General Info

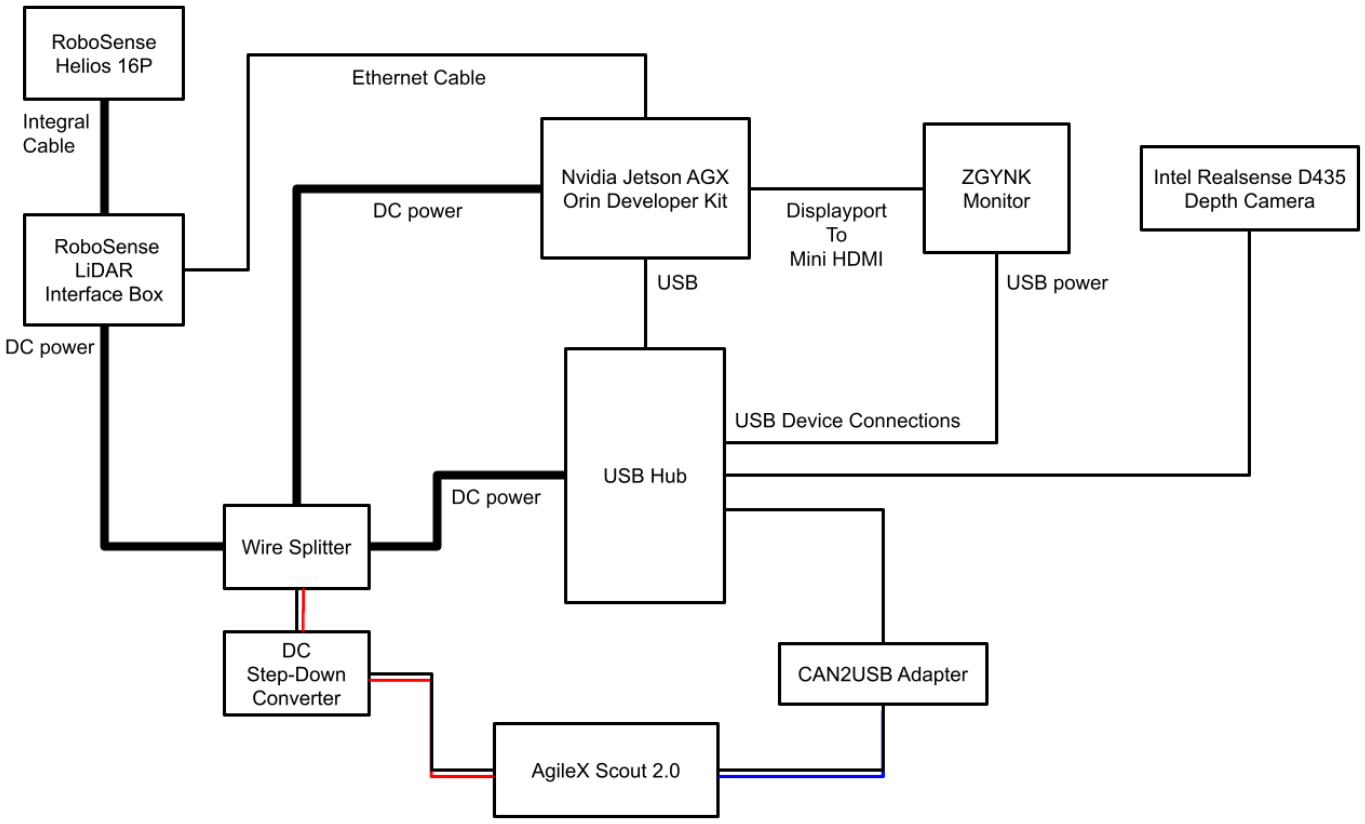
Username: orin
Password: agx

IP: 128.153.163.166

Ubuntu 22.04

Nvidia Jetson Linux 36.3
Linux Kernel 5.15
Part of JetPack 6.0

Hardware Layout



ROS2 Humble

Robot Operating System (ROS) is a collection of tools and libraries for robot applications. The Humble distribution of ROS2 was used in this project. When choosing a ROS distribution it is important to select one that has compatibility with all of your needed software packages (LiDAR SDKs, UGV drivers. etc.). The version of Ubuntu that is installed is also a factor to take into consideration.

