ERDC TR-23-13

US Army Corps of Engineers®
Engineer Research and Development Center

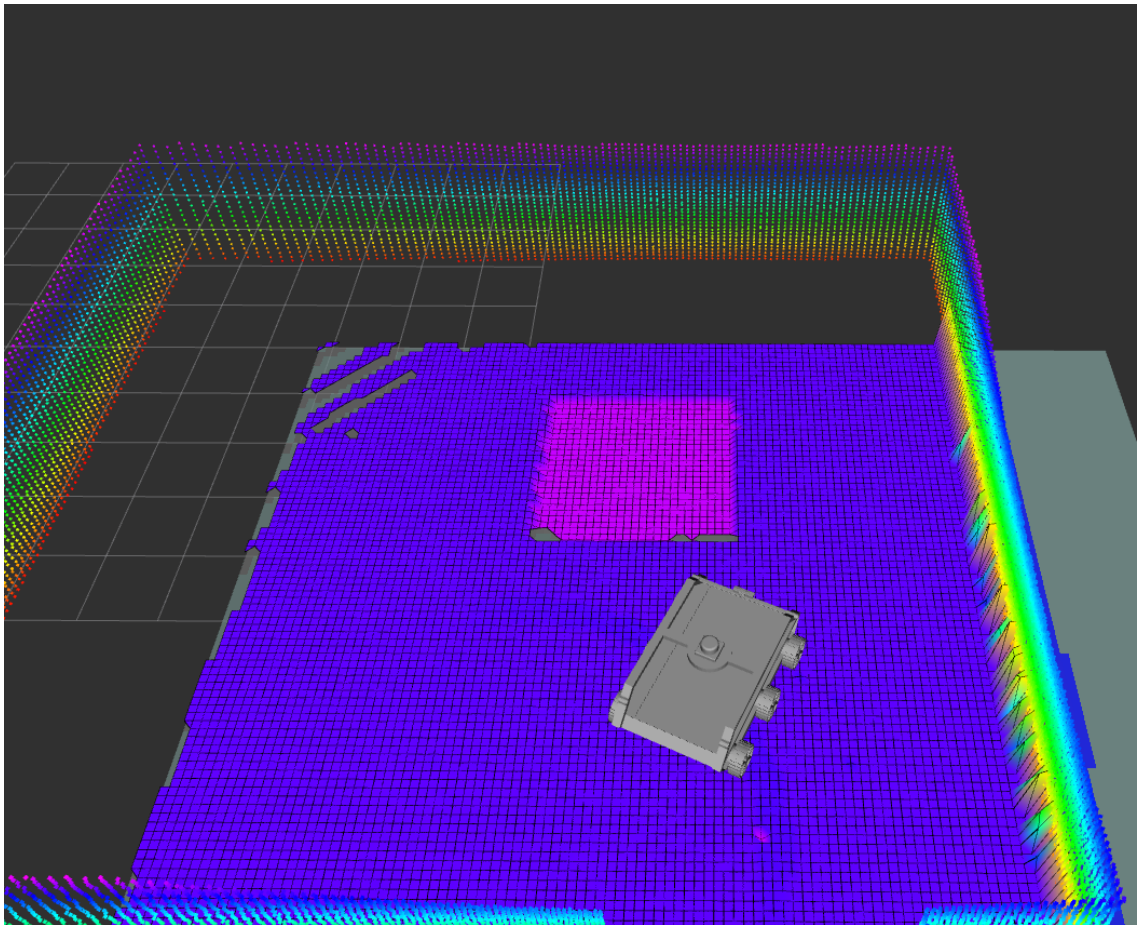
ERDC
ENGINEER RESEARCH & DEVELOPMENT CENTER

UGV-Localization in 3D and Path Planning (U-L3AP)

Unmanned Ground Vehicle (UGV) Full Coverage Planning with Negative Obstacles

Jin-Kyu Lee, Amir Naser, Osama Ennasr, Ahmet Soylemezoglu, and Garry Glaspell

August 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-23-13
August 2023

Unmanned Ground Vehicle (UGV) Full Coverage Planning with Negative Obstacles

Jin-Kyu Lee, Amir Naser, Osama Ennasr, and Garry Glaspell

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

Ahmet Soylemezoglu

US Army Engineer Research and Development Center (ERDC)
Construction Engineering Research Laboratory (CERL)
2902 Newmark Drive
Champaign, IL 61822

Final Technical Report (TR)

Prepared for US Army Engineer Research and Development Center (ERDC)
3909 Halls Ferry Road
Vicksburg, MS 39180-6199

Under FLEX-4 funding

Abstract

We explored approaches that offer full coverage path planning while simultaneously avoiding negative obstacles. These approaches are specific to unmanned ground vehicles (UGVs), which need to constantly interact with a traversable ground surface. We tested multiple potential solutions in simulation, and the results are presented herein. Full coverage path planner (FCPP) approaches were evaluated based on their ability to discretize their paths, use waypoints effectively, and be easily integrated with our current robot platform. For negative obstacles, we explored approaches that will integrate with our current navigation stack. The preferred solution will allow for teleoperation, waypoint navigation, and full autonomy while avoiding positive and negative obstacles.

Contents

Figures

Preface

This study was conducted for the US Army Engineer Research and Development Center (ERDC) of the US Army Corps of Engineers (USACE). It was funded by ERDC under FLEX-4.

The work was performed by the Data Representation Branch, Topography Imagery and Geospatial Research Division of the ERDC Geospatial Research Laboratory (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 GRL. The deputy director of ERDC-GRL was Ms. Valerie L. Carney, and the director was Mr. David R. Hibner. Work was also performed by the Warfighter Engineering Branch, Operational Science and Engineering Division of the ERDC Construction Engineering Research Laboratory (CERL). At the time of publication, Mr. Jeff Burkhalter was branch chief, Dr. George Calfas was division chief, and Mr. Jim Allen was the technical direction of CERL. The deputy director of ERDC-CERL was Ms. Michelle Hanson, and the director was Dr. Andrew Nelson.

The authors would like to acknowledge the following individuals for their contributions to this project: Dr. Anton Netchaev, Mr. Steven Bunkley, and Mr. Charles Ellison.

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

This page intentionally left blank.

1 Introduction

1.1 Background

In September of 2021, we demonstrated our robotic platform in the Maneuver Support, Sustainment, and Protection Integration Experiments (MSSPIX) 2022 hosted by the Army's Maneuver Support and Sustainment Capability Development Integration Directorates. Three soldiers were trained to use the robotic platform over a three-day span. The Soldiers learned to use manual navigation, semi-autonomous waypoint navigation, and autonomous exploration to map tunnels and building interiors. The team successfully demonstrated a platform-agnostic unmanned ground vehicle (UGV) edge compute (millisecond low latency decisions with onboard hardware) and sensor payload for surveying and mapping interior structures (including subterranean environments). Specifically, the UGV was capable of three modes of operation without Soldiers entering a potentially hazardous environment: teleoperation, waypoint navigation, or autonomous mapping. Based on that experience and on Soldier feedback, the team identified two aspects of navigation that needed improvement.

Initially, we assumed with limited battery capacity (i.e., approximately 3 hours), the main objective for autonomy was to cover as much ground as possible in the shortest amount of time. Thus, we adopted an approach based on frontier exploration. A frontier, in this context, is defined as a boundary between an area that the robot has explored and an area that has yet to be explored. Priority was given to the mathematically largest frontiers. In context, this would provide an overall floorplan of a building in relatively short order, but smaller rooms would not be explored thoroughly. Thus, if the concept of operation (CONOP) also involved identifying an object of interest, this approach could potentially miss the object of interest, especially if it was located in one of the smaller rooms. As a result, the team investigated full coverage path planners (FCPPs) that can be used to explore rooms thoroughly. One aspect of this report addresses using full coverage planners.

Another issue that arose during the MSSPIX 22 demo was negative obstacles. While the robot was quite capable of identifying and avoiding positive obstacles, negative obstacles, such as descending stairs or a hole in the

floor, were beyond its initial capability. Because these negative obstacles were known before the demonstration, we were able to use virtual obstacles to keep the robot from exploring these areas. However, an approach must be developed to deal with negative obstacles, especially if prior knowledge is unavailable. Thus, the second aspect of this report focuses on identifying negative obstacles.

1.2 Objectives

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, we feel 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). By incorporating full coverage planning and the ability to detect negative obstacles, we met the objective of creating a more efficient autonomous systems.

1.3 Approach

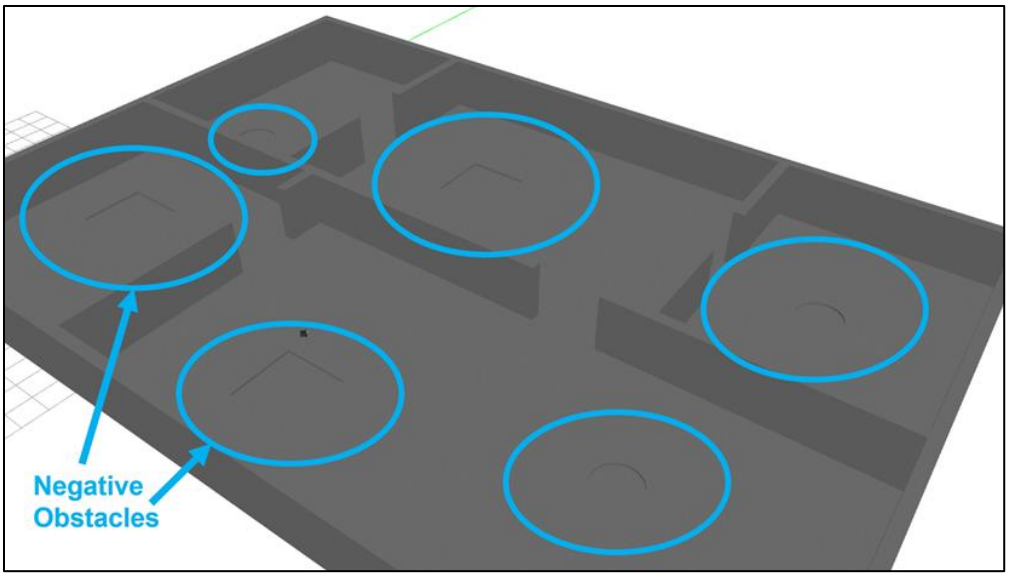
Our approach involved running full coverage planners and negative obstacle detection methods in simulation. To test both scenarios, we used the Obstacle World from Clearpath Robotics (2021). The Obstacle World is a virtually defined indoor world that serves as a confined area for the full coverage planners, and the nonplanar floor allows for the simulated testing of negative obstacle detection. Figure 1 shows the world as it appears in the simulation environment. The negative obstacles that appear throughout the world are labeled. The full launch file for loading the virtual environment and robot can be found in 0, Section Appendix A. Several parameters included in the launch file can be used to tweak the environment, including the starting position of the robot and the world scale. To launch the world, we used the following node: Here a node is defined as a process that performs computation. The node to launch the virtual environment is as follows:

```
<node name="agriculture_world_spawner" pkg="gazebo_ros"
    type="spawn_model" args="-urdf -model obstacle_geom
    -param obstacle_geom -x 0 -y 0 -z 0 -Y 0" />
```

The node to spawn the robot is as follows:

