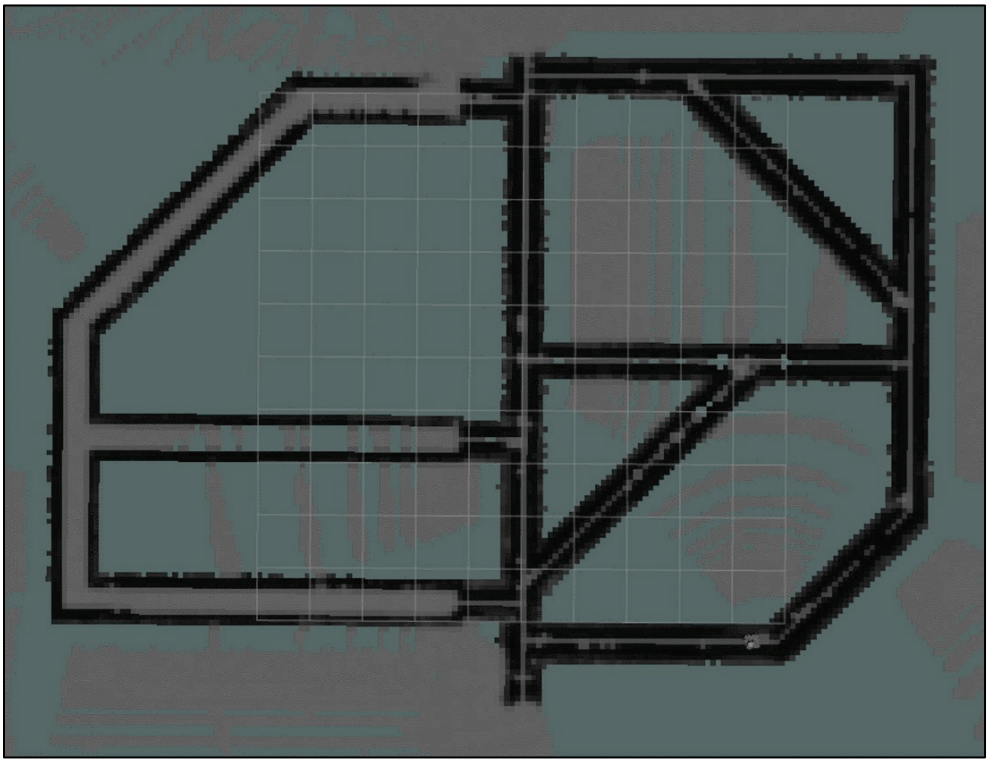


Furthermore, we can generate an occupancy grid from the 2.5D elevation map (Figure 10). The occupancy grid created from elevation mapping is not trinary in nature and uses values ranging from 0 to 255. The values increase as we approach the upper and lower boundaries set in the

`costmap.yaml` file provided in Appendix G. As the value approaches 253, the robot will slow down and eventually stop. The node to convert the 2.5D grid map to an occupancy grid follows.

```
    <node pkg="grid_map_visualization" type="
grid_map_visualization" name="grid_map_visualization "
output="screen">
  <rosparam command="load" file="/home/garry/
Downloads/tb3/elevation_mapping/costmap.yaml" />"
  </node>
```

Figure 10. Occupancy grid generated by elevation mapping.

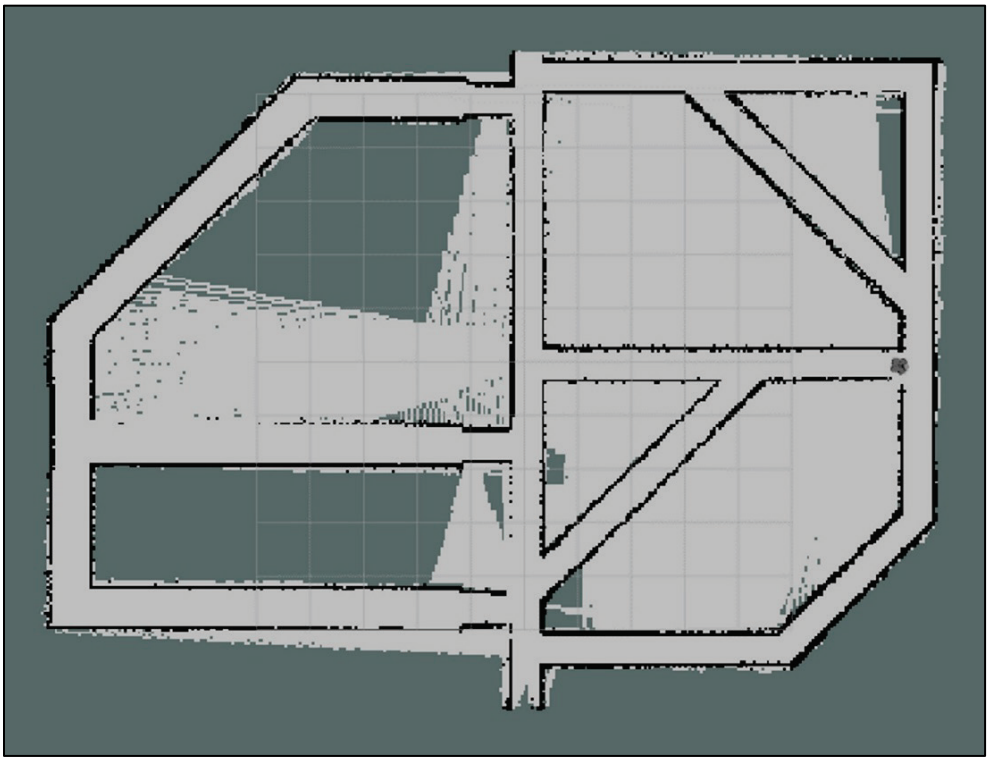


2.1.2.4 RTABMap

The final SLAM approach that we investigated was RTABMap (Labbé and Michaud 2019). Unlike the previous approaches, which only used lidar for localization and mapping, our setup, RTABMap, also used a camera to initiate a loop-closure detector. Loop closure occurs when the robot recognizes a location it has previously visited. The detector uses a bag-of-words approach to compare scenes. This is accomplished by picking points of interest in an image. To increase efficiency, similar points are clustered together. These clusters are then compared across subsequent images. Once a loop closure is suspected using the image data, lidar-based iterative closest point (ICP) is used to determine the robot's current position relative to