ROS2 Humble documentation:
https://docs.ros.org/en/humble/index.html

ROS2 Humble install guide (Debian packages for Ubuntu 22.04)
https://docs.ros.org/en/humble/Installation/Ubuntu-Install-Debians.html

Colcon

Colon is the build tool used to create ROS2 workspaces. This tool will be needed when compiling packages from source. The following link contains a guide on using Colcon to build packages. The guide goes into more detail than is necessary for basic usage, but it can provide helpful background information.

https://docs.ros.org/en/humble/Tutorials/Beginner-Client-Libraries/Colcon-Tutorial.html

RVIZ

RVIZ is a program for visualizing various data types in ROS. This program can also be used to set goal positions for navigation. Programs such as slam-toolkit can provide custom panels for control via the GUI in RVIZ. RVIZ should be installed alongside ROS2 by default and can be launched with the following command (named rviz2 since it is for ROS2):

```
$ rviz2
```

Documentation for RVIZ can be found here:
https://wiki.ros.org/rviz

scout_base

AgileX Scout 2.0 User manual:
https://agilexrobotics.gitbook.io/scout2.0

SDK:
https://github.com/agilexrobotics/scout_ros2

IMPORTANT: If you are using ROS2 humble, you must download the humble branch of the scout_ros2 github

An installation guide can be found in the readme on the github page. There are also can2usb scripts that need to be executed as well that are not mentioned in the github and these can be found in the “ugv_sdk/scripts” folder.

After a fresh installation, the “setup_can2usb.bash” script must be executed. This will attempt to enable the kernel module “gs_usb”. If you are using an Nvidia Jetson, then this “modprobe gs_usb” command may fail since this kernel module was not compiled by default as of Jetson Linux version 36.3. Consult the “Missing gs_usb kernel module solution” section of this document for how to fix this.

Whenever the PC is restarted, the “bringup_can2usb_500k.bash” must be run to reenable the CAN communication to the scout. The following command can be used to verify that the CAN communication is successful. It will dump all data from the CAN port, if the output is blank or an error message pops up, then debugging is needed. This command only dumps the traffic over the CAN bus and is not necessary to control the Scout 2.0.

```
$ candump can0
```

scout_base usage:

```
$ ros2 launch scout_base scout_base.launch.py

# optional manual keyboard control in a separate terminal
# REDUCE THE SPEED BEFORE MOVING THE SCOUT 2.0 (z)
# Note: when changing the speed, the robot may move a short
# distance depending on the last inputted direction

$ ros2 run teleop_twist_keyboard teleop_twist_keyboard
```

How to get battery voltage info (with scout_base running):

```
$ ros2 topic echo /scout_status | grep battery_voltage
```

scout_description

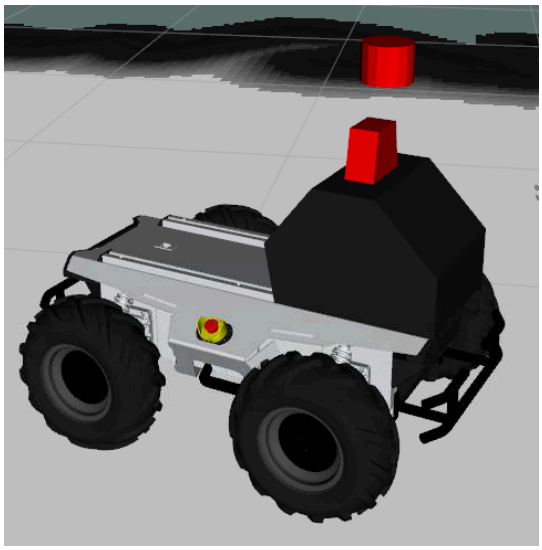
Scout_Description is one of the packages contained in the scout_ros2 github provided by AgileX. This package is created to provide a URDF model of the scout robot. This package provides the dimensions and visual appearance of the scout robot, as well as the transform information of the robot. However, as currently stated in the github commit history for scout_ros2, the provided code is not fully functional.