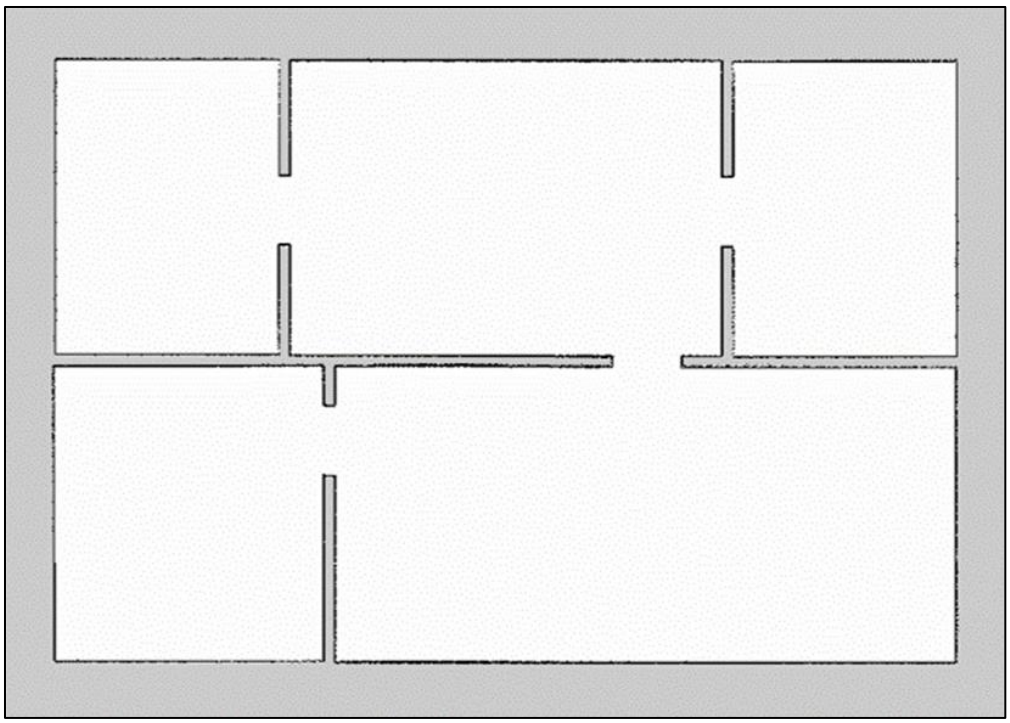
```
<node name="spawn_urdf" pkg="gazebo_ros" type="
    spawn_model" args="-urdf␣-model␣turtlebot3␣-
    x␣$(arg␣x)␣-y␣$(arg␣y)␣-z␣$(arg z)␣-param␣ro-
    bot_description"
/>
```

Figure 1. Clearpath Robotics Obstacle World.



Our goal was to use the full coverage planner to thoroughly explore each room while simultaneously using the negative obstacle detection node to keep the robot from getting stuck. The full coverage planner requires an occupancy grid to plan the route. Figure 2 shows the occupancy grid used for path planning. Because the occupancy grid is typically trinary in nature, space is typically labeled as occupied (black), free (white), and unknown (gray). As a result, negative obstacles are not identifiable, and the planned paths will traverse the nonplanar floor geometry. The negative obstacles are approximately the same size as the wheel diameter, which means that if the robot falls in, it will be unlikely to escape; thus, it will be unable to complete its mission.

Figure 2. A 2D occupancy grid of the Obstacle World.

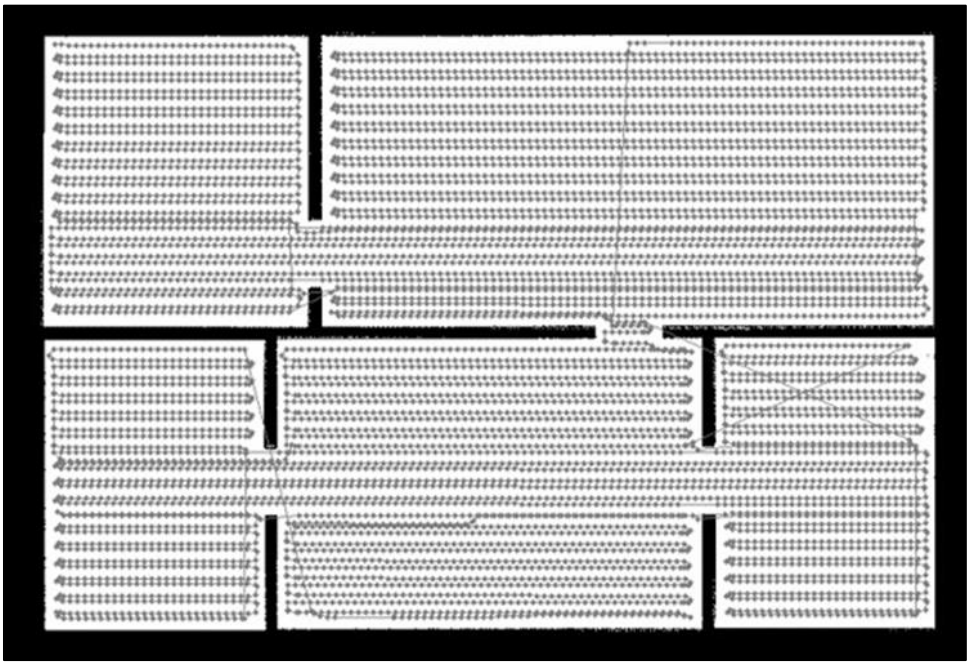


2 Full Coverage Planning

2.1 IPA Coverage Planning

The first package we looked at for full coverage planning was called `ipa_coverage_planning` (Fraunhofer IPA 2016). The IPA coverage planner contains several full coverage algorithms, including boustrophedon, grid-based traveling salesman problem (TSP), neural-network-based, grid-Based local energy minimization, contour line, convex sensor placement, and Voronoi random field (Bormann et al. 2016). Each of these algorithms have different weights for path lengths and rotations, which result in various patterns of coverage. Out of that list, we chose to investigate the boustrophedon, grid-based local energy minimization, and Voronoi random field algorithms based on open source availability. Each of these algorithms was applied to the 2D occupancy grid shown in Figure 2. The boustrophedon algorithm is shown in Figure 3. While the boustrophedon algorithm fully explores the entire region, it treats the occupancy grid as a whole rather than as individual rooms. Specifically, paths planned along the horizon are favored, and multiple trips across the entire building are traversed before an individual room is explored.

Figure 3. Full coverage boustrophedon algorithm.



The full coverage grid-based local energy minimization algorithm is shown in Figure 4. Compared to the boustrophedon algorithm, the grid-based

local energy minimization algorithm appears to isolate patterns to individual rooms before exploring new areas. Even the large room is divided into two sections. However, each path is backtracked, which is redundant and unnecessary. In regard to the object detection scenario, however, this methodology for exploration is preferable to the boustrophedon algorithm because the rooms are isolated. Room isolation is favorable when using multiple robots for exploration.

Figure 4. Full coverage grid-based local energy minimization algorithm.

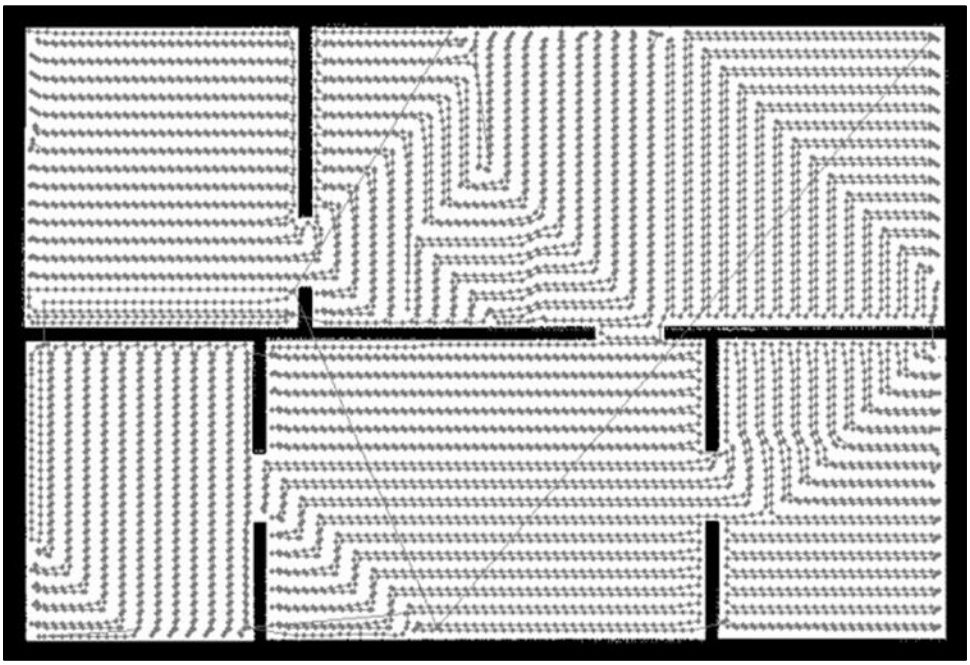
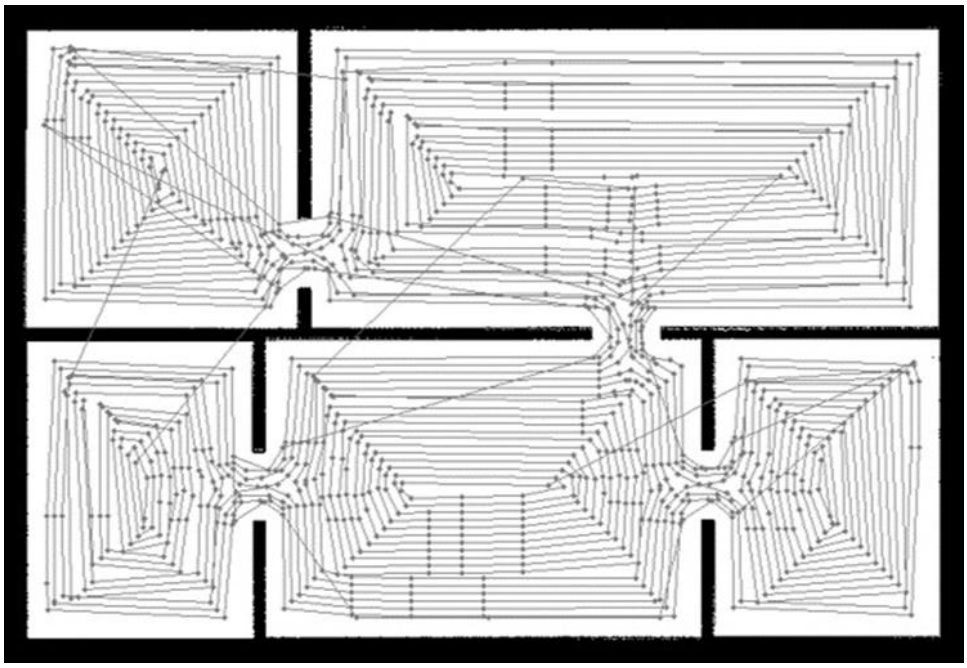


Figure 5 shows the planned path using the Voronoi random field algorithm. Similar to the grid-based local energy minimization algorithm, this approach also seems to confine exploration to individual rooms before expanding to new areas. Compared to the previous two algorithms, this approach also appears to have significantly fewer waypoints and does not backtrack along the same path. However, when transitioning between areas, paths are planned through walls more frequently than in the aforementioned approaches.

Of the three approaches we tested, the Voronoi random field algorithm appeared to be the best, specifically in using waypoints and discretizing the map. However, all the approaches in this section seemed to disregard walls when transitioning to different areas. While outside the scope of this work, adding a feasibility check would mitigate this issue. While the availability of multiple full coverage algorithms is attractive, a fair amount of work is

required to optimize search patterns to prevent transitions through walls. Also, it appears that this approach does not integrate with `move_base`. For `ipa_coverage_planning` to support `move_base`, the source code would have to be extensively modified, which is outside the scope of this work. The `move_base` package uses the navigation stack to move the robot to desired positions and is the preferred method of locomotion for our physical robot. Thus, we looked for alternative full-coverage approaches.

Figure 5. Full coverage Voronoi random field algorithm.