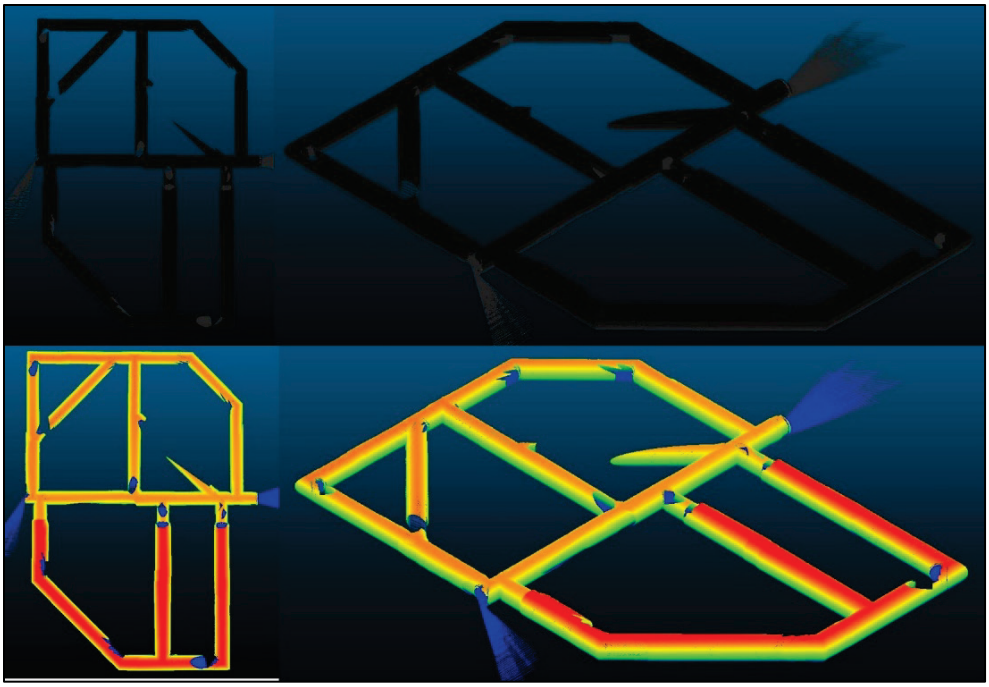
the detected prior position based on the image data, and a link is added to the underlying pose graph for use in updating the map. Optimization of this graph is how RTABMap fuses all of the stored images and lidar data. The ability to fuse both lidar and cameras to simultaneously localize and map makes this a very robust approach. We have discussed RTABMap at length in previous reports (Glaspell et al. 2020; Christie et al. 2021). Due to its length, the full launch file is provided in Appendix H. Similar to the previous SLAM approaches, we passed the LaserScan topic `pc_2_ls`. In addition, we also passed the camera topics `/camera/rgb/image_raw`, `/camera/depth/image_raw`, and `/camera/rgb/camera_info`. Also similar to the aforementioned SLAM approaches, RTABMap produces an occupancy grid (Figure 11).

Figure 11. Occupancy grid generated by RTABMap.



In contrast to the aforementioned SLAM approaches, RTABMap can also generate 3D point clouds of the environment. Figure 12 shows the 3D-generated point clouds of the mock sewer system. In the top images in the figure, the point clouds are colored based on the red-green-blue (RGB) values identified by the camera. Because it is difficult to see features with the RGB-colored point cloud, the bottom images, where the point cloud is false colored based on the *z*-axis, are also provided.

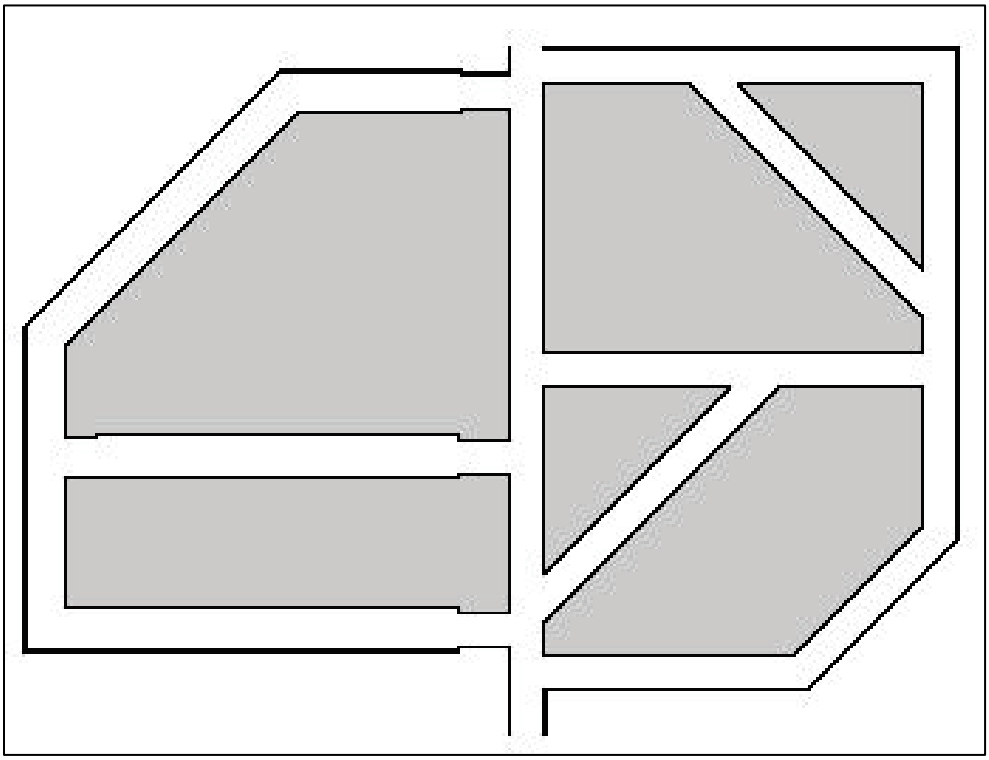
Figure 12. A 3D point cloud generated by RTABMap in the simulation.



2.1.3 Localization

While SLAM assumes no prior information is provided, localization assumes that a 2D occupancy grid is supplied. The robot will use the occupancy grid and data from its onboard sensors to determine its position. Localization is especially useful if we need to revisit a previously mapped area. For instance, future work will center around dropping sensors. If a previously dropped sensor is damaged, we can leverage the occupancy grid, generated via SLAM, to traverse back to that location and replace the broken sensor. While the occupancy grids in Figures 7, 8, 10, and 11 were possible candidates for localization, in this section, we chose a different approach. Specifically, we used the package `gazebo_ros_2Dmap_plugin` to generate an occupancy grid directly from the simulated mock sewer system (Kollmitz 2020). The occupancy grid generated from the `gazebo_ros_2Dmap_plugin` package is shown in Figure 13 and is used in subsequent sections.

Figure 13. Occupancy grid generated from `gazebo_ros_2Dmap_plugin`.



2.1.3.1 Adaptive Monte Carlo Localization (AMCL)

The Adaptive Monte Carlo Localization (AMCL) is the de facto standard in regard to 2D localization approaches. Similar to gmapping, the current repository has been maintained and improved upon since 2009 (ROS Planning 2023). The AMCL package requires two inputs. The first is the LaserScan topic `pc_2_ls`. The other is the OccupancyGrid topic `map`. To pass the map in Figure 13 to AMCL, we used the `map_server` node. The `map_server` node is provided here, and it generates the required OccupancyGrid topic `map`.

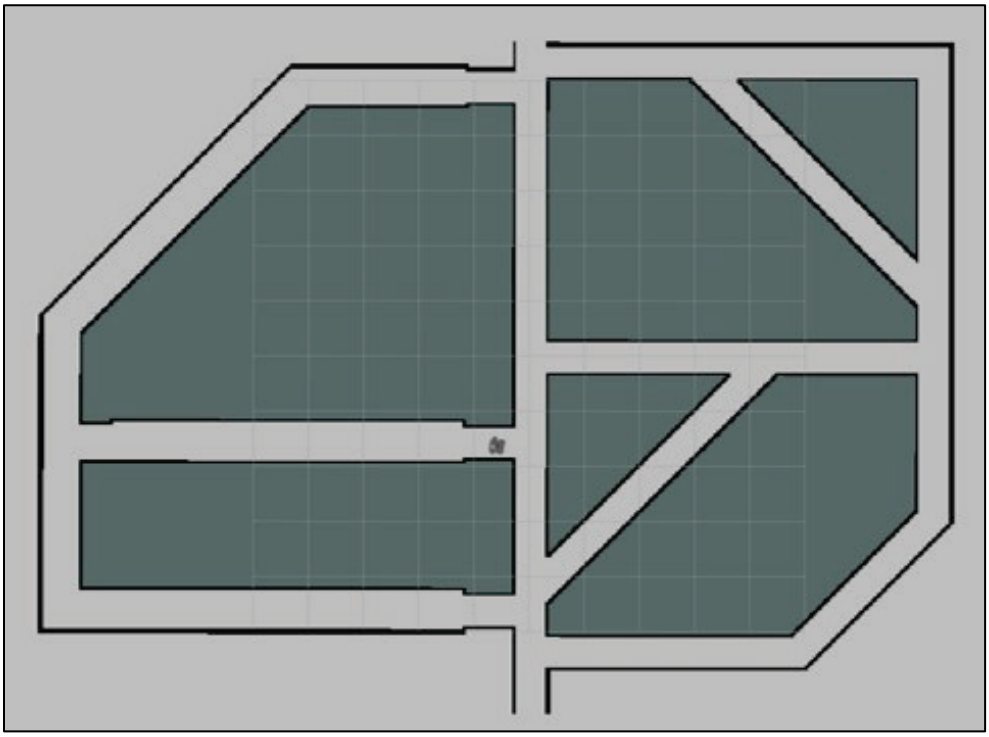
```
   <node pkg="map_server" name="map_server" type=" map_server"
args="/home/garry/Downloads/sewer_map. yaml"/>
```