In the scout_description package, XACRO files are provided to generate the URDF model. The URDF files should not be manually edited as any changes will be overwritten when the URDF file is regenerated from the XACRO file. These files can be found in the “pathto_ws//src/scout_ros2-humble/scout_description/urdf” directory from the workspace used to install scout_ros2.

To fix the errors for the scout 2.0 description, several modifications of multiple XACRO files will be needed. Most of these edits involve correcting the paths to mesh files and wheel XACRO files. Key changes to the XACRO files are shown at the end of this section. The full XACRO files with the modifications can be found in our github page.

An additional link will also need to be added for the LiDAR sensor. This describes where the LiDAR sensor is relative to the center of the robot (the base_link). Collision and visual information can also be added to the LiDAR sensor. The modifications needed to add the LiDAR transform can be found in bold at the end of this section.

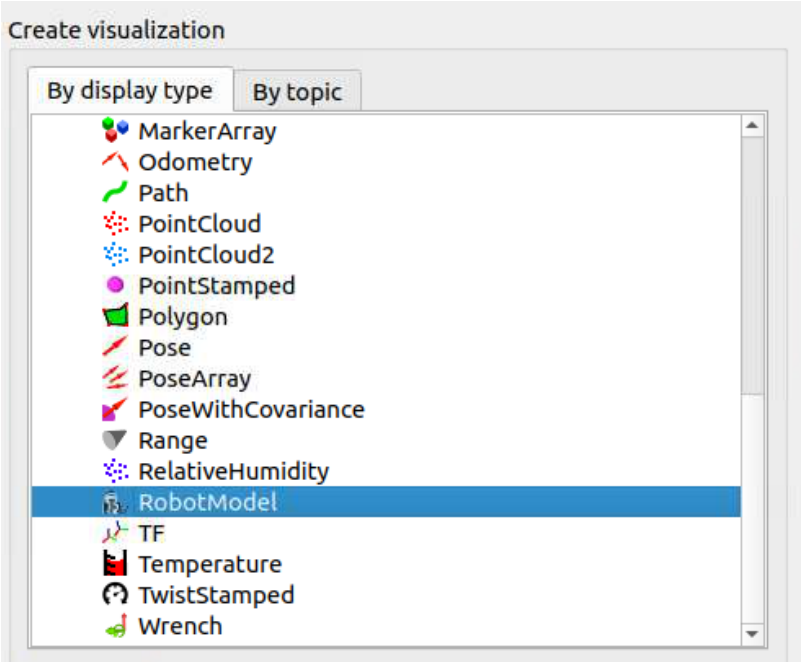
(Optional) 3D modeling programs such as blender can be used to modify the DAE files for the scout. The autokit mount for the scout 2.0 was not present in the dae files, and a simplified model of this was added.

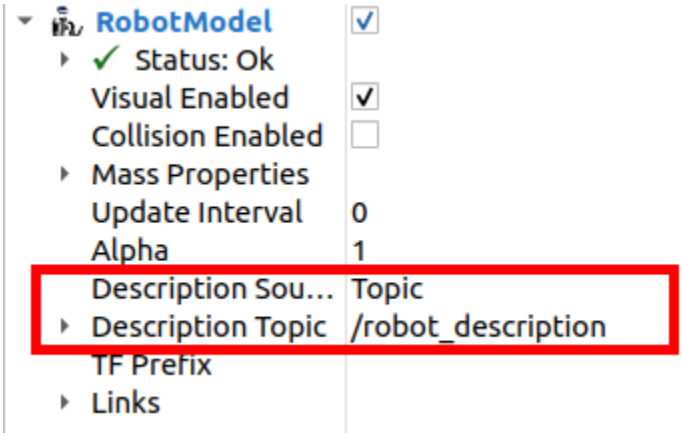


Once the XACRO files are modified. The URDF file must be updated. Assuming you modified the XACRO files in the src directory, rerunning the “colcon build” command will update the URDF files. The scout_description package can then be relaunched to verify if the URDF model was successfully generated.