2.2 Full Coverage Path Planner (FCPP)

The other full coverage approach that we investigated was FCPP (Brodskiy et al. 2004). This approach uses the backtracking spiral algorithm to plan its path (Gonzalez et al. 2005). The steps required to clone and compile FCPP are provided here:

```
Cd catkin_workspace/src
git clone https://github.com/nobleo/
   full_coverage_path_planner.git
cd ../
catkin_make
```

FCPP integrates with move base flex (Magazino et al. 2018) as a global path planner plugin. The following code demonstrates how to set the global planner in move base flex to use the backtracking spiral

algorithm. We discussed `move_base_flex` at length in our previous reports (Glaspell et al. 2020; Christie et al. 2021a).

```
planners:
  - name: 'SpiralSTC'
  type: 'full_coverage_path_planner/SpiralSTC'
```

The default local planner for FCPP is `tracking_pid`, which can be set with the code that follows. The tracking pid local planner is designed to implicitly follow the path defined by the global path planner. As a result, it is incapable of detouring around negative obstacles, such as a hole in the floor. In testing, when the pid tracking local planner encountered a negative obstacle, it just stopped in front of the obstacle.

```
controllers:
  - name: 'tracking_pid'
  type: 'tracking_pid/TrackingPidLocalPlanner'
```

The detection of negative obstacles is discussed at length in Section 3. However, we require a local planner that can plan routes around both positive and negative obstacles. Two common path planners are elastic band (Building-Wide Intelligence Project 2012) and timed elastic band (TU Dortmund 2016). We have discussed both of these planners in previous reports (Christie et al. 2021a; Glaspell et al. 2020). Typically, we use elastic band in simulation and timed elastic band on the physical robot. This is due, in part, to timed elastic band's ability to avoid both static and dynamic obstacles that the physical robot would encounter in the real world. Dynamic obstacles are less of an issue in simulation because most virtual environments are static in nature. The code for setting the elastic band local planner follows. The complete YAML file that sets the elastic band parameters is provided in 0, Section A.2.

```
controllers:
  - name: 'EBandPlannerROS'
  type: 'eband_local_planner/EBandPlannerROS'
```

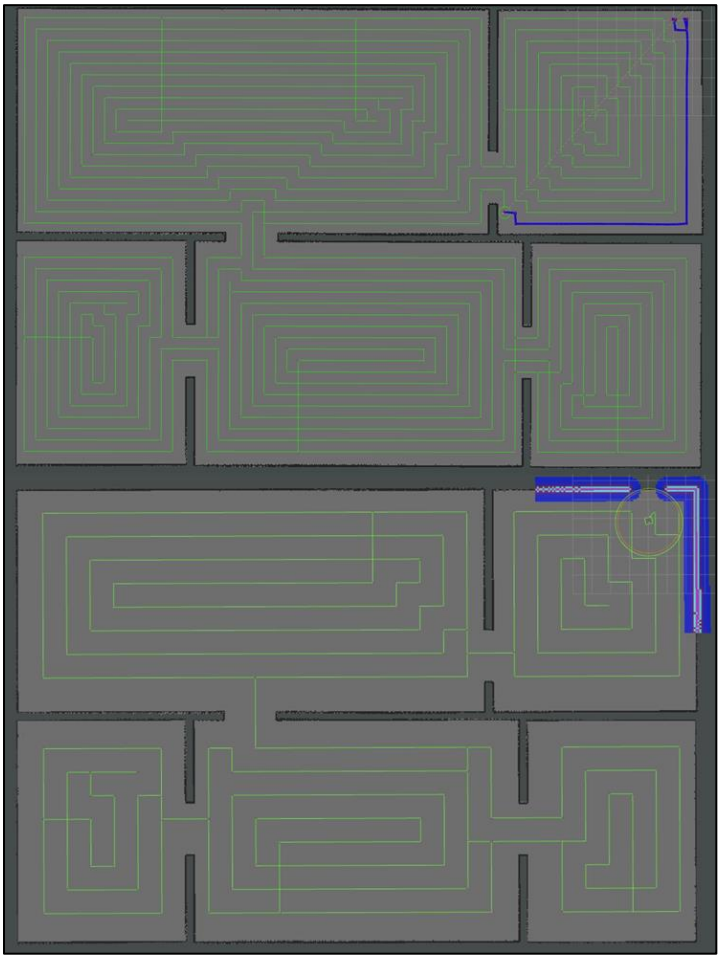
To load the 2D occupancy grid from Figure 2, we used the node from the grid server. The node to launch the grid server follows. We used the arg declaration map to pass the location of the 2D occupancy grid to the grid server node.

```
<arg name="map" default="/home/garry/Downloads/ tb3_obsta-
    cle_map.yaml"/>
```

```
<node name="grid_server" pkg="map_server"
    type=" map_server" args="$(arg map)">
     <param name="frame_id" value="map"/>
</node>
```

ROS uses messages to pass information between nodes. The FCPP package subscribes to `tf` and `tf_Static` and publishes `coverage_grid` and `coverage_progress`. It also uses the parameter `tool_radius` to discretize the provided occupancy grid for full coverage planning. Figure 6 (top) shows an example of the default `tool_radius` value of 0.3 m, and as a result, many paths are planned per room. For our testing, we set the value of `tool_radius` to 0.65 m. The results of modifying the `tool_radius` parameter are shown in the bottom of Figure 6. This new value provided a minimum of two paths through each negative obstacle. While this value was appropriate for testing the efficacy of our negative obstacle detection approaches, in a real-world scenario (to conserve battery), a value of 1.0 m or higher is probably sufficient when looking for objects of interest.

Figure 6. Full coverage backtracking spiral algorithm.



All subsequent tests in this report used the FCPP global planner and elastic band local planner. Specifically, when compared to the IPA coverage planner, none of the FCPP transitions between rooms bisected any walls. At some point, it may be of interest to integrate the various coverage planners provided in the IPA coverage planner in the FCPP framework, but that is outside the scope of this report. In short, the FCPP planner achieved our goal of full coverage exploration. As a result, the sections that follow focus on identifying and avoiding negative obstacles.

3 Negative Obstacles

We programmed our robots to use three methods of navigation: teleoperation, waypoints, or full autonomy. For teleoperation, we require an efficient method to visualize negative obstacles so the operator can avoid them. For waypoint and full autonomy, the robot needs its own method to identify and path plan around the negative obstacles. Negative obstacles are defined here as holes or cliffs that exist below the ground plane.

3.1 Grid Map

The first package we looked at to identify negative obstacles was `grid_map` (ANYbotics 2016). The `grid_map` package was developed to create 2.5D grid maps (Fankhauser and Hutter 2016). The `grid_map` package depends on eigen. Eigen can be installed with the command `sudo apt-get install libeigen3-dev`. The `grid_map` package can be installed from a repository using the command `sudo apt-get install ros-noetic-grid-map` or can be compiled from the source with the following commands:

```
cd catkin_ws/src
git clone https://github.com/anybotics/grid_map.git
cd ../
catkin_make
```

The `grid_map` package uses a layered approach to generate the 2.5D grid maps. We initially considered using the elevation layer to identify negative obstacles. To accomplish this, we leveraged grid map's `image_to_gridmap` node as a proof of concept. This node converts a 2D image to a 2.5D occupancy grid. The image used is shown in the top left of Figure 7. This image was loaded in the `image_to_gridmap` node using the `image_publisher.py` script provided here:

```
<node pkg="grid_map_demos" type="image_publisher.py"
 name="image_publisher" output="screen">
  <param name="image_path" value="$(terrian.png" />
  <param name="topic" value="~image" />
</node>
```

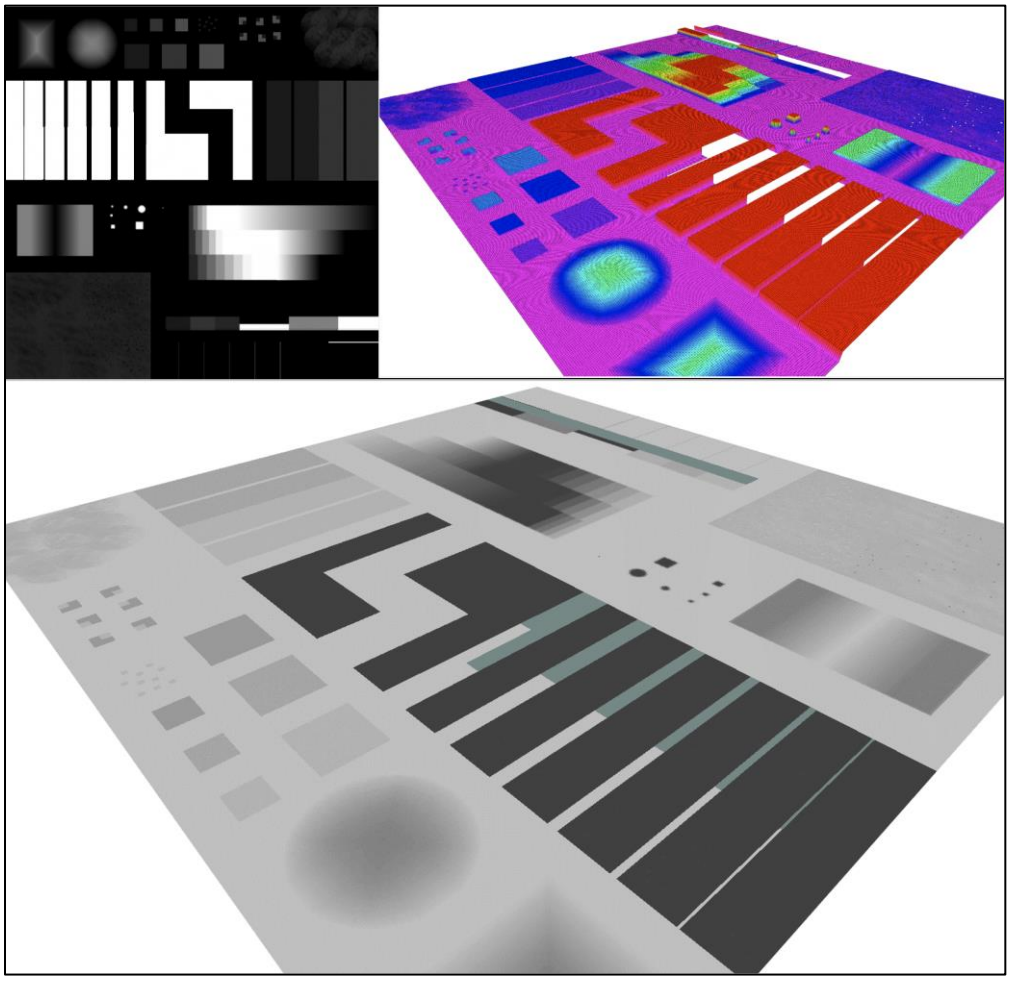
3.1.1 Elevation Layer

With the image loaded, we used the following node to generate the 2.5D elevation grid map shown in the top-right portion of Figure 7. Note the `image_to_gridmap` node interprets lighter colors to be higher in elevation

and darker colors to be lower in elevation. The elevation layer does well in interpreting various features, such as ramps and stairs. Thus, it should be possible to use the 2.5D elevation map as a way for an operator to quickly identify negative obstacles as well.

```
    <node pkg="grid_map_demos" type="
image_to_gridmap_demo" name="image_to_gridmap_demo"
output="screen" />
```

Figure 7. Converting a 2D image to 2.5D grid map and occupancy grid using the elevation layer. (*Top left* image reproduced, with permission, from Fankhauser 2017.)