The YAML file referenced in the preceding node follows. Both the YAML file and the portable gray map (PGM) image were generated from the aforementioned `gazebo_ros_2Dmap_plugin` package. However, we could just as easily have used SLAM to generate these files. The YAML file was used to scale the PGM image to match the expected values from the LaserScan topic.

```
image: sewer_map.pgm
resolution: 0.050000
origin: [-10.000000, -10.000000, 0.000000]
negate: 0
occupied_thresh: 0.65
free_thresh: 0.196
```

Figure 14 shows the result of localizing the robot with AMCL. The robot's current position is based on comparing the onboard lidar to the provided occupancy grid. Due to its length, the full AMCL launch file is provided in Appendix I. The default values worked well in our use case. However, we changed the parameter `laser_model_type` to `likelihood_field_prob`. This allowed for beam skipping. Beam skipping is useful in dynamic environments and will ignore individual beams from the LaserScan topic when they do not match the provided occupancy grid. Also, the YAML file passes an initial position of x, y and yaw. If this did not align to the robot's current position, we used `2D pose estimate` to match the LaserScan topic to the occupancy grid.

Figure 14. Simulated robot localizing off of an occupancy grid.



2.1.3.2 The i_see_pee Package

An alternative approach to localization is the `i_see_pee` package (Dorezyuk 2019). At its core, `i_see_pee` uses the ICP algorithm, which minimizes the difference between point clouds, to localize the robot. Also,

`i_see_pee` is very fast because it leverages a lookup table to store reusable information for the ICP calculation. The OccupancyGrid topic `map` is used to precompute cells close to obstacles using the k-nearest neighbor approach. The k-nearest neighbor approach compares adjacent points and removes outliers if a predetermined threshold is not met. These values are stored in the lookup table, and when the LaserScan topic `pc_2_ls` has points in one of these cells, the lookup table is used to quickly retrieve all the k-nearest neighbors. The node to launch `i_see_pee` follows, and the corresponding YAML file is found in Appendix J. We successfully used the `i_see_pee` package to localize in the simulated mock sewer system. Note `i_see_pee` automatically determines the robot's position within the occupancy grid. Thus, there is no need to use `2D pose estimate` to match the LaserScan topic to the occupancy grid, which was sometimes required when using AMCL.

```
  <node pkg="i_see_pee" type="i_see_pee_node" name="
i_see_pee_node" output="screen">
      <rosparam file="/home/garry/Downloads/tb3/
i_see_pee/i_see_pee.yaml" command="load"/>
</node>
```

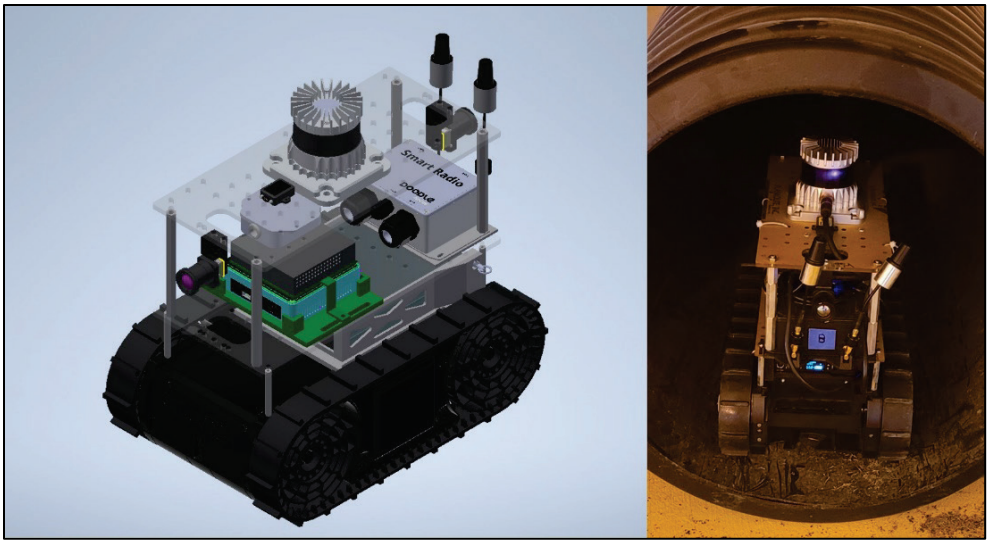
2.2 Physical Robot

The first step when transitioning from simulation to the physical robot was to ensure the robot, with the attached payload, would fit in the smaller 24 in. diameter pipes. As a result, we used computer-aided design (CAD) to determine an optimal payload configuration for the hardware. The top images in Figure 15 demonstrate the final design configuration that was used for testing. The configuration worked well for both the flipper and 4WD configurations of the Rover Robotics UGV, as shown in the bottom images of Figure 15. However, during testing, the end caps on the flipper configuration made enough contact with the sides of the tube to restrict movement. Furthermore, transitioning between tubes was difficult with the 4WD configuration. Therefore, we reached out to Rover Robotics to design shorter axles in order to run the Rover without the flippers attached. Rover Robotics designed, manufactured, and sent us the short axles so we could continue testing in the mock sewer system.

Figure 15. Computer-aided design (CAD) representations (*top*) and inside the mock sewer system (*bottom*) of the Rover flipper and 4WD configurations.

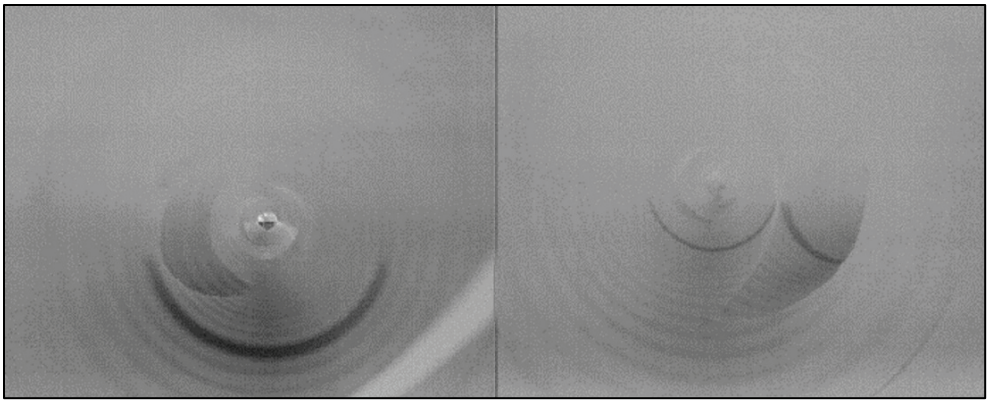
The CAD model and the Rover with the shorter axles installed are shown in Figure 16 and are referenced throughout the report as the *crawler* configuration. The Rover crawler was able to traverse the pipes and to transition between tubes with minimal effort. This was definitively the preferred configuration to test what worked in simulation on the physical robot. Of the aforementioned SLAM approaches, we chose to test RTABMap on the physical robot in an attempt to generate a 3D point cloud of the mock sewer system similar to that shown in Figure 12.