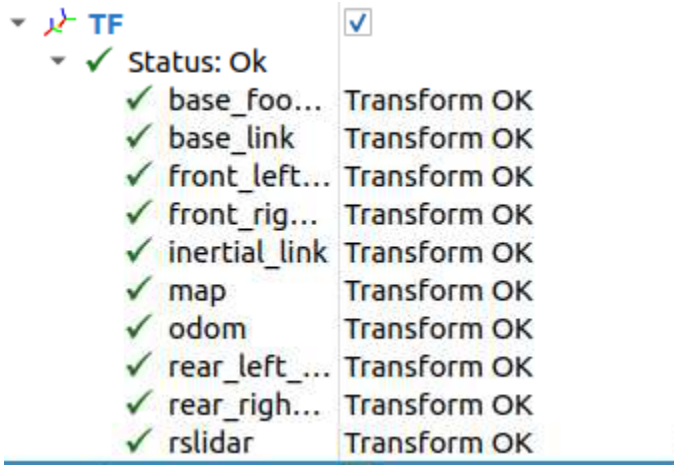
```
$ ros2 launch scout_description scout_base_description.launch.py
```

Launch RVIZ to visualize the robot model. Click the “Add” button and add a RobotModel. You must then select the /robot_description topic that the scout_description is publishing to. If the robot model is not properly loading at this point, any error messages from RVIZ and the scout_description terminal can be used for debugging.





It is also important to verify that all the desired transforms are present, including your specific lidar transform.



XACRO files

scout_v2.xacro

scout_v2.xacro

```
…

   <xacro:include filename="$(find scout_description)/urdf/scout_wheel_type1.xacro" />
   <xacro:include filename="$(find scout_description)/urdf/scout_wheel_type2.xacro" />

…

            <mesh filename="file://$(find scout_description)/meshes/base_link.dae" />

…

<!--@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@-->
<link name="rslidar">
  <inertial>
    <origin xyz="0 0 0" rpy="0 0 0"/>
    <mass value="0.125"/>
    <inertia ixx="0.001"  ixy="0"  ixz="0" iyy="0.001" iyz="0" izz="0.001" />
  </inertial>

  <collision>
    <origin xyz="0 0 0" rpy="0 0 0"/>
    <geometry>
      <cylinder radius="0.05" length="0.075"/>
    </geometry>
  </collision>

  <visual>
    <origin xyz="0 0 0" rpy="0 0 0"/>
    <geometry>
      <cylinder radius="0.05" length="0.075"/>
    </geometry>
  </visual>
</link>

<joint name="lidar_joint" type="fixed">
  <parent link="base_link"/>
  <child link="rslidar"/>
  <origin xyz="0.25 0 0.57" rpy="0 0 0"/>
</joint>
<!--@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@-->
```

scout_wheel_type1.xacro

scout_wheel_type1.xacro
<mesh filename="**file://$(find scout_description)**/meshes/wheel_type1.dae" />

scout_wheel_type2.xacro

scout_wheel_type2.xacro
<mesh filename="**file://$(find scout_description)**/meshes/wheel_type2.dae" />

Robosense LiDAR

Our hardware:
Robosense Helios 16P

Manufacturer site:
https://www.robosense.ai/en/rslidar/RS-Helios

SDK used:
https://github.com/RoboSense-LiDAR/rslidar_sdk

Link to Helios 16P user manual (3rd party site):
https://cdn-reichelt.de/documents/datenblatt/C900/RS-HELIOS-16P_USER_GUIDE_V1.0.1_EN.pdf