Robots typically use an occupancy grid to identify and avoid obstacles. Robots also use occupancy grids for path planning. Therefore, we used the `grid_map_visualization` node to convert our 2.5D grip map to an occupancy grid. The resulting 2.5D occupancy grid is shown at the bottom of Figure 7. The general assumption made when using the elevation layer to generate 2.5D occupancy grids is that lower elevation is less costly for the robot to traverse. Higher elevation incurs more cost and is avoided.

```
<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_2.yaml" />"
  </node
```

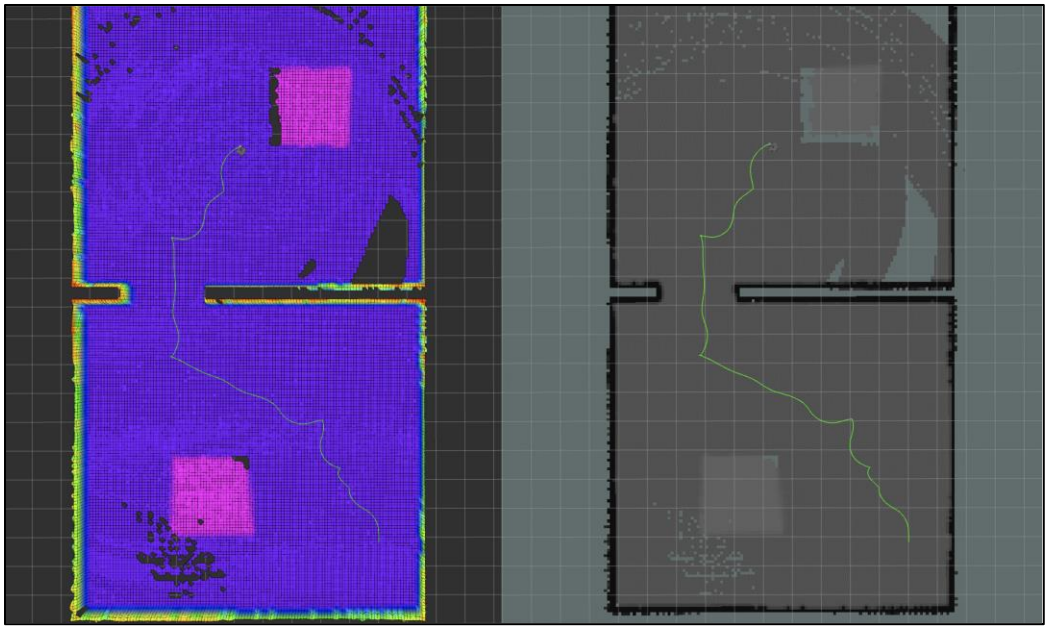
For completeness, the referenced `costmap_2.yaml` file follows.

```
grid_map_topic: /grid_map_filter_demo/filtered_map

grid_map_visualizations:
   -  name: elevation_grid
      type: occupancy_grid
      params:
         layer: elevation
         data_min: -0.6
         data_max: 0.6
   -  name: traversability_grid
      type: occupancy_grid
      params:
         layer: traversability
         data_min: 0.6 #original value -0.6
         data_max: -0.6 #original value 0.6
```

Armed with the ability to generate 2.5D grid maps and occupancy grids, we were ready to test this approach on the virtual world shown in Figure 1. For testing, we installed the package `elevation_mapping` (ANYbotics 2019). Elevation mapping, built around the `grid_map` package, was designed to use depth sensors to generate 2.5D grid maps (Fankhauser et al. 2014; Fankhauser et al. 2018). In our use case, we used a 3D lidar as the input source. The elevation grid map generated in the Obstacle World is shown in the left image of Figure 8. The elevation grid map does a good job of identifying negative obstacles (pink squares), and an operator could easily avoid them when teleoperating the robot. However, issues arise when converting the 2.5D grid map to an occupancy grid, as shown in the right image of Figure 8. With an elevation-based approach, lower elevation is assigned less cost. As a result, the negative obstacle is lighter gray (i.e., lower cost), and the ground is darker gray (i.e., higher cost). If the negative obstacle was significantly lower in elevation, the cost attributed to the ground would be too high for the robot to even move. While effective for teleoperation, this approach prevented us from using the waypoint or full autonomy modes of operation. As a result, we returned to the `grid_map` package and investigated some of the other negative obstacle layers.

Figure 8. Elevation 2.5D grid map and occupancy grid identifying negative obstacles.



3.1.2 Traversability Layer

In addition to the elevation layer, grid maps also include slope, roughness, and edge detection layers. We applied each of these layers to the terrain image shown in Figure 7.

The slope layer was generated from the code that follows. When applied to the terrain image, the slope grid map can be seen in Figure 9*A*.

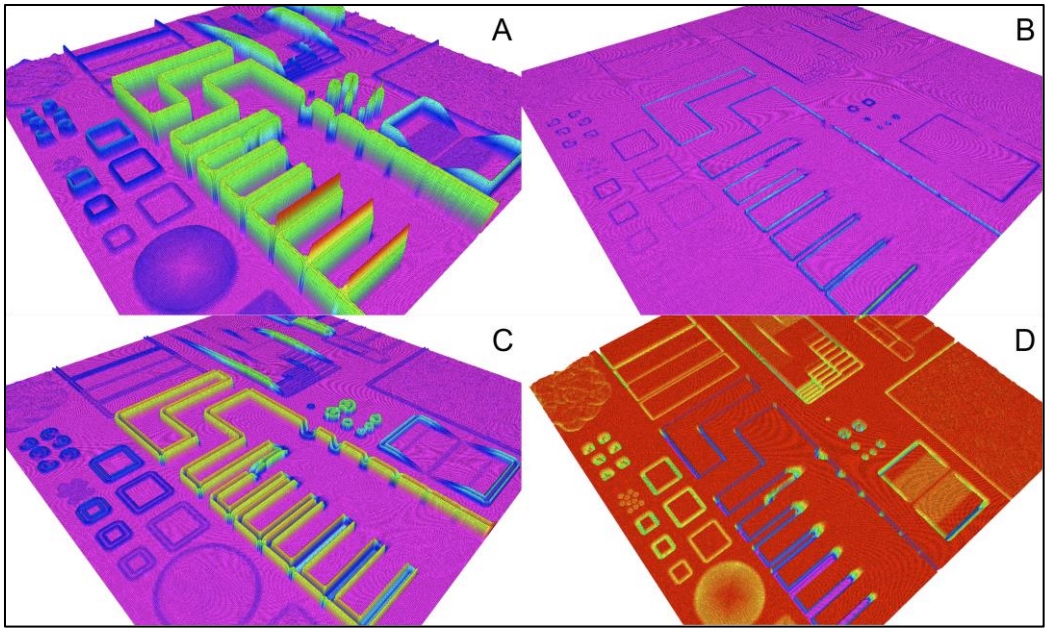
```
- name: slope
  type: gridMapFilters/MathExpressionFilter
  params:
    output_layer: slope
    expression: acos(normal_vectors_z)
```

The roughness layer was generated from the code that follows, and the roughness grid map is shown in Figure 9*B*.

```
- name: roughness
  type: gridMapFilters/MathExpressionFilter
  params:
    output_layer: roughness

    expression: abs(elevation_inpainted - elevation_smooth
```

Figure 9. The 2.5D slope (*A*), roughness (*B*), edge detection (*C*), and traversability (*D*) grid maps.



The edge detection layer was generated from the code that follows, and the edge detection grid map is shown in Figure 9*C*.

```
- name: edge_detection
  type: gridMapFilters/
 SlidingWindowMathExpressionFilter
  params:
     input_layer: elevation_inpainted
     output_layer: edges
     expression: 'sumOfFinites
 ([0,-1,0;-1,5,-1;0,-1,0].*elevation_inpainted)'
     compute_empty_cells: false
     edge_handling: mean
     window_size: 3
```

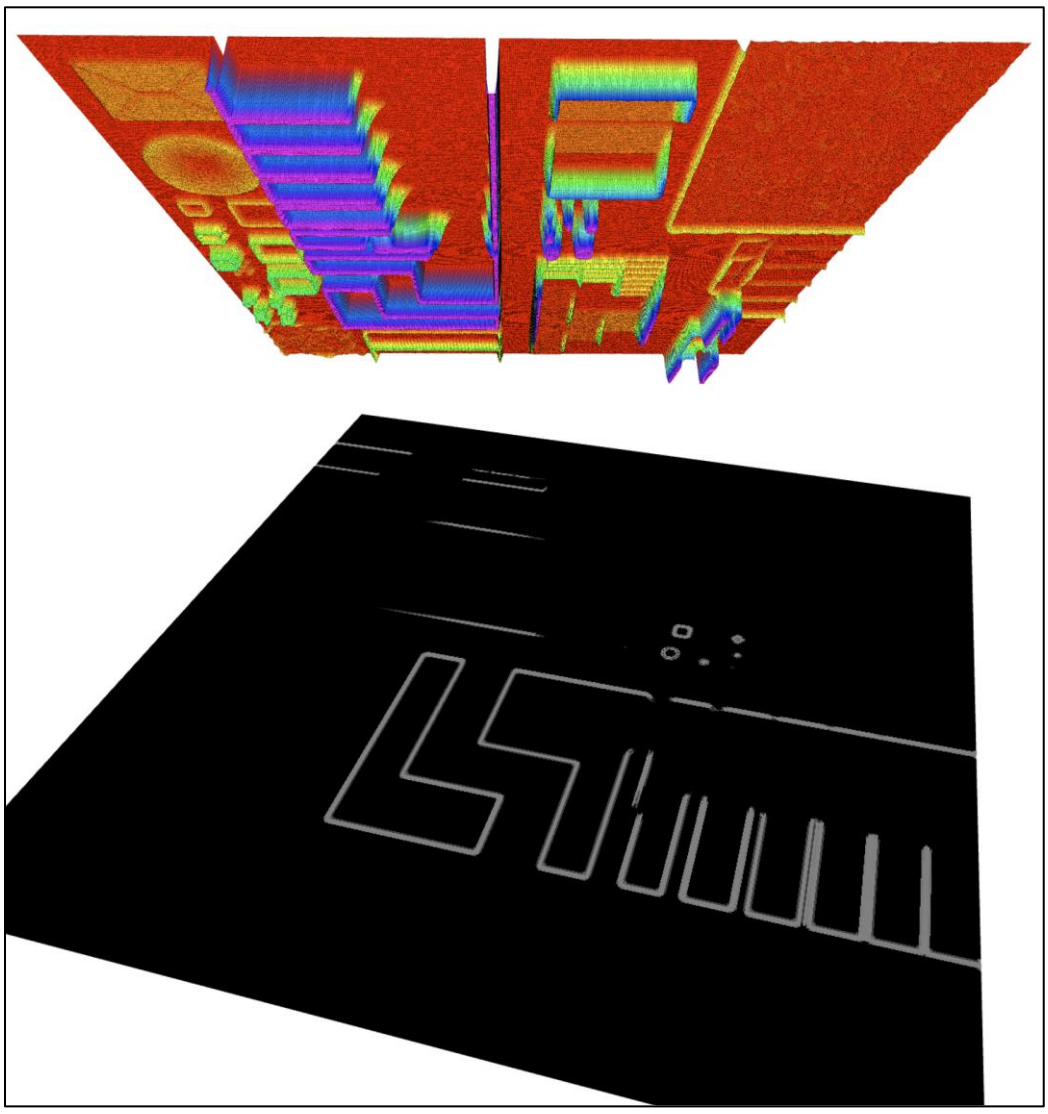
We set the traversability layer to be the average of the slope, roughness, and edge detection layers. The traversability layer was generated from the code that follows, and the traversability grid map is shown in Figure 9*D*.

```
- name: traversability
  type: gridMapFilters/MathExpressionFilter
  params:
     output_layer: traversability
     expression: ((slope + roughness + edges) / 3)
```

If we use the default values of −0.6 for `data_min` and 0.6 for `data_max` in the aforementioned `costmap_2.yaml` file, all the traversability grid map