Figure 16. CAD representation of Rover crawler (*left*) and inside the mock sewer system (*right*).



As mentioned previously, RTABMap uses cameras for loop-closure detection. In simulation, we used a front-facing RGB camera; however, on the physical robot, we used front and rear Boson 640 thermal cameras. Using thermal cameras instead of RGB cameras meant we did not have to add external lighting to the robot. Also, having both front and rear cameras increased our chances of a loop closure being detected. Figure 17 is representative of the views from the front-facing (left) and rear-facing (right) cameras. The thermal cameras generated enough contrast that we were able to teleoperate and transition the robot to different tubes within the mock sewer system.

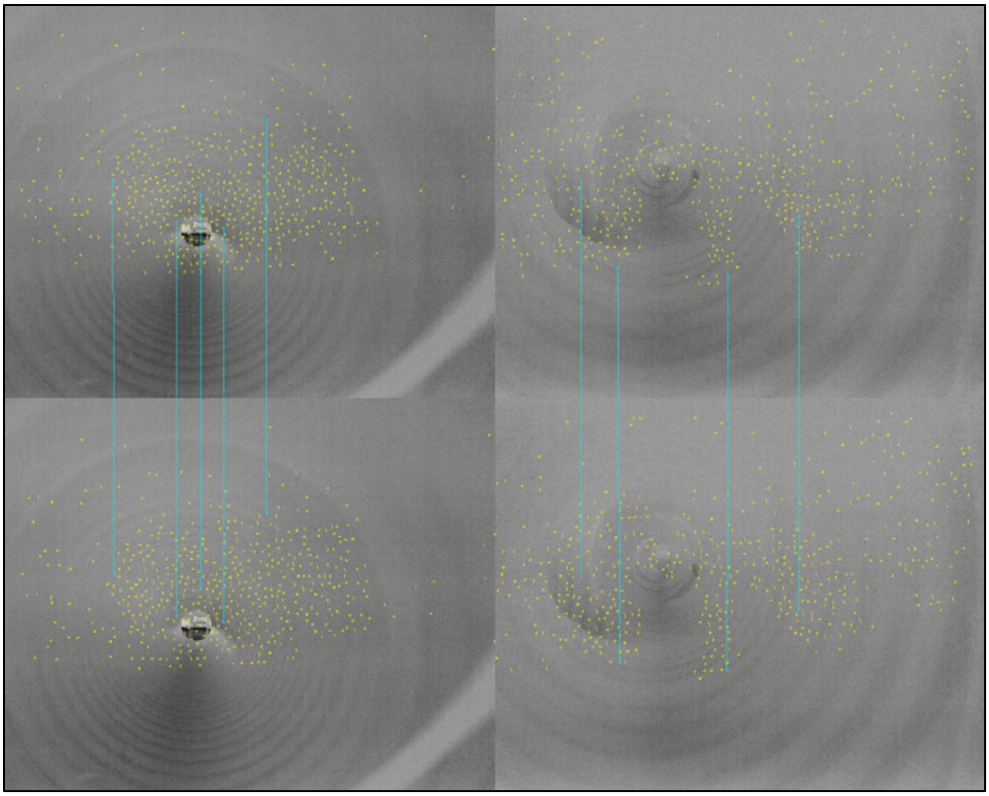
Figure 17. Image from front (*left*) and rear (*right*) thermal cameras while inside the mock sewer system.



RTABMap has a number of methodologies to detect features in images, including, but not limited to, SURF, SIFT, GFTT/ORB (default), and ORB-

Octree. These approaches take an image and isolate particular features. These features are tracked from frame to frame and can be used to determine the robot's pose. Out of that list, we chose to use ORB-Octree because it typically identifies more features than the other approaches. Our thermal camera has a resolution of 640 × 480, which is significantly lower than that of our RGB camera (1,920 × 1,080); thus, for the lower resolution thermal cameras, having more features is preferred for loop-closure detection. Figure 18 shows two images in sequence. Detected features are shown as yellow dots, and features tracked between image sequences are shown as blue lines. For this image sequence, the front camera tracked five features, and the rear camera tracked four features. Also, we restricted feature identification to the upper half of the image. This was done specifically to avoid identifying dynamic features that may have occurred along the bottom of the tube.

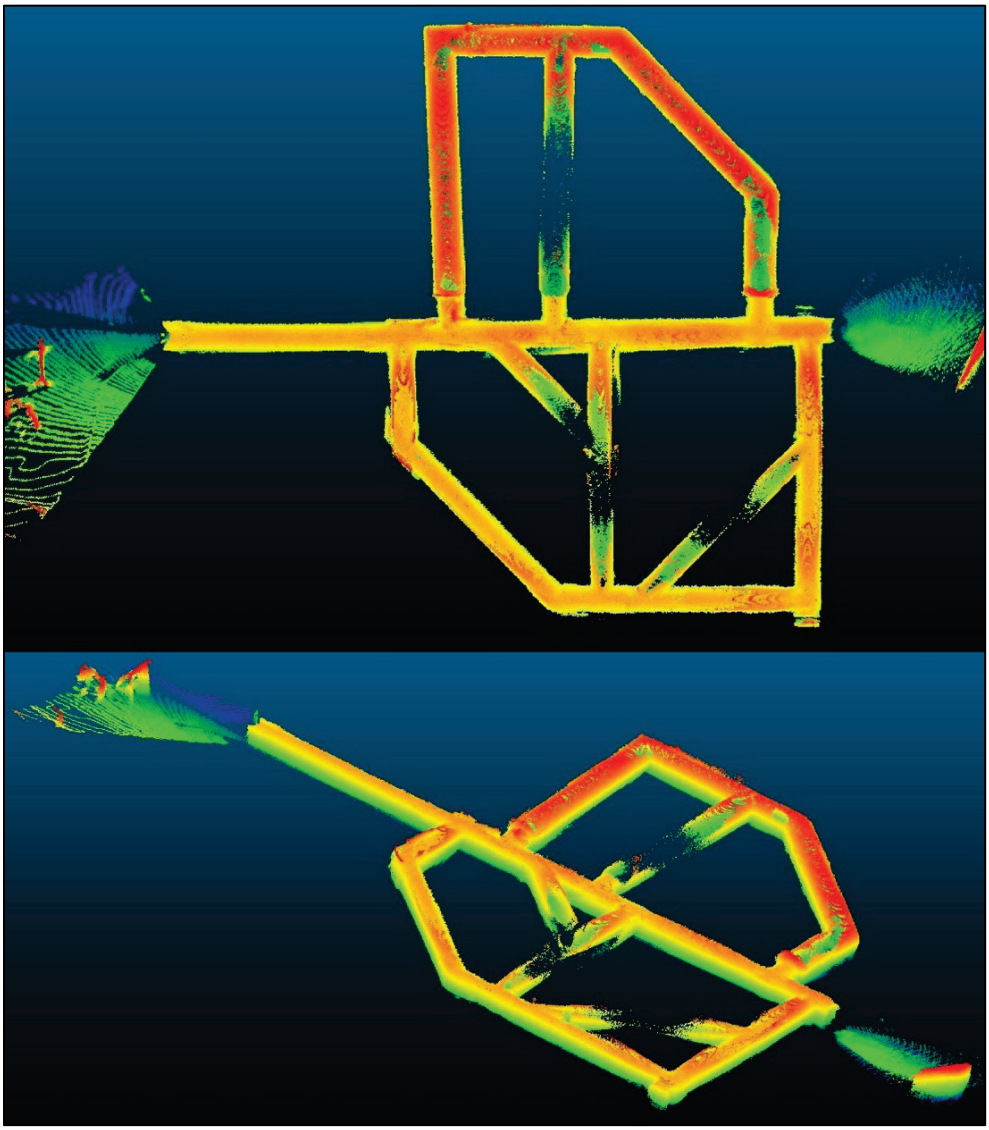
Figure 18. Loop-closure detection with thermal cameras within the mock sewer system.