Configuration

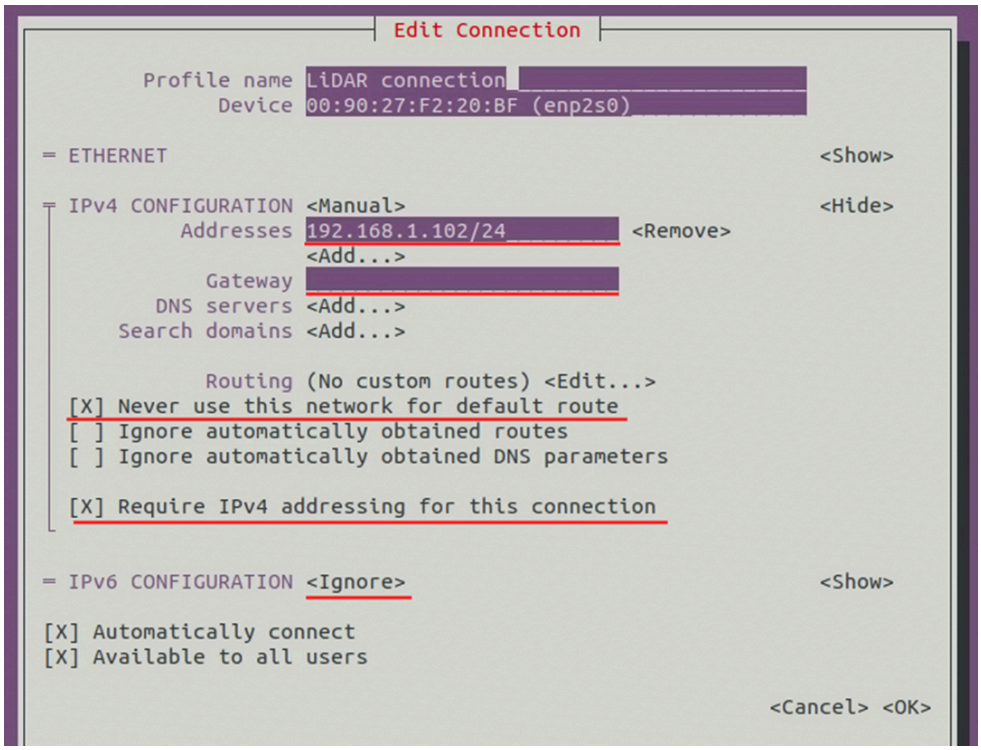
Documentation for how to install the SDK can be found in the readme of the SDK's github. Once the SDK is installed, it is necessary to modify the config.yaml file present in the src directory of the workspace. From this config file you must specify the name of your Robosense LiDAR model. For our purposes ensure that the output point cloud option is enabled. The name of the output topic and frame link can also be modified here.

The Robosense Helios 16 is connected to the PC via an ethernet cable. The network settings on the PC must be properly configured to successfully communicate with the LiDAR sensor. The default IP addresses for the sensor and PC can be found in the table below (taken from Helios 16 manual).

Device	Default IP
Robosense Helios 16	192.168.1.200
PC	192.168.1.102

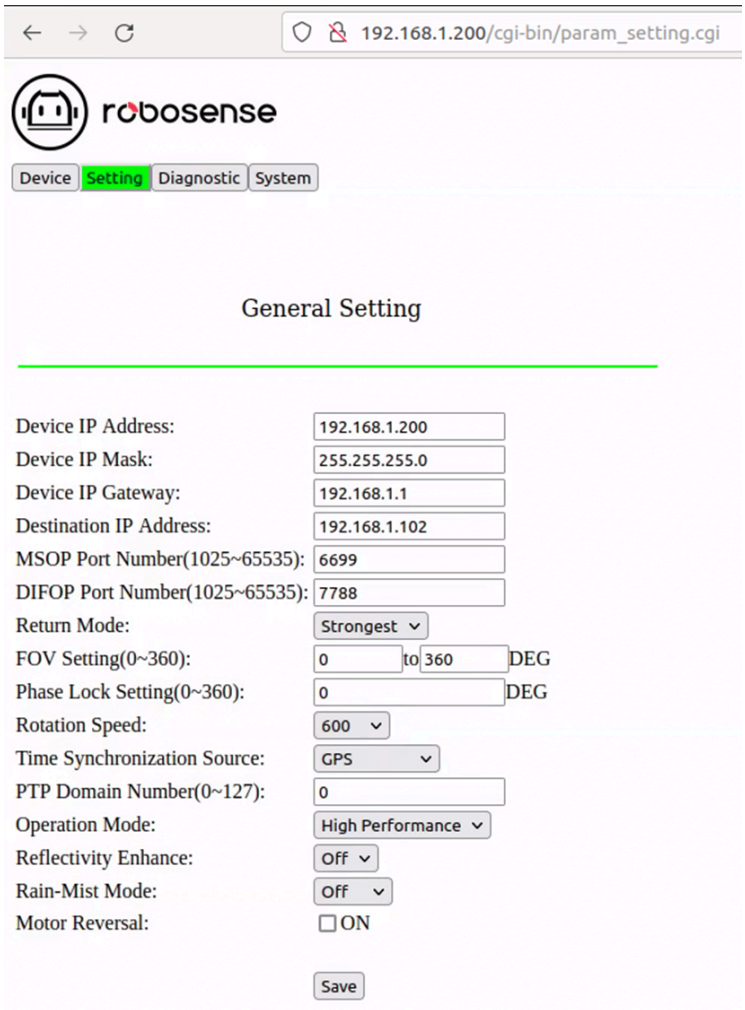
The program “nmtui” can be used to configure the network settings of the ethernet connection in Ubuntu. Some important settings to note are:

1. Set IPv4 configuration to “Manual” and add the expected PC IP Address
2. Enable the option “Never use this network for default route”
3. Enable “Require IPv4 addressing for this connection”
4. Set IPv6 configuration to “Ignore”



Web interface / debugging

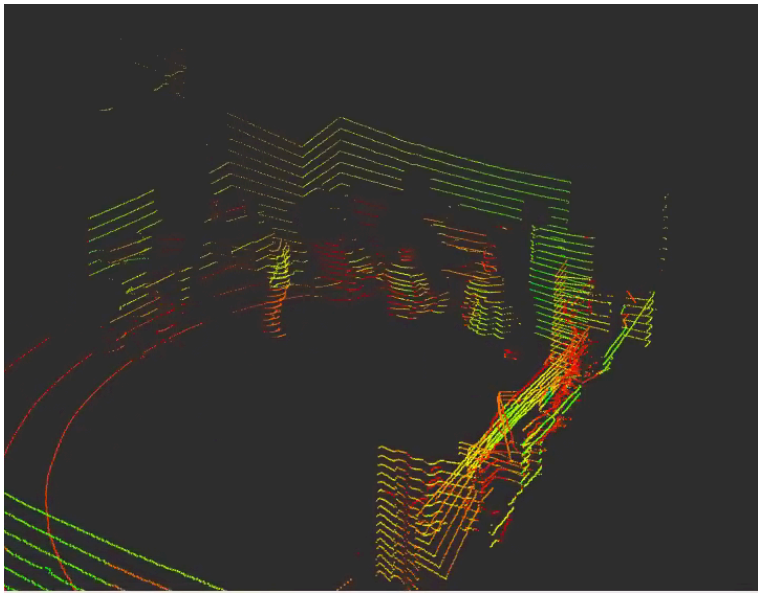
The Helios 16 features a web interface that can be loaded from a web browser by typing in the IP address of the LiDAR sensor. From here various settings for the LiDAR sensor can be configured. This web interface is useful as a method to confirm successful communication between the LiDAR sensor and the PC. If this web interface does not load, then debugging is required. Ensure that the LiDAR is properly powered and connected to the PC. Programs such as “wireshark” can be used to analyze the messages coming from the LiDAR sensor in case any of the default IP addresses were modified.



Publishing the point cloud data

Once everything is configured and the SDK is installed, the following command can be run to publish the point cloud data and visualize the output in RVIZ.

```
$ ros2 launch rslidar_sdk start.py
```



pointcloud to laserscan

To generate a 2D map with slam-toolbox, it is necessary to convert the 3D point cloud data from the LiDAR sensor into a 2D laser scan. This can be done using the pointcloud_to_laserscan package, and can be installed using the following command.

```
$ sudo apt-get install ros-humble-pointcloud-to-laserscan
```