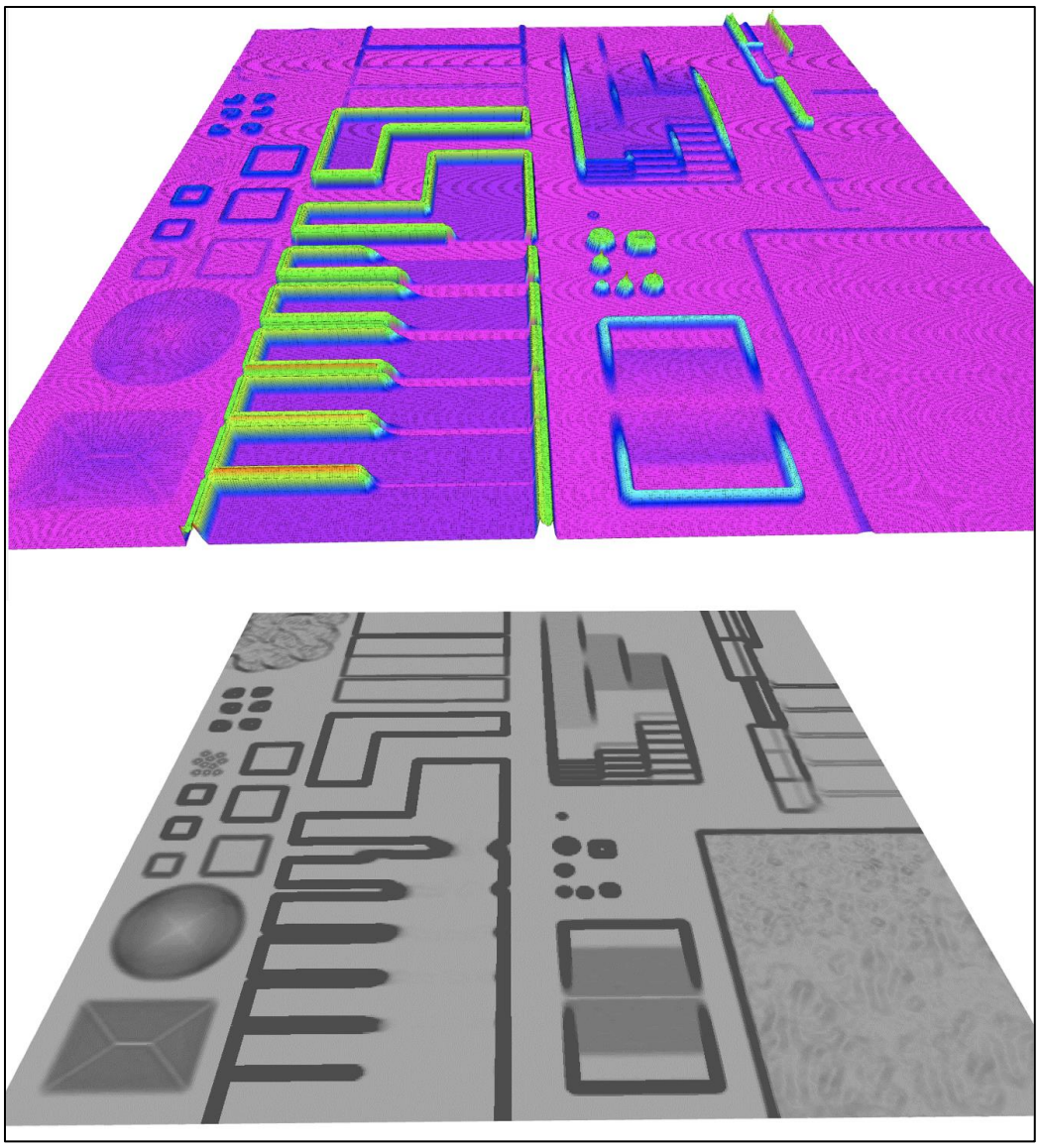
features are negative, as shown in the top image of Figure 10. Because all the features are negative, the resulting 2.5D cost map labels the ground as an obstacle, and the robot is unable to move. This is shown in the bottom image of Figure 10.

Figure 10. Inverted 2.5D traversability and occupancy grid.



The simple fix is to swap the `data_min` and `data_max` values as reflected in the aforementioned `costmap_2.yaml`. Swapping these values results in the 2.5D traversability grid map shown in the top image of Figure 11. With positive traversability map features, the resulting 2.5D occupancy grid does not label the floor as an obstacle, and the robot is able to move freely.

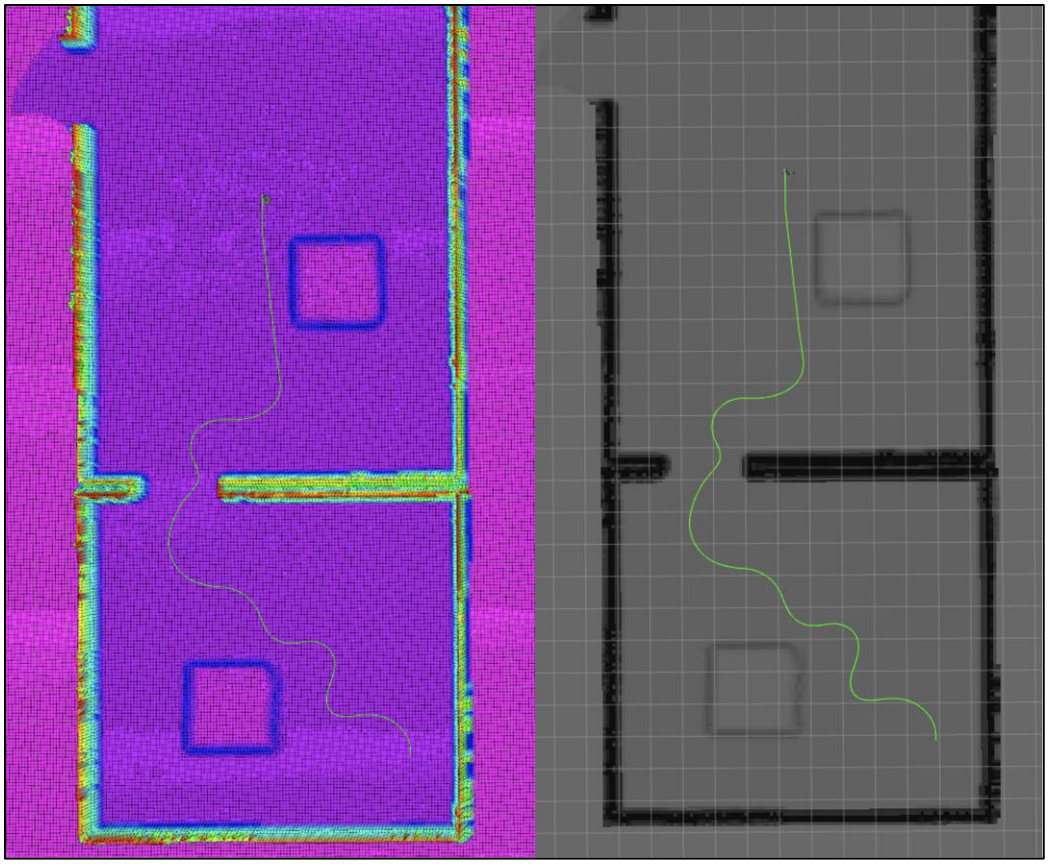
Figure 11. The 2.5D traversability and occupancy grid.



The primary differences between the 2.5D occupancy grid created from the traversability layer (Figure 11) and the 2.5D occupancy grid created from the elevation layer (Figure 7) are that a majority of the traversability occupancy grid can be explored by the robot (light gray) and that the features that could cause issues are outlined and marked with higher cost (dark gray). The next step was to test the traversability layer on the virtual world shown in Figure 1. The results can be seen in Figure 12. The left image in Figure 12 is the 2.5D traversability grid map. It is important to note that the negative obstacles are outlined (with a blue square), and an operator could use the traversability grid map to avoid them while teleoperating the robot. Moreover, the 2.5D traversability occupancy grid is shown in the

right image of Figure 12. In contrast to the 2.5D occupancy grid created from the elevation grid map, the 2.5D occupancy grid created from the traversability layer does not inflate the cost of the ground, and the robot is free to explore the map. As a result, we could use the traversability 2.5D occupancy grid for both waypoint and fully autonomous navigation. Overall, the traversability approach effectively identified negative obstacles and was compatible with the various navigation methods we typically use. Future tests will involve implementing this approach on a physical robot.

Figure 12. Traversability 2.5D grid map and occupancy grid identifying negative obstacles.



3.2 Cliff Detector

Another potential negative obstacle that we could encounter is a cliff. The term *cliff* is used rather loosely in this context and, in an urban environment, could take the form of a loading dock or a downward set of stairs. In regard to the robot, a cliff is a negative obstacle that is difficult to identify with ray tracing. The cliff detector approach we investigated can be found on GitHub (JohnTGZ 2022a). This approach assumes a downward facing 2D lidar. Based on the `lidar_height` and `lidar_pitch` parameters, the

assumed distance to the ground is calculated. The parameter `cliff_threshold_constant` can be used to add a buffer to the calculated ground distance, and this summed value (i.e., ground plus buffer) is added to a lookup table. The lookup table contains values the sensor would compare against. The readings from the lidar are compared to the look up table, and if the range is longer than the value in the lookup table or returns an infinite value, an obstacle is added to the cost map. However, with our current robot payload, we used a horizontal 3D lidar, and not a downward facing 2D lidar. Thus, we had to modify the cliff detector package to meet our needs. The first step was to use the `velodyne_laserscan_node` to select a single ring of our multibeam 3D lidar and to output that ring as a laser scan topic (`velodyne_scan`). The node to select a single ring follows. In this example, we specifically selected the first ring to identify potential cliffs.

```
<arg name="ring" default="0" />
<arg name="resolution" default="0.007" />
<node name="velodyne_laserscan_node" pkg=" velodyne_laserscan"
 type="velodyne_laserscan_node">
  <param name="ring" value="$(arg ring)"/>
  <param name="resolution" value="$(arg_resolution)"/>
  <remap from="velodyne_points" to="/velodyne_points"/>
  <remap from="scan" to="/velodyne_scan" />
</node>
```

The results of using this node are shown in the top image of Figure 13. The white points are the full lidar point cloud, and the red ring is the laser scan topic (`velodyne_scan`) from the `velodyne_laserscan_node`. The next step was to pass the `velodyne_scan` topic from the `velodyne_laserscan_node` to the `laser_filters` node. The node is shown here and serves two purposes. First, it acts as an angular filter, and second, it labels lidar values as infinite when they exceed an upper threshold. The referenced YAML file is provided in 0, Section A.3.

```
<arg name="cliff_scan_topic_raw" default="/ velodyne_scan"/>
<arg name="cliff_scan_topic_filtered" default=" scan_cliff_fil-
 tered"/>
<arg name="cliff_scan_topic_output" default=" scan_cliff"/>
<arg name="cliff_frame_id" default="cliff_lidar"/>
<node pkg="laser_filters" type="
 scan_to_scan_filter_chain" name="scan_cliff_filter"
 output="screen" >
   <remap from="scan" to="$(arg_cliff_scan_topic_raw)"/>
    <remap from="scan_filtered" to="$(arg_       _
 cliff_scan_topic_filtered)" />
   <rosparam command="load" file="/home/garry/
 Downloads/tb3/neg_obs/config/scan_cliff_filter_tb3. yaml" />
</node>
```

The output of the `laser_filters` node is the topic `scan_cliff_filtered`. However, the pattern for `scan_cliff_filtered` is hard coded into the `cliff_detector.cpp` file found on GitHub (JohnTGZ 2022b). The resulting pattern is shown in the bottom image of Figure 13. Again the red V-shaped pattern is based off a downward pointing 2D lidar. However, for our use case, we need it to be a circle. Thus, we modified the aforementioned line with the following code.

```
thresh_laser_scan.push_back(middle_laser_length/ cos( ( i *
    lidar_resolution_) * DEG2RAD)*cos( (i * lidar_resolution_) *
    DEG2RAD) + cliff_thresh_constant_);
```

The results of modifying the code are shown in the top image of Figure 14. Specifically, instead of a V pattern, we now have a circle, shown in green, to compare our lidar values against, shown in orange. Specifically, the green circle is the topic `scan_cliff_filtered`, and the orange circle is the laser scan topic `velodyne_scan`. The red line, which denotes an obstacle, is where the live lidar values (orange) exceed the value from the lookup table (green) and is derived from the `cliff_detector_node`. The code for the `cliff_detector_node` is provided here, and the referenced YAML file is provided in 0, Section A.4.

```
<node pkg="cliff_detector" type="cliff_detector_node"
  name="cliff_detector" output="screen" >
   <remap from="scan" to="$(arg␣cliff_scan_topic_filtered)" />
   <remap from="scan_out" to="$(arg cliff_scan_topic_output)" />
   <rosparam command="load" file="/home/garry/ Down-
 loads/tb3/neg_obs/config/ cliff_detector_params_tb3.yaml" />
</node>
```

Figure 13. Velodyne laser scan node (*top*) and default laser filters node (*bottom*).