Due to its length, the full launch file for RTABMap on the physical robot is provided in Appendix K. We have discussed RTABMap at length in previous reports (Glaspell et al. 2020; Christie et al. 2021). In short, we used the `pointcloud_to_depthimage` node to project the lidar points into the camera frame to generate a depth image. This was done for both the front

and rear cameras. We then used the `rgbd_sync` node to synchronize the image from the thermal camera and the depth image generated by the `pointcloud_to_depthimage` node. This synchronized topic was passed to the `rtabmap_ros` node along the odometry topic generated in Section 2.1.1. RTABMap maintains the map frame transform. This setup was used to generate the 3D point cloud shown in Figure 19. When comparing the 3D point cloud from the physical robot (Figure 19) to the point cloud done in simulation (Figure 12), some distortion is evident from the physical robot. To correct this distortion, we will repeat the test with adjusted odometry and SLAM parameters. Overall, the approach was effective in mapping and localizing within the mock sewer system, and while not perfect, it is still quite usable for our intended application.

Figure 19. The 3D point cloud generated by RTABMap with the physical robot.



2.3 Future Work

In the future, we plan to address the following tasks, which are listed in order of increasing complexity:

- Repeat the mock sewer test with a physical robot with modified odom and SLAM parameters.
- Transfer the payload to QinetiQ SPUR.
- Repeat the mock sewer test with QinetiQ SPUR.
- Integrate way point navigation.
- Design and integrate a payload to drop sensors.

For the first task, which is to repeat the mock sewer test with a physical robot using modified odom and SLAM parameters, adjusting the `kf_dist_linear` and `kf_dist_angular` for odometry and the ICP parameters in the RTABMap launch file for SLAM should fix the observed deviation from the physical robot. The goal is for the 3D point cloud from the physical robot to have a root mean square error (RMSE) within 5 cm in reference to the point cloud collected in simulation.

For the second task, transferring the payload to QinetiQ SPUR should be straightforward. The payload was designed to be modular. Even the power connections are similar between the robots.

The main objective for the third task, which is to repeat the mock sewer test with the QinetiQ SPUR, is to ensure that the developed IOP-to-ROS bridge is compatible with our navigation stack. In short, passing the `cmd_velocity` topic from our navigation stack to the IOP bridge should be all that is required. The IOP bridge will translate the `cmd_velocity` topic to the corresponding velocity commands in IOP.

The fourth task, integrating way point navigation, is a bit more complicated. The main issue that we need to address is that, when the robot is asked to return home, it backs up instead of trying to rotate in place to drive forward. This should be accomplished by increasing the weights for the respective navigation tasks.

The final task, which is to design and integrate a payload to drop sensors, is the most involved and requires both design and testing elements. The robot's small footprint and the requirements of the navigation stack make

this task especially difficult. Once the solution is integrated onto the robot, we anticipate testing its efficacy in the mock sewer system.

3 Summary

We explored odometry, SLAM, and localization within the confines of the mock sewer system. Due to the nature of the environment, we explored odometry approaches that did not leverage visual cues or wheel encoders. Both in simulation and on the physical robot, the lidar odometry package `laser_scan_matcher` was used successfully to track the robot's pose within the mock sewer system. We also explored a variety of SLAM approaches, including gmapping, SLAM Toolbox, elevation mapping, and RTABMap. Each approach offers advantages and could be the appropriate choice, depending on the CONOP. In short, gmapping is a 2D approach that worked well with the default parameters. SLAM Toolbox, which requires tuning, offers lifelong mapping, which would be useful for change detection. Elevation mapping incorporates elevation into its 2.5D approach and can be used to set lower limits. The lower limit is useful to prevent the robot from exploring an area where standing water is expected. RTABMap uses cameras for loop-closure detection and can be used to generate 3D point clouds. In simulation, we also explored the AMCL and `i_see_pee` localization approaches when the CONOP required revisiting a location, such as when replacing a damaged sensor. Both methods were successful in simulation. While the `i_see_pee` was designed to be faster, either approach would work well in the mock sewer system with the robot's onboard computer. Finally, we were able to take the results from the simulation and apply them to the physical robot. Specifically, we used RTABMap to generate a 3D point cloud of the mock sewer system. While there were some deviations present compared to the results in simulation, further parameter tuning should remedy the issue.

References

AutoLidarPerception. 2020. "Segmenters Lib." *GitHub*. https://github.com/AutoLidarPerception/segmenters_lib.

CCNY Robotics Lab. 2022a. "Laser Ortho Projector." *GitHub*. https://github.com/CCNYRoboticsLab/scan_tools/tree/ros1/laser_ortho_projector.

CCNY Robotics Lab. 2022b. "Laser Scan Matcher." *GitHub*. https://github.com/CCNYRoboticsLab/scan_tools/tree/ros1/laser_scan_matcher.

Censi, A. 2008. "An ICP Variant Using a Point-to-Line Metric," In *Proceedings, 2008 IEEE International Conference on Robotics and Automation*, 19–23 May, Pasadena, CA. New York: IEEE. https://doi.org/10.1109/ROBOT.2008.4543181.

Charles River Analytics. 2022. "Robot Localization." *GitHub*. https://github.com/cra-ros-pkg/robot_localization/tree/noetic-devel.

Christie, B. A., O. Ennasr, and G. P. Glaspell. 2021. *ROS Integrated Object Detection for SLAM in Unknown, Low-Visibility Environments*. ERDC/GRL TR-21-6. Alexandria, VA: US Army Engineer Research and Development Center, Geospatial Research Laboratory. http://dx.doi.org/10.21079/11681/42385.

Dorezyuk, D. 2019. "I See Pee." *GitHub*. https://github.com/dorezyuk/i_see_pee.

Glaspell, G. P., S. R. Lessard, B. A. Christie, K. Jannak-Huang, N. C. Wilde, W. He, O. Ennasr, et al. 2020. *Optimized Low Size, Weight, Power and Cost (SWaP-C) Payload for Mapping Interiors and Subterranean on an Unmanned Ground Vehicle*. ERDC/GRL TR-20-6. Alexandria, VA: US Army Engineer Research and Development Center, Geospatial Research Laboratory. https://doi.org/10.21079/11681/35878.

Kollmitz, M. 2020. "Gazebo ROS 2D Map Plugin." *GitHub*. https://github.com/marinaKollmitz/gazebo_ros_2Dmap_plugin.

Labbé, M., and F. Michaud. 2019. "RTAB-Map as an Open-Source Lidar and Visual Simultaneous Localization and Mapping Library for Large-Scale and Long-Term Online Operation." *Journal of Field Robotics* 36 (2): 416–446. https://onlinelibrary.wiley.com/doi/abs/10.1002/rob.21831.

Macenski, S. 2023. "SLAM Toolbox." *GitHub*. https://github.com/SteveMacenski/slam_toolbox.

Macenski, S., and I. Jambrecic. 2021. "SLAM Toolbox: SLAM for the Dynamic World." *Journal of Open Source Software* 6 (61): 2783. https://doi.org/10.21105/joss.02783.