https://index.ros.org/p/pointcloud_to_laserscan/

https://github.com/ros-perception/pointcloud_to_laserscan

To achieve the desired functionality, you must modify the provided sample python launch files or create a custom one. The launch file used to convert the point cloud data from the rslidar_sdk can be found below or on our github page. If you are using a different LiDAR sensor, then the remapping for the input point cloud must be modified. If you are using nav2, the output should be left as /scan, since that is what nav2 and slam-toolkit are expecting. Here the parameters of the conversion can be modified and the details of these parameters can be found on the pointcloud_to_laserscan github page. If the pointcloud_to_laserscan package was installed using apt-get, then the launch files can be found in the following directory: “/opt/ros/humble/share/pointcloud_to_laserscan/launch” (root privileges are needed to modify these files).

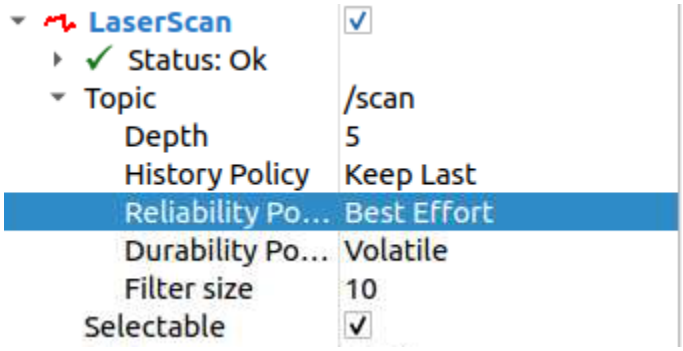
pointcloud-to-laserscan.py

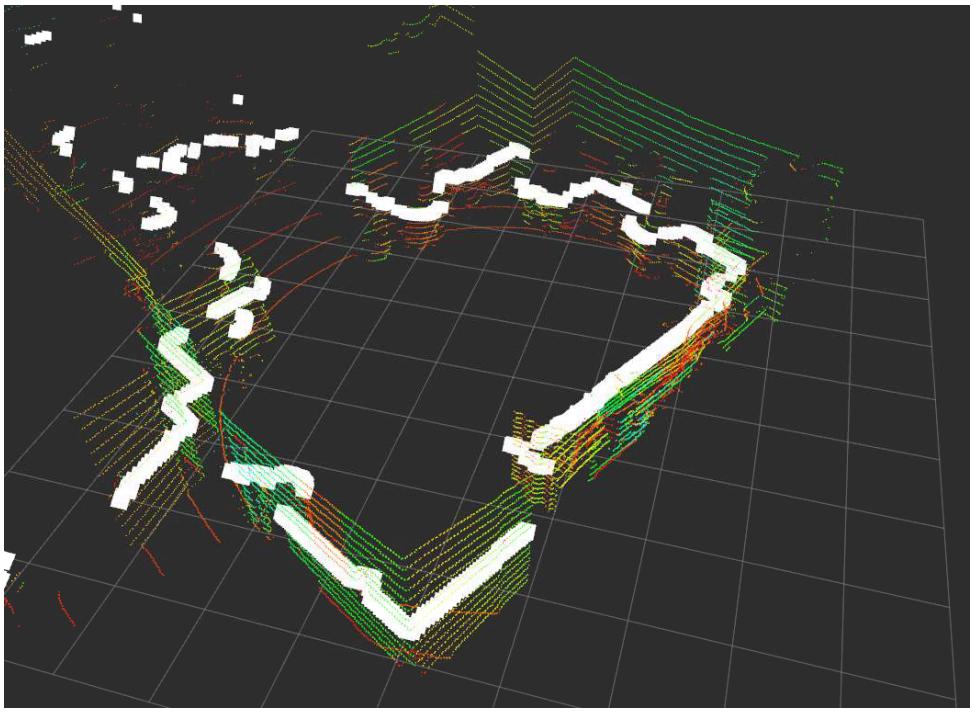
```
from launch import LaunchDescription
from launch.actions import DeclareLaunchArgument
from launch.substitutions import LaunchConfiguration
from launch_ros.actions import Node


def generate_launch_description():
    return LaunchDescription([
        Node(
            package='pointcloud_to_laserscan',
executable='pointcloud_to_laserscan_node',
            remappings=[('cloud_in', '/rslidar_points'),
                        ('scan', '/scan')],
            parameters=[{
                'target_frame': '',
                'transform_tolerance': 0.01,
                'min_height': -0.4,
                'max_height': 1.0,
                'angle_min': -3.1415,  # -M_PI/2
                'angle_max': 3.1415,  # M_PI/2
                'angle_increment': 0.0087,  # M_PI/360.0
                'scan_time': 0.3333,
                'range_min': 0.45,
                'range_max': 30.0,
                'use_inf': True,
                'inf_epsilon': 1.0
            }],
            name='pointcloud_to_laserscan'
        )
    ])
```