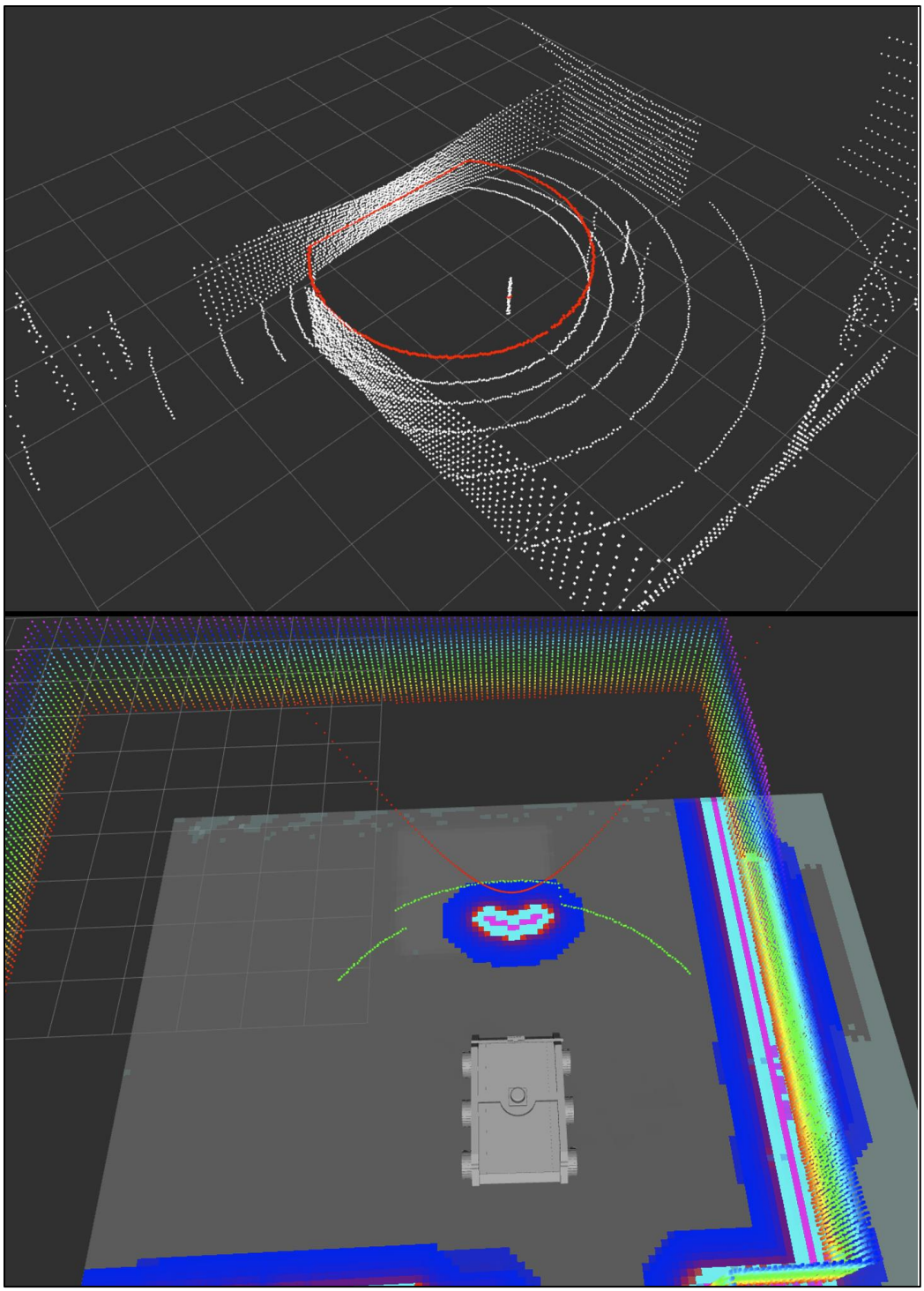
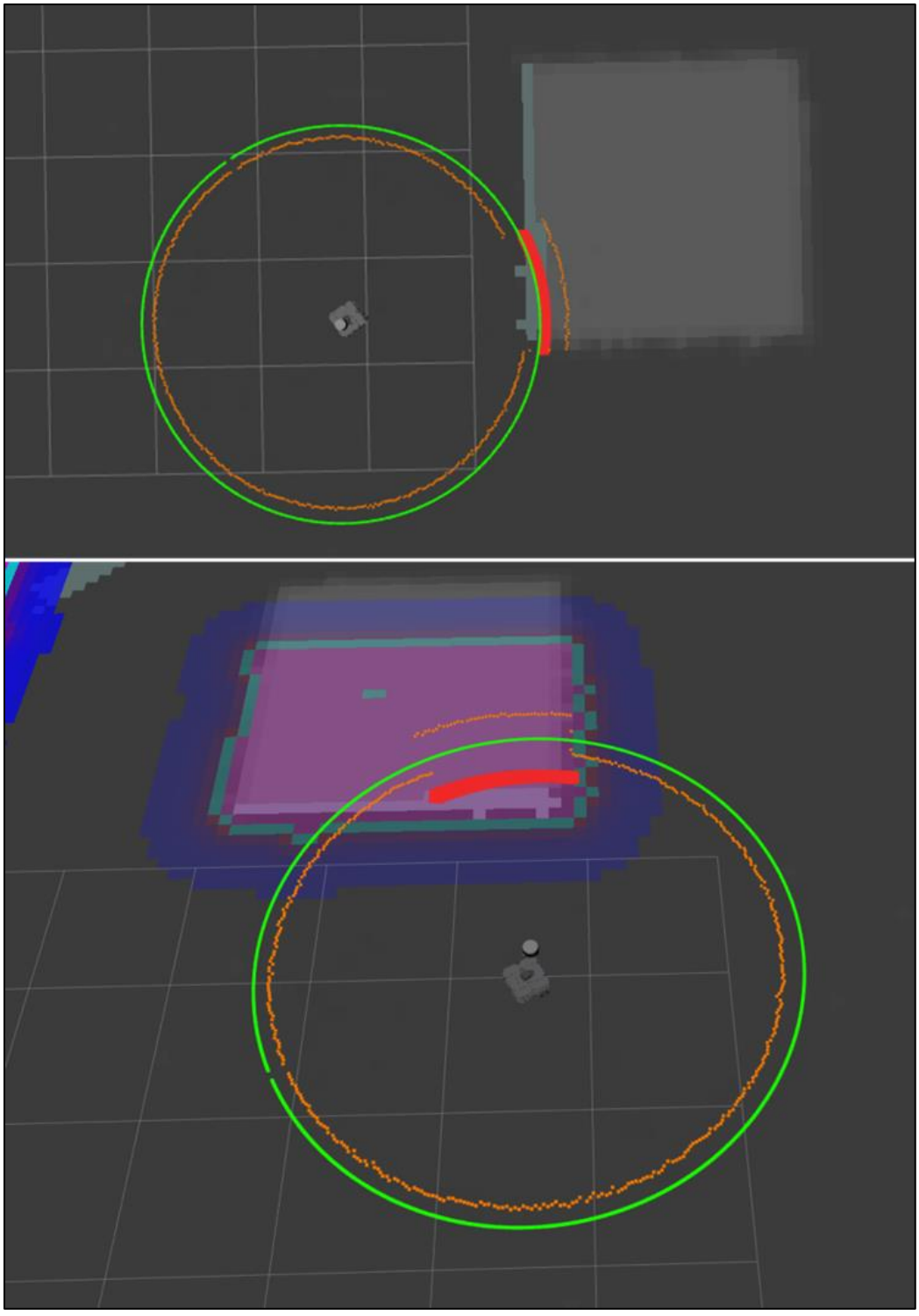


Figure 14. Modified laser filters node detecting (*top*) and marking (*bottom*) a negative obstacle.



The output of the `cliff_detector_node` is the laser topic `scan_cliff`, and we can pass this topic to `move_base_flex` 12. We discussed `move_base_flex` at length in our previous reports (Glaspell et al. 2020; Christie et al.

2021a). In short, to include the `scan_cliff` topic into `move_base_flex`, we add the following plugins to the `local_costmap.yaml` file.

```
plugins:
 -  {name: obstacle_layer,  type: "costmap_2d:: ObstacleLayer"}
 -  {name: neg_obstacle_layer,  type: "costmap_2d:: Obsta-
    cleLayer"}
 -  {name: inflation_layer, type: "costmap_2d:: InflationLayer"}
```

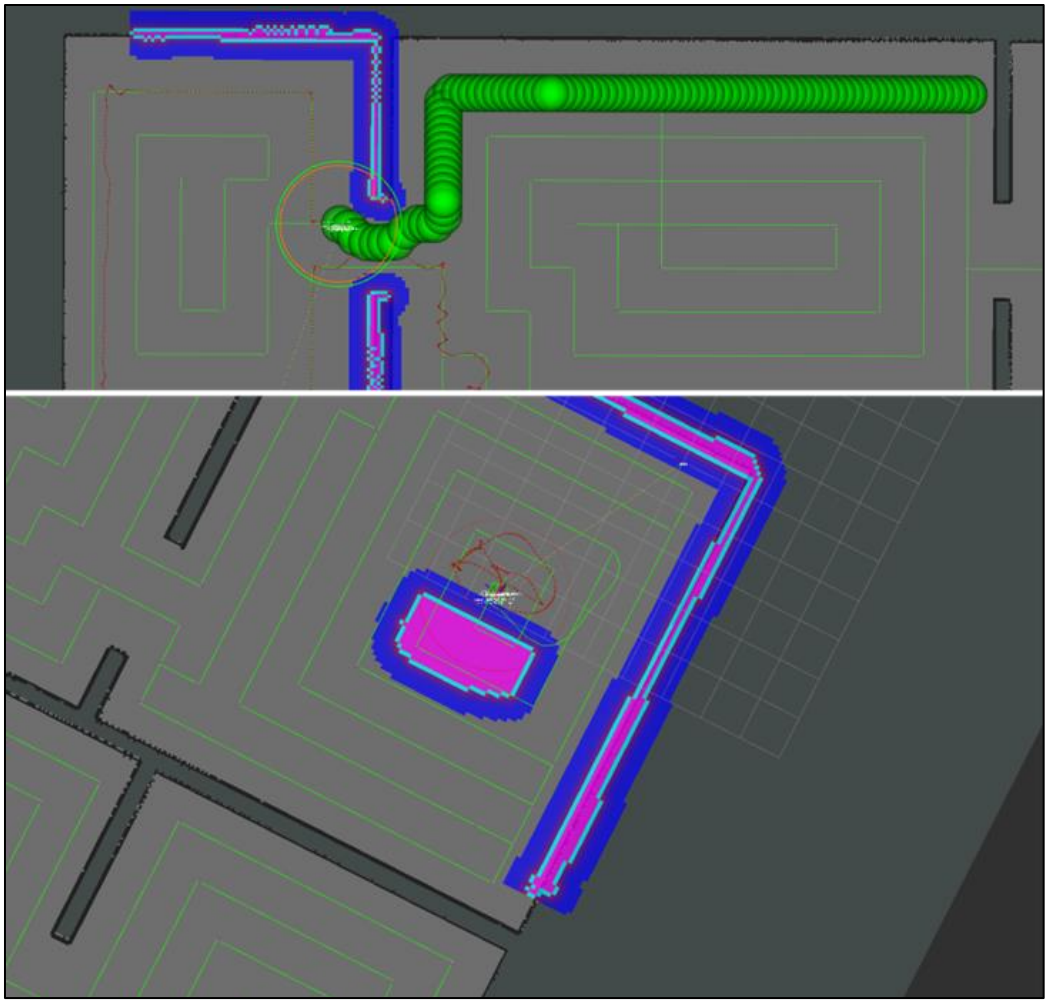
We define the plugins with the following code in the file called `common_costmap.yaml`.

```
obstacle_layer:
   track_unknown_space: true
   inf_is_valid: true
   obstacle_range: 7.5
   raytrace_range: 8.0
   observation_sources: point_cloud_lidar
   point_cloud_lidar: {data_type: PointCloud2,
    expected_update_rate: 0.5, topic: /segmenter/
    points_nonground, marking: true, clearing: true,
    max_obstacle_height: 1.0, min_obstacle_height: 0.0}
neg_obstacle_layer:
   track_unknown_space: true
   inf_is_valid: true
   obstacle_range: 4.0
   raytrace_range: 4.5
    observation_sources: cliff_scan
    cliff_scan: {data_type: LaserScan,expected_update_rate: 0.5,
    topic: /scan_cliff, marking: true, clearing: false,
    max_obstacle_height:9999, min_obstacle_height: -9999}
inflation_layer:
   cost_scaling_factor:  10.0
   inflation_radius:     3.0
```

While it is possible to have more than one observation source in an obstacle plugin, this is not recommended with the cliff detection node. If combined, the cliff detection node would mark the negative obstacle, and that obstacle would be subsequently cleared by the point cloud source. This is because both plugins are projected onto the same 2D plane, and the clearing mechanism of the point cloud topic ray-traces through that negative obstacle and removes it. By separating the obstacle nodes, we can mark the negative obstacles with the cliff detection node so they will not be cleared by the point cloud obstacle detection plugin. This is shown in the bottom image of Figure 14. Specifically, the pink square represents the negative obstacle being marked, but not cleared, on the cost map. The next step was to test the code in the Obstacle World with FCPP.