Moore, T., and D. Stouch. 2014. "A Generalized Extended Kalman Filter Implementation for the Robot Operating System." In *Intelligent Autonomous Systems 13: Advances in Intelligent Systems and Computing* (Vol. 302), edited by E. Menegatti, N. Michael, K. Berns, and H. Yamaguchi, 335–348. Cham, Switzerland: Springer. https://doi.org/10.1007/978-3-319-08338-4_25.

Office of the Deputy Chief of Staff. 2020. *Army Multi-Domain Intelligence: FY21–22 S and T Focus Areas*. AD1114490. Washington, DC: Department of the Army. https://apps.dtic.mil/sti/pdfs/AD1114489.pdf.

QinetiQ. n.d. "SPUR Next Generation Backable Robot." Accessed May 31, 2023. https://www.qinetiq.com/en/what-we-do/services-and-products/spur-next-generation-backpackable-robot.

ROS Perception. 2020. "SLAM Gmapping." *GitHub*. https://github.com/ros-perception/slam_gmapping.

ROS Perception. 2022. "Pointcloud to Laserscan." *GitHub*. https://github.com/ros-perception/pointcloud_to_laserscan.

ROS Planning. 2023. "AMCL." *GitHub*. https://github.com/ros-planning/navigation/tree/noetic-devel/amcl.

Appendix A: ekf_flipperbot.yaml

```
#Configuation for robot odometry EKF
#
frequency: 30
publish_tf: true
two_d_mode: true

pose0: /pose_with_covariance_stamped
pose0_config: [true,  true,  true,
               false, false, false,
               false, false, false,
               false, false, false,
               false, false, false]
pose0_differential: false
pose0_relative: false
pose0_queue_size: 10

imu0: /imu/data
imu0_config: [false, false, false,
              true,  true,  true,
              false, false, false,
              false, false, false,
              false, false, false]

imu0_nodelay: false
imu0_differential: false
imu0_relative: true
imu0_queue_size: 10

odom_frame: odom
base_link_frame: base_link
world_frame: odom

predict_to_current_time: true
```

Appendix B: gmapping_params.yaml

```
map_update_interval: 2.0
maxUrange: 100.0
sigma: 0.05
kernelSize: 1
lstep: 0.05
astep: 0.05
iterations: 5
lsigma: 0.075
ogain: 3.0
lskip: 0
minimumScore: 50
srr: 0.1
srt: 0.2
str: 0.1
stt: 0.2
linearUpdate: 1.0
angularUpdate: 0.2
temporalUpdate: 0.5
resampleThreshold: 0.5
particles: 100
xmin: -10.0
ymin: -10.0
xmax: 10.0
ymax: 10.0
delta: 0.05
llsamplerange: 0.01
llsamplestep: 0.01
lasamplerange: 0.005
lasamplestep: 0.005
```

Appendix C: mapper_params_onlinesync.yaml

```
# Plugin params
solver_plugin: solver_plugins::CeresSolver
ceres_linear_solver: SPARSE_NORMAL_CHOLESKY
ceres_preconditioner: SCHUR_JACOBI
ceres_trust_strategy: LEVENBERG_MARQUARDT
ceres_dogleg_type: TRADITIONAL_DOGLEG
ceres_loss_function: None

# ROS Parameters
odom_frame: odom
map_frame:  map
base_frame: base_footprint
scan_topic: /pc_2_ls
mode: mapping #localization

debug_logging: false
throttle_scans: 1
transform_publish_period: 0.02 #if 0 never publishes odometry
map_update_interval: 5.0
resolution: 0.05
max_laser_range: 100.0 #for rastering images
minimum_time_interval: 0.5
transform_timeout: 0.2
tf_buffer_duration: 30.
stack_size_to_use: 40000000 #// program needs a larger stack size
    to serialize large maps
enable_interactive_mode: true

# General Parameters
use_scan_matching: true
use_scan_barycenter: true
minimum_travel_distance: 0.5
minimum_travel_heading: 0.5
scan_buffer_size: 10
scan_buffer_maximum_scan_distance: 10
link_match_minimum_response_fine: 0.1
link_scan_maximum_distance: 1.5
loop_search_maximum_distance: 3.0
do_loop_closing: true
loop_match_minimum_chain_size: 10
loop_match_maximum_variance_coarse: 3.0
loop_match_minimum_response_coarse: 0.35
loop_match_minimum_response_fine: 0.45

# Correlation Parameters - Correlation Parameters
correlation_search_space_dimension: 0.5
correlation_search_space_resolution: 0.01
correlation_search_space_smear_deviation: 0.1

# Correlation Parameters - Loop Closure Parameters
loop_search_space_dimension: 8.0
```

```
loop_search_space_resolution: 0.05
loop_search_space_smear_deviation: 0.03

# Scan Matcher Parameters
distance_variance_penalty: 0.5
angle_variance_penalty: 1.0

fine_search_angle_offset: 0.00349
coarse_search_angle_offset: 0.349
coarse_angle_resolution: 0.0349
minimum_angle_penalty: 0.9
minimum_distance_penalty: 0.5
use_response_expansion: true
```

Appendix D: tb3.yaml

```
# Robot.
map_frame_id:                                   map
robot_base_frame_id:                            base_link
robot_pose_with_covariance_topic:               /
    base_link_pose
robot_pose_cache_size:                          200
sensor_frame_id:                                base_link
point_cloud_topic:                              /
   velodyne_points
track_point_frame_id:                           base_link
track_point_x:                                  0.0
track_point_y:                                  0.0
track_point_z:                                  0.0
min_update_rate:                                1.0
time_tolerance:                                 1.0
time_offset_for_point_cloud:                    0.0
sensor_processor/ignore_points_above:           0.8
robot_motion_map_update/covariance_scale:       0.01

# Map.
length_in_x:                                    50.0
length_in_y:                                    50.0
position_x:                                     0.0
position_y:                                     0.0
resolution:                                     0.1
min_variance:                                   0.0001
max_variance:                                   0.05
mahalanobis_distance_threshold:                 2.5
multi_height_noise:                             0.001
surface_normal_positive_axis:                   z
fused_map_publishing_rate:                      1.0
enable_visibility_cleanup:                      false
visibility_cleanup_rate:                        1.0
scanning_duration:                              1.0

# Init submap
initialize_elevation_map:                       true
initialization_method:                          0
length_in_x_init_submap:                        1.0
length_in_y_init_submap:                        1.0
margin_init_submap:                             0.3
init_submap_height_offset:                      0.01
target_frame_init_submap:                       base_link
```

Appendix E: velodyne_HDL-32E.yaml

```
# Velodyne_HDL-32E
# TODO
sensor_processor:
  type: laser
  min_radius: 0.018
  beam_angle: 0.0006
  beam_constant: 0.0015
```

Appendix F: postprocessor_pipeline.yaml

```
postprocessor_pipeline: # set by
    postprocessor_pipeline_name
   # Fill holes in the map with inpainting.
   - name: inpaint
      type: gridMapCv/InpaintFilter
      params:
         input_layer: elevation
         output_layer: elevation_inpainted
         radius: 0.05

   # Compute Surface normals
   - name: surface_normals
      type: gridMapFilters/NormalVectorsFilter
      params:
         input_layer: elevation_inpainted
         output_layers_prefix: normal_vectors_
         radius: 0.1
         normal_vector_positive_axis: z
```

Appendix G: costmap.yaml

```
grid_map_topic: /elevation_mapping/elevation_map

grid_map_visualizations:

  - name: elevation_grid
    type: occupancy_grid
    params:
      layer: elevation
      data_min: -10.0
      data_max: 10.0

  - name: elevation_pc
    type: point_cloud
    params:
      layer: elevation
      data_min: -10.0
      data_max: 10.0
```

Appendix H: RTABMap_sim.launch

```
<?xml version="1.0"?>
<launch>
  <!-- Arguments -->
  <arg name="with_camera" default="true"/>
  <arg name="database_path" default="~/.ros/rtabmap.db"
  />
  <arg name="rtabmap_args" default="-d"/>

  <group ns="rtabmap">
    <node pkg="rtabmap_sync" type="rgbd_sync" name=" rgbd_sync"
  output="screen">
      <remap from="rgb/image" to="/camera/rgb/image_raw
  "/>
      <remap from="depth/image" to="/camera/depth/
  image_raw"/>
      <remap from="rgb/camera_info" to="/camera/rgb/
  camera_info"/>
    </node>

    <node name="rtabmap" pkg="rtabmap_slam" type=" rtabmap" out-
  put="screen" args="$(arg rtabmap_args)">
      <param name="database_path" type="string" value="
  $(arg database_path)"/>
      <param name="frame_id" type="string" value=" base_link"/>
      <param name="subscribe_rgb" type="bool" value=" false"/>
      <param name="subscribe_depth" type="bool" value=" false"/>
      <param name="subscribe_rgbd" type="bool" value=" true"/>
      <param name="subscribe_scan" type="bool" value=" true"/>
      <param name="approx_sync" type="bool" value="true
  "/>
      <!-- inputs -->
      <remap from="scan" to="/pc_2_ls"/>
      <remap from="odom" to="/odom"/>
      <remap from="rgbd_image" to="rgbd_image"/>

      <!-- output -->
      <remap from="grid_map" to="/map"/>

      <!-- RTAB-Maps parameters -->
      <param name="Reg/Strategy" type="string" value="1
  "/>
      <param name="Reg/Force3DoF" type="string" value="
  true"/>
      <param name="GridGlobal/MinSize" type="string"
  value="20"/>
    </node>
  </group>
</launch>
```

Appendix I: amcl.launch

```
<launch>
  <!-- Arguments -->
  <arg name="scan_topic" default="/pc_2_ls"/>
  <arg name="initial_pose_x" default="0.0"/>
  <arg name="initial_pose_y" default="0.0"/>
  <arg name="initial_pose_a" default="0.0"/>

  <node pkg="amcl" type="amcl" name="amcl">
    <param name="initial_pose_x" value="$(arg␣ini-
  tial_pose_x)"/>
    <param name="initial_pose_y" value="$(arg␣ini-
  tial_pose_y)"/>
    <param name="initial_pose_a" value="$(arg␣ini-
  tial_pose_a)"/>
  <param name="gui_publish_rate" value="1.0"/>
  <remap from="scan" to="$(arg␣scan_topic)"/>

    <param name="min_particles" value="100"/>
    <param name="max_particles" value="5000"/>
    <param name="kld_err" value="0.02"/>
    <param name="update_min_d" value="0.20"/>
    <param name="update_min_a" value="0.20"/>
    <param name="resample_interval" value="2"/>
    <param name="transform_tolerance" value="1.0"/>
    <param name="recovery_alpha_slow" value="0.00"/>
    <param name="recovery_alpha_fast" value="0.00"/>
    <param name="laser_max_range" value="100"/>
    <param name="laser_max_beams" value="180"/>
    <param name="laser_z_hit" value="0.95"/>
    <param name="laser_z_short" value="0.1"/>
    <param name="laser_z_max" value="0.05"/>
    <param name="laser_z_rand" value="0.05"/>
    <param name="laser_sigma_hit" value="0.2"/>
    <param name="laser_lambda_short" value="0.1"/>
    <param name="laser_likelihood_max_dist" value="2.0"
/>
    <param name="laser_model_type" value=" likeli-
  hood_field_prob"/>
    <param name="odom_model_type" value="diff"/>
    <param name="odom_alpha1" value="0.1"/>
    <param name="odom_alpha2" value="0.1"/>
    <param name="odom_alpha3" value="0.1"/>
    <param name="odom_alpha4" value="0.1"/>
    <param name="odom_frame_id" value="odom"/>
    <param name="base_frame_id" value="base_link"/>
  </node>
</launch>
```

Appendix J: i_see_pee.yaml

```yaml
map:
  topic: "/map"
  k: 5      # min: 1, max: 10
  radius: 5 # min: 0, max: 10

odom:
  map_frame: "map"
  odom_frame: "odom"
  base_frame: "base_link"

scan:
  topic: "/pc_2_ls"

icp:
  t_norm: 0.01 # min: 0.01
  r_norm: 0.01 # min: 0.01
  max_iter: 100 # min: 1, max: 100
  stride: 3 # min: 1, max: 100
```