Figure 15 shows the result of integrating cliff detection with FCPP. The top image shows the elastic band local planner (green circles) following the global path generated by the backtracking spiral algorithm (green lines). The robot traverses the global path until it reaches a negative obstacle. The bottom image shows the negative obstacle being marked with the cliff detector node, and the red lines indicate potential paths the local planner is calculating, before converging on a solution, to avoid the negative obstacle. This pattern of following the global path and avoiding negative obstacles repeated until the robot finished exploring the entire area. We set a waypoint to bring the robot back to its starting location, but we really need to add a return-to-home function when it is completed. The FCPP package has a topic called `coverage_progress` that monitors coverage. Specifically, the values range from 0 (none) to 1 (full). Thus, we could have the robot return home when the `coverage_progress` was greater than 0.95.

Figure 15. Full coverage planning of the Obstacle World (*top*), and path planning around negative obstacles (*bottom*).



4 Future Work

In the future, we plan to continue investigating the image to grid map node, specifically for muti-robot teaming applications. Point clouds can be quite large, ranging from a few hundred megabytes to multiple terabytes. Transferring these large files between robots can be inefficient and time-consuming, especially in bandwidth-limited environments. However, 2D grayscale images are typically only a few hundred kilobytes in size. Once the robot receives the 2D image, image to grid map can generate the 2.5D grid map. We can then use the 2.5D grid map to generate 2.5D traversability occupancy grids for navigation. Figure 16 is an example of converting a 2D image to a 2.5D grid map.

To generate the 2D image, a point cloud was loaded into CloudCompare (2009). We selected "Set top view" for a top-down perspective. Next, we selected Edit → "colors" → "Height Ramp" (direction *z*) to generate the 2D image that is false colored along the *z*-axis. Finally, we selected Display → "Render to file" to generate the top image in Figure 16. To generate the bottom image of Figure 16, we first used the same `image_to_gridmap` node that was presented in Section 3.1.

While Figure 16 shows the conversion from 2D to 2.5D for an outdoor scene, Figure 17 demonstrates the conversion process for the interior of a building. The aforementioned steps were used to create the 2D grayscale image shown at the top of Figure 17, and the resulting 2.5D grid map is shown at the bottom.

Our next step will focus on streamlining the process to generate the top-down grayscale images. While initial steps will focus on converting data collected from UGVs or UAVs, we also plan to explore using digital elevation models as potential input sources. Finally, we plan to investigate various methods for transferring the images to the UGV in degraded communication scenarios.

Figure 16. Conversion of a 2D image to 2.5D grid map for an outdoor environment.

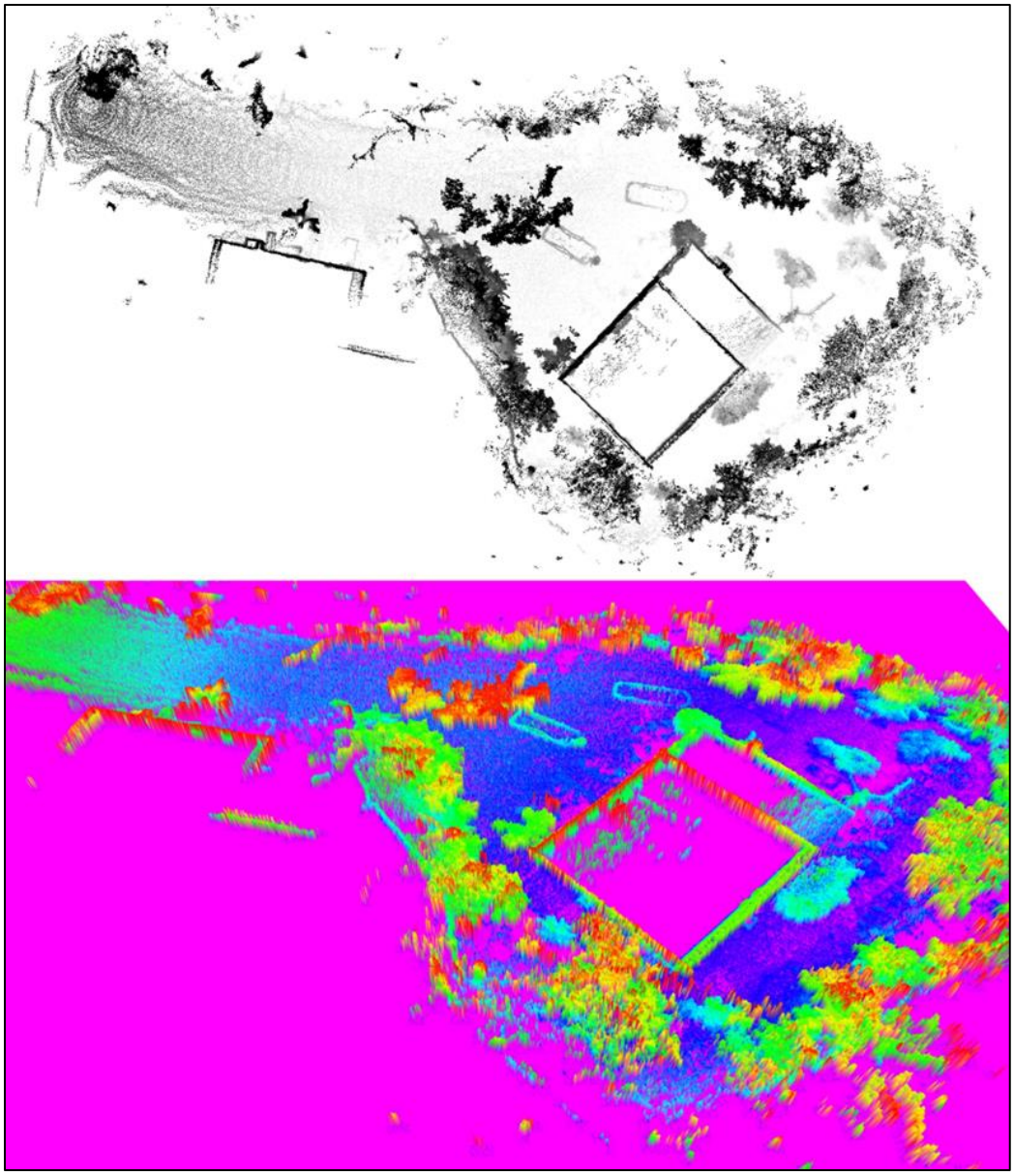
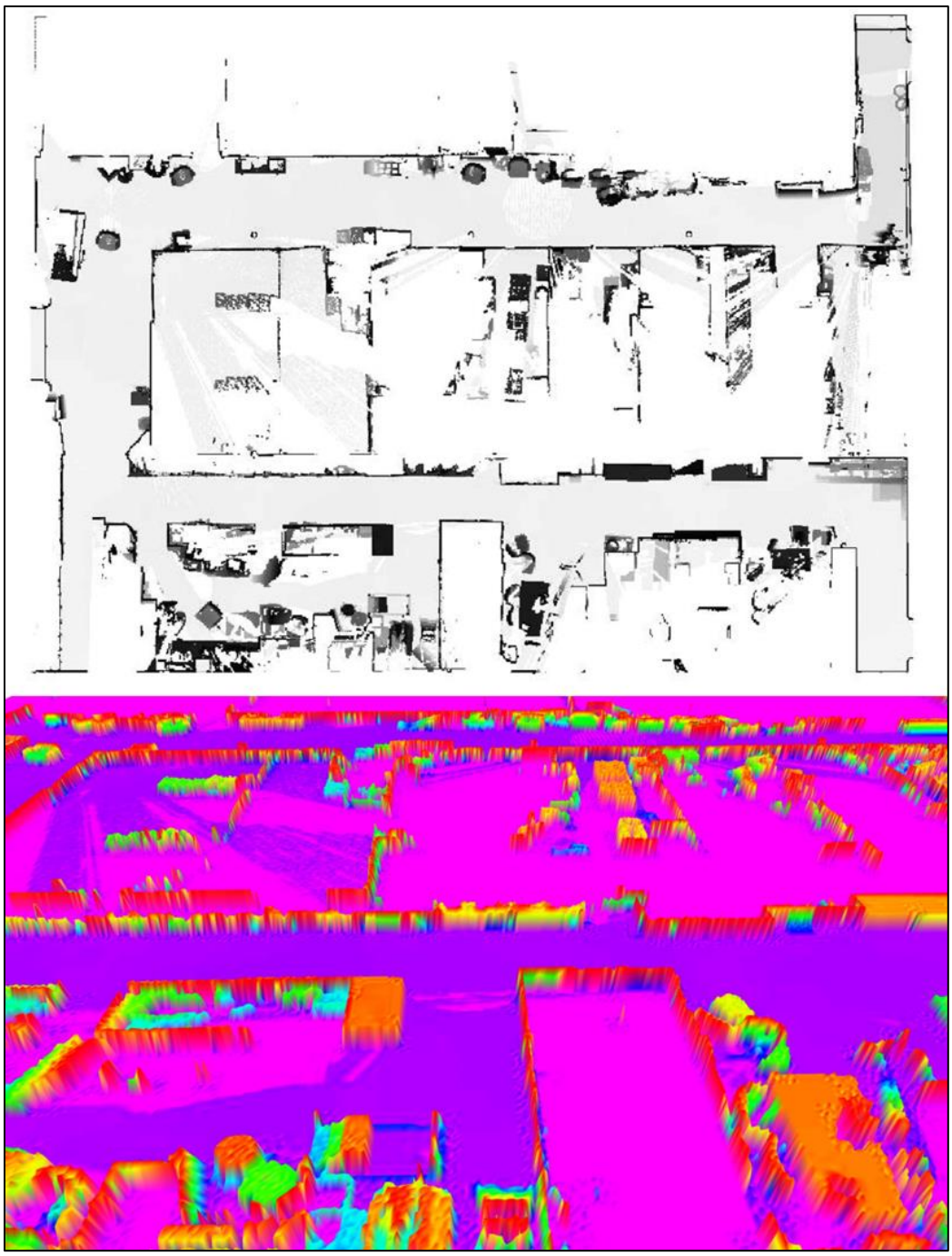


Figure 17. Conversion of a 2D image to 2.5D grip map for an indoor environment.



5 Summary

We explored various ways to thoroughly explore rooms while simultaneously avoiding negative obstacles. We evaluated these methods in simulated environments using an indoor virtual world with nonplanar floor geometry. Of the approaches we evaluated, the FCPP package was the preferred solution. The FCPP discretized individual rooms, used waypoints efficiently, integrated with move base flex, and did not path plan through walls. For detecting negative obstacles, using the traversability layer of grid maps in conjunction with the cliff detector works with teleoperation, waypoint navigation, and full autonomy. Subsequent reports will focus on testing these approaches on a physical robot. We also plan to combine these approaches with our low-visibility object identification approach (Christie et al. 2021b) to test their efficacy in detecting objects of interest.

References

ANYbotics. 2016. "Grid Map." *GitHub*. https://github.com/ANYbotics/grid_map.

ANYbotics. 2019. "Elevation Mapping." *GitHub*. http://github.com/ANYbotics/elevation_mapping.

Bormann, R., F. Jordan, W. Li, J. Hampp, and M. Hägele. 2016. "Room Segmentation: Survey, Implementation, and Analysis." In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, May, 1019–1026. https://doi.org/10.1109/ICRA.2016.7487234.

Brodskiy, Y., F. Schoenmakers, T. Clephas, J. Unkel, L. van Beek, and C. Lopez. 2004. "Full Coverage Path Planner." *GitHub*. https://github.com/nobleo/full_coverage_path_planner.

Building-Wide Intelligence Project. 2012. "EBand Local Planner." *GitHub*. https://github.com/utexas-bwi/eband_local_planner.

Christie, B. A., O. Ennasr, and G. P. Glaspell. 2021a. *Autonomous Navigation and Mapping in a Simulated Environment*. ERDC/GRL TR-21-5. Alexandria, VA: US Army Engineer Research and Development Center, Geospatial Research Laboratory. https://doi.org/10.21079/11681/42006.

Christie, B. A., O. Ennasr, and G. P. Glaspell. 2021b. *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.

Clearpath Robotics. 2021. "CPR Obstacle Gazebo." *GitHub*. https://github.com/clearpathrobotics/cpr_gazebo.

CloudCompare. 2009. 3D Point Cloud and Mesh Processing V.2. [Software]. *CloudCompare*. https://www.cloudcompare.org/.

Fankhauser, P. 2017. "Updated Terrain for Filters Demo." *GitHub*. https://github.com/ANYbotics/grid_map/blob/master/grid_map_demos/data/terrain.png.

Fankhauser, P., M. Bloesch, C. Gehring, M. Hutter, and R. Siegwart. 2014. "Robot-Centric Elevation Mapping with Uncertainty Estimates." *Mobile Service Robotics* 2014: 433–440. https://doi.org/10.1142/9789814623353_0051.

Fankhauser, P., M. Bloesch, and M. Hutter. 2018. "Probabilistic Terrain Mapping for Mobile Robots with Uncertain Localization." *IEEE Robotics and Automation Letters (RA-L)* 3 (4): 3019–3026. https://doi.org/10.1109/LRA.2018.2849506.

Fankhauser, P., and M. Hutter. 2016. "A Universal Grid Map Library: Implementation and Use Case for Rough Terrain Navigation." In *Robot Operating System (ROS)–The Complete Reference* (Volume 1), edited by Anis Koubaa, Section 5. https://doi.org/10.1007/978-3-319-26054-9 5. http://www.springer.com/de/ book/9783319260525.

Fraunhofer IPA. 2016. "IPA Coverage Planning." *GitHub*. https://github.com/ipa320/ipa_coverage_planning.

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.

Gonzalez, E., O. Alvarez, Y. Diaz, C. Parra, and C. Bustacara. 2005. "BSA: A Complete Coverage Algorithm." In *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, 18–22 April, Barcelona, Spain, 2040–2044. https://doi.org/10.1109/ROBOT.2005.1570413.

JohnTGZ. 2022a. "Negative-Obstacle Detection." *GitHub*. https://github.com/JohnTGZ/negative-obstacle-detection.

JohnTGZ. 2022b. "Negative-Obstacle-Detection Cliff Detector." *GitHub*. https://github.com/JohnTGZ/negative-obstacle-detection/blob/90959add8f3cc160af983f86dcd76643197ebeb1/src/cliff_detector.cpp#L13.

Magazino, S. Pütz, and J. Santos Simón. 2018. "Move Base Flex." *GitHub*. https://github.com/magazino/move_base_flex.

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.

TU Dortmund. 2016. "TEB Local Planner." *GitHub*. https://github.com/rst-tu-dortmund/teb_local_planner.

Appendix: Launch Files

A.1 Simulated World and Robot Launch File

```
<launch>
    <arg name="world_scale" default="3.0" />
    <arg name="walls" default="true" />
    <param name="obstacle_geom" command="$(find␣xacro)/
   xacro␣'$(find␣cpr_obstacle_gazebo)/urdf/
   obstacle_geometry.urdf.xacro'␣world_scale:=$(arg␣wor
   ld_scale)␣walls:=$(arg␣walls)" />
    <arg name="x" default="1.0"/>
    <arg name="y" default="1.0"/>
    <arg name="z" default="1.0"/>
    <arg name="yaw" default="0.0" />
    <arg name="use_sim_time" default="true" />
    <arg name="gui" default="true" />
    <arg name="headless" default="false" />
    <arg name="world_name" default="$(find␣
   cpr_obstacle_gazebo)/worlds/actually_empty_world.
   world" />
    <include file="$(find␣gazebo_ros)/launch/
   empty_world.launch">
      <arg name="debug" value="0" />
      <arg name="gui" value="$(arg␣gui)" />
      <arg name="use_sim_time" value="$(arg␣use_sim_time)" />
      <arg name="headless" value="$(arg␣headless)" />
      <arg name="world_name" value="$(arg␣world_name)"
   />
    </include>
  <node name="obstacle_world_spawner" pkg="gazebo_ros "
type="spawn_model" args="-urdf␣-model␣
obstacle_geom␣-param␣obstacle_geom␣-x␣0␣-y␣0␣-z␣0␣-Y
␣0" />
    <!-- Load robot_description param for tf, rviz and ga-
   zebo spawn. -->
    <param name="robot_description" command="$(find␣
   xacro)/xacro␣$(find␣turtlebot3_description)/urdf/
   turtlebot3_waffle_3d.urdf.xacro" />
    <!-- Spawn turtlebot into gazebo based on
   robot_description. -->
    <node name="spawn_urdf" pkg="gazebo_ros" type="
   spawn_model" args="-urdf␣-model␣turtlebot3␣-x␣$(arg␣x)
   ␣-y␣$(arg␣y)␣-z␣$(arg␣z)␣-param␣robot_description"
    />
   <!-- Publish turtlebot3 tf's.␣-->
␣␣␣␣<node␣pkg="robot_state_publisher"␣type="
   robot_state_publisher"␣name="waffle_state_publisher"
   ␣/>
</launch>
```

A.2 eband_local_planner.yaml

```
    EBandPlannerROS:
    odom_topic: /odom
    map_frame: map
  # Robot Configuration Parameters
    max_vel_x: 0.26
    min_vel_x: -0.26
    max_vel_y: 0.0
    min_vel_y: 0.0
  # The velocity when robot is moving in a straight line
    max_vel_trans:  0.26
    min_vel_trans:  -0.26
    max_vel_theta: 1.82
    min_vel_theta: -1.82
    acc_lim_x: 2.5
    acc_lim_y: 0.0
    acc_lim_theta: 1.1
    # Common Parameters
   xy_goal_tolerance: 0.01  # Distance tolerance for
    reaching the goal pose
  yaw_goal_tolerance: 0.01  # Orientation tolerance for
    reaching the desired goal pose
  rot_stopped_vel: 0.05  # Angular velocity lower bound that
    determines if the robot should stop to avoid
    limit-cycles or locks
   trans_stopped_vel: 0.05  # Linear velocity lower bound
    that determines if the robot should stop to avoid limit-
    cycles or locks
    # Visualization Parameters
    marker_lifetime: 1.0  # Lifetime of eband
    visualization markers
    # Trajectory Controller Parameters
    k_prop: 4.0  # Proportional gain of the PID controller
    k_damp: 3.5  # Damping gain of the PID controller
    k_int: 0.001       # I gain of the PID controller
    Ctrl_Rate: 10  # Control rate
   differential_drive: true  # Denotes whether to use
    the differential drive mode
    # rotation_correction_threshold: 2.0
    #
    bubble_velocity_multiplier:
    2.0 disallow_hysteresis:
    true
```

A.3 Scan_cliff_filter_tb3.yaml

```
    scan_filter_chain:
    #removes points outside of the specified bounds
    - name: angular_bounds
      type: laser_filters/LaserScanAngularBoundsFilter
      params:
        lower_angle: -3.124139361 #40(0.785398163),
```

```
   45(0.785398163), 50 (0.872664626), 55(0.9599311)
   70(1.221730476), 75 (1.308996939), 80 (1.3962)
    upper_angle: 3.124139361
- name: range_filter
  type: laser_filters/LaserScanRangeFilter
  params:
    use_message_range_limits: false       # if not specified de-
   faults to false
    lower_threshold: 0.05                 # if not specified de-
   faults to 0.0
   upper_threshold: 12.0                  # if not specified de-
  faults to 100000.0
   lower_replacement_value: -.inf         # if not specified de-
  faults to NaN
   upper_replacement_value: .inf          # if not specified de-
  faults to NaN
```

A.4 cliff_detector_params_tb3.yaml

```
####################
#Physical parameters
####################
#[meters] Height from base_link
lidar_height: 0.45 #1.0
#[degrees] Lidar pitch (about y axis of base link)
lidar_pitch: 15.0 #30.0
#[degrees] Lidar resolution (about y axis of base link)
lidar_resolution: 1.0
#################
#Failsafe params
#################
#[bool] whether to turn on failsafe. This will issue a
   /cmd_vel command that stops the robot
# should more than a user-specified percentage of
    the lidar be obstructed.
failsafe: false
#[meters] ranges less than this distance is consid-
    ered as an obstructed ray
failsafe_threshold_dist: 0.2
#[percentage] if the percentage of ranges with
    value less than
#"failsafe_threshold_dist" is exceeded, then lidar
    is obstructed.
failsafe_threshold_percentage: 0.4
#[seconds] Time out from when the lidar is obstructed
# before failsafe is activated
failsafe_timeout: 2.0
######################
#Cliff detector params
######################
#[meters] Distance along the ray below the ground
   plane to detect as a cliff area
```

```
cliff_threshold_constant: 0.05
##############
#Segmentation
##############
# Minimum points required to comprise a segment
    of cliff points
min_cliff_pts_in_seg: 3
```

Abbreviations

AI	Artificial intelligence
CONOP	Concept of operation
FCPP	Full coverage path planner
ML	Machine learning
MSSPIX	Maneuver Support, Sustainment, and Protection Integration Experiments
RAS	Robotics and autonomous systems
TSP	Traveling salesman problem
UGV	Unmanned ground vehicle

REPORT DOCUMENTATION PAGE

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED	
		START DATE	END DATE
August 2023	Final Technical Report (TR)	FY21	FY23

4. TITLE AND SUBTITLE

Unmanned Ground Vehicle (UGV) Full Coverage Planning with Negative Obstacles

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

6. AUTHOR(S)

Jin-Kyu Lee, Amir Naser, Osama Ennasr, Ahmet Soylemezoglu, and Garry Glaspell

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

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

8. PERFORMING ORGANIZATION REPORT NUMBER

ERDC TR-23-13

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)	11. SPONSOR/MONITOR'S REPORT NUMBER(S)
US Army Engineer Research and Development Center (ERDC) 3909 Halls Ferry Road Vicksburg, MS 39180-6199		

12. DISTRIBUTION/AVAILABILITY STATEMENT

DISTRIBUTION STATEMENT A. Approved for public release: distribution is unlimited.

13. SUPPLEMENTARY NOTES

Funding provided by FLEX-4 funds.

14. ABSTRACT

We explored approaches that offer full coverage path planning while simultaneously avoiding negative obstacles. These approaches are specific to unmanned ground vehicles (UGVs), which need to constantly interact with a traversable ground surface. We tested multiple potential solutions in simulation, and the results are presented herein. Full coverage path planner (FCPP) approaches were evaluated based on their ability to discretize their paths, use waypoints effectively, and be easily integrated with our current robot platform. For negative obstacles, we explored approaches that will integrate with our current navigation stack. The preferred solution will allow for teleoperation, waypoint navigation, and full autonomy while avoiding positive and negative obstacles.

15. SUBJECT TERMS

Autonomous robots--Control systems; Autonomous robots--Navigation; Military robots--Control systems; Military robots--Navigation

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES
a. REPORT	b. ABSTRACT	C. THIS PAGE		
Unclassified	Unclassified	Unclassified	SAR	45

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)
Construction Engineering Research Laboratory (CERL)
2902 Newmark Drive
Champaign, IL 61822