Appendix K: RTABMap_rl.launch

```
<launch>
  <arg name="gui_cfg" default="~/.ros/rtabmap_gui.ini"
   />
  <arg name="launch_prefix" default=""/>
  <arg name="output" default="screen"/>
  <arg name="node_start_delay" default="15.0" />

  <group ns="rtabmap">
    <node pkg="nodelet" type="nodelet" name="
   pointcloud_to_depthimage_0" args="standalone␣
   rtabmap_ros/pointcloud_to_depthimage" output="screen ">
      <remap from="camera_info" to="/flir_boson_f/ camera_info"/>
      <remap from="cloud" to="/outlier_removal/output"/>
      <remap from="image_raw" to="image_raw_0"/>
      <remap from="image" to="image_0"/>
      <param name="fixed_frame_id" type="string"
   value="RLodom"/>
      <param name="fill_holes_size" type="int" value=
   "5"/>
      <param name="decimation" type="int" value="0"/>
      <param name="fill_iterations" type="int" value= "2"/>
  </node>

  <node pkg="nodelet" type="nodelet" name="
  rgbd_sync_0" args="standalone␣rtabmap_ros/rgbd_sync"
   output="screen">
      <remap from="rgb/image" to="/flir_boson_f/ image_rect"/>
      <remap from="depth/image" to="image_raw_0"/>
      <remap from="rgb/camera_info" to="/flir_boson_f
  /camera_info"/>
      <remap from="rgbd_image" to="rgbd_image0"/>
      <param name="approx_sync" value="true"/>
  </node>

  <node pkg="nodelet" type="nodelet" name="
  pointcloud_to_depthimage_1" args="standalone␣
  rtabmap_ros/pointcloud_to_depthimage" output="screen ">
      <remap from="camera_info" to="/flir_boson_r/ camera_info"/>
      <remap from="cloud" to="/outlier_removal/output"/>
      <remap from="image_raw" to="image_raw_1"/>
      <remap from="image" to="image_1"/>
      <param name="fixed_frame_id" type="string"
  value="RLodom"/>
      <param name="fill_holes_size" type="int" value="5"/>
      <param name="decimation" type="int" value="0"/>
      <param name="fill_iterations" type="int" value="2"/>
  </node>

  <node pkg="nodelet" type="nodelet" name="
rgbd_sync_1" args="standalone␣rtabmap_ros/rgbd_sync"
```

```
  output="screen">
      <remap from="rgb/image" to="/flir_boson_r/ image_rect"/>
      <remap from="depth/image" to="image_raw_1"/>
      <remap from="rgb/camera_info" to="/flir_boson_r
  /camera_info"/>
      <remap from="rgbd_image" to="rgbd_image1"/>
      <param name="approx_sync" value="true"/>
  </node>

  <node name="rtabmap" pkg="rtabmap_ros" type="
 rtabmap" output="screen" args="--delete_db_on_start"
   launch-prefix="bash␣-c␣'sleep␣$(arg␣
  node_start_delay);␣$0␣$@'␣">
      <param name="frame_id" type="string" value=" base_link"/>
      <param name="odom_frame_id" type="string" value="" />
      <param name="subscribe_depth" type="bool" value="false"/>
      <param name="subscribe_scan_cloud" type="bool"
value="true"/>
      <param name="subscribe_scan" type="bool" value= "false"/>
      <param name="subscribe_odom_info" type="bool"
value="false"/>
      <param name="subscribe_rgb" type="bool" value=" false"/>
      <param name="subscribe_rgbd" type="bool" value= "true"/>
      <param name="rgbd_cameras" type="int" value="2"/>
      <remap from="rgbd_image0" to="rgbd_image0"/>
      <remap from="rgbd_image1" to="rgbd_image1"/>
      <remap from="odom" to="/odometry/filtered"/>
      <remap from="scan_cloud" to="/velodyne_points"
/>
      <remap from="scan" to="/pc_2_ls"/>
      <param name="queue_size" type="int" value="50"
/>
      <!-- RTAB-Maps parameters -->
      <param name="RGBD/AngularUpdate" type="string"
value="0.1"/>
      <param name="RGBD/LinearUpdate" type="string" value=
"0.1"/>
      <param name="RGBD/NeighborLinkRefining" type=" string"
value="true"/>
      <param name="RGBD/ProximityBySpace" type=" string"
value="true"/>
      <param name="RGBD/ProximityByTime" type="string "
value="false"/>
      <param name="RGBD/ProximityPathMaxNeighbors" type="string"
value="30"/>
      <param name="RGBD/OptimizeFromGraphEnd" type=" string"
value="false"/>
      <param name="RGBD/OptimizeMaxError" type=" string"
value="3"/>
      <param name="RGBD/LocalRadius" type="string" value="10"/>
      <param name="Reg/Strategy" type="string" value=
"1"/>
      <param name="Reg/Force3DoF" type="string" value
="false"/>
```

```
      <param name="Grid/FromDepth" type="string" value="false"/>
      <param name="Mem/STMSize" type="string" value="
30"/>
      <param name="Vis/MinInliers" type="string"
value="20"/>
      <param name="Kp/DetectorStrategy" type="string" value="10"/>
      <param name="Vis/FeatureType" type="string" value="10"/>
      <param name="Vis/EstimationType" type="string" value="0"/>
      <param name="Optimizer/Strategy" type="string" value="2"/>
      <param name="Kp/RoiRatios" type="string" value=
"0.0␣0.0␣0.0␣0.4"/>
      <param name="Grid/MaxObstacleHeight" type=" string"
value="0.51"/>
      <param name="Grid/3D" type="string" value="true"/>
      <param name="Grid/RayTracing" type="string"value="true"/>
      <param name="Grid/RangeMax" type="string" value="10"/>
      <param name="Grid/RangeMin" type="string" value="0.4"/>
      <param name="Grid/NormalsSegmentation" type=" string"
value="true"/>
      <param name="Grid/MinClusterSize" type="int" value="10" />
      <param name="Grid/ClusterRadius" type="double" value="1" />
      <param name="Grid/MaxGroundAngle" type="double" value="45.0"
/>
      <param name="Grid/NormalK" type="int" value="20" />
      <param name="GridGlobal/AltitudeDelta" type="string"
value="0.0"/>
      <param name="Rtabmap/DetectionRate" type=" string"
value="1"/>

      <!-- ICP parameters -->
      <param name="Icp/VoxelSize" type="string" value="0.1"/>
      <param name="Icp/PointToPlaneK" type="string" value="20"/>
      <param name="Icp/PointToPlaneRadius" type=" string"
value="0"/>
      <param name="Icp/PointToPlane" type="string" value="true"/>
      <param name="Icp/Iterations" type="string"
value="10"/>
      <param name="Icp/Epsilon" type="string" value=" 0.001"/>
      <param name="Icp/MaxTranslation" type="string" value="3"/>
      <param name="Icp/MaxCorrespondenceDistance" type="string"
value="0.3"/>
      <param name="Icp/Strategy" type="string" value= "true"/>
      <param name="Icp/OutlierRatio" type="string" value="0.7"/>
      <param name="Icp/CorrespondenceRatio" type=" string"
value="0.2"/>
      <param name="OdomF2M/BundleAdjustment" type=" string"
value="3"/>
      <param name="Icp/RangeMax" type="string" value= "40"/>
      <param name="Icp/PointToPlaneGroundNormalsUp"
type="string" value="0.3"/>
   </node>
 </group>
</launch>
```

Abbreviations

AI	Artificial intelligence
AMCL	Adaptive Monte Carlo Localization
CAD	Computer-aided design
CONOP	Concept of operations
ERDC	US Army Engineer Research and Development Center
ICP	Iterative closest point
IMU	Inertial measurement unit
IOP	Interoperability profile
LSM	Laser scan matcher
ML	Machine learning
PGM	Portable gray map
RAS	Robotics and autonomous systems
RGB	Red-green-blue
RGB-D	Red-green-blue and depth
RMSE	Root mean square error
ROS	Robot Operating System
SLAM	Simultaneous localization and mapping
SNIP	Sewer Network Interrogation Program
SubT	Subterranean
tf	Transform
TIC/Ms	Toxic industrial chemicals and materials
UET	Understanding the Environment as a Threat

UGV	Unmanned ground vehicle

REPORT DOCUMENTATION PAGE

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED	
		START DATE	END DATE
September 2023	Final Technical Report (TR)	FY22	FY23

4. TITLE AND SUBTITLE

Mapping and Localization Within a Mock Sewer System

5a. CONTRACT NUMBER	5b. GRANT NUMBER	5c. PROGRAM ELEMENT
		0603463A
5d. PROJECT NUMBER	**5e. TASK NUMBER**	**5f. WORK UNIT NUMBER**
AR6		

6. AUTHOR(S)

Brandon Dodd, Osama Ennasr, Amir Naser, Chuck Ellison, Jason Ray, Garry Glaspell, and Anton Netchaev

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)	8. PERFORMING ORGANIZATION REPORT NUMBER
See reverse.	ERDC TR-23-16

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)	11. SPONSOR/MONITOR'S REPORT NUMBER(S)
Headquarters, US Army Corps of Engineers Washington, DC 20314-1000		

12. DISTRIBUTION/AVAILABILITY STATEMENT

Distribution Statement A. Approved for public release: distribution is unlimited.

13. SUPPLEMENTARY NOTES

Program Element No. 0603463A, Project No. AR6

14. ABSTRACT

Herein, we explored a robot's ability to localize and map, both in simulation and on a physical robot, within a mock sewer system. Mapping and localization techniques were first developed and tested in simulation and were then transitioned to the actual robot for additional physical testing. Several odometry and simultaneous localization and mapping (SLAM) techniques, including gmapping, SLAM toolbox, elevation mapping, and RTABMap, were evaluated for this particular environment. The results of the odometry and the various SLAM approaches are discussed in detail.

15. SUBJECT TERMS

Autonomous robots; Mobile robots; Sewerage--Mapping; Sewerage--Navigation

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES
a. REPORT	b. ABSTRACT	C. THIS PAGE		
Unclassified	Unclassified	Unclassified	SAR	56

19a. NAME OF RESPONSIBLE PERSON	19b. TELEPHONE NUMBER (include area code)

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) (concluded)

US Army Engineer Research and Development Center (ERDC)
Geospatial Research Laboratory (GRL)
7701 Telegraph Road
Alexandria, VA 22315-3864

US Army Engineer Research and Development Center (ERDC)
Information Technology Laboratory (ITL)
3909 Halls Ferry Road
Vicksburg, MS 39180-6199