Once the launch files are properly configured, it can be launched using the following command (the name of the .py file will vary based on the name of your desired launch .py file). The package is programmed to only output the conversion if there is a subscriber for the output topic. Any errors output to the terminal can be used for debugging.

```
$ ros2 launch pointcloud_to_laserscan sample_pointcloud_to_laserscan_launch.py
```

RVIZ can be used to view the output laser scan. Simply add the /scan topic, and change the reliability policy for that topic to “Best Effort”. The laserscan conversion will not be output unless the reliability policy is correct. Shown below is a screenshot of the output 2D laser scan in white layered on top of the 3D point cloud data in color.





Nav2

Introduction/Installation

https://docs.nav2.org/index.html

Nav2 is a navigation stack for ROS2 allowing robots to autonomously navigate environments and perform various tasks. Nav2 can be installed using the commands shown below. In the nav2 documentation there are references to using gazebo and turtlebot for simulation. However as of writing this documentation, the packages for simulation were not found or easily installable on the Jetson. Any mention of simulation in the documentation for Nav2 can safely be ignored.

```
$ sudo apt install ros-humble-navigation2
$ sudo apt install ros-humble-nav2-bringup
```

https://docs.nav2.org/getting_started/index.html#installation

Robot Setup

The Nav2 documentation features a helpful set of pages regarding setting up your own robot from scratch. Since we are using scout 2.0 from AgileX, the information we need has already been provided via the scout_base and scout_description packages (assuming you fixed the broken scout_description package and added your specific sensor setup). The provided information is helpful, even if you are not necessarily setting things up from scratch yourself.

Nav2 documentation “First-Time Robot Setup Guide”
https://docs.nav2.org/setup_guides/index.html

Slam Toolbox

https://github.com/SteveMacenski/slam_toolbox

The documentation for Nav2 recommends using slam-toolbox for map building. It is possible to use another slam package so long as it outputs the data topics that Nav2 is expecting.

slam_toolbox can be installed using the following command:

```
# replace ros2-distro with your distro
$ sudo apt install ros-<ros2-distro>-slam-toolbox
```

Before running slam_toolbox or nav2, ensure that the following checklist has been completed:

1. Launch your robot's interface and state publisher (scout_base and scout_description in our case)
2. Launch your LiDAR's point cloud data publisher (rslidar_sdk in our case)
3. Launch pointcloud-to-laserscan to convert 3D data to 2D for slam-toolkit

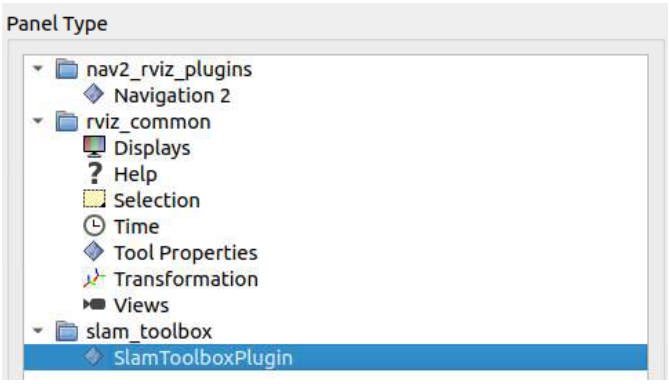
To begin building a map with slam_toolbox, run the following command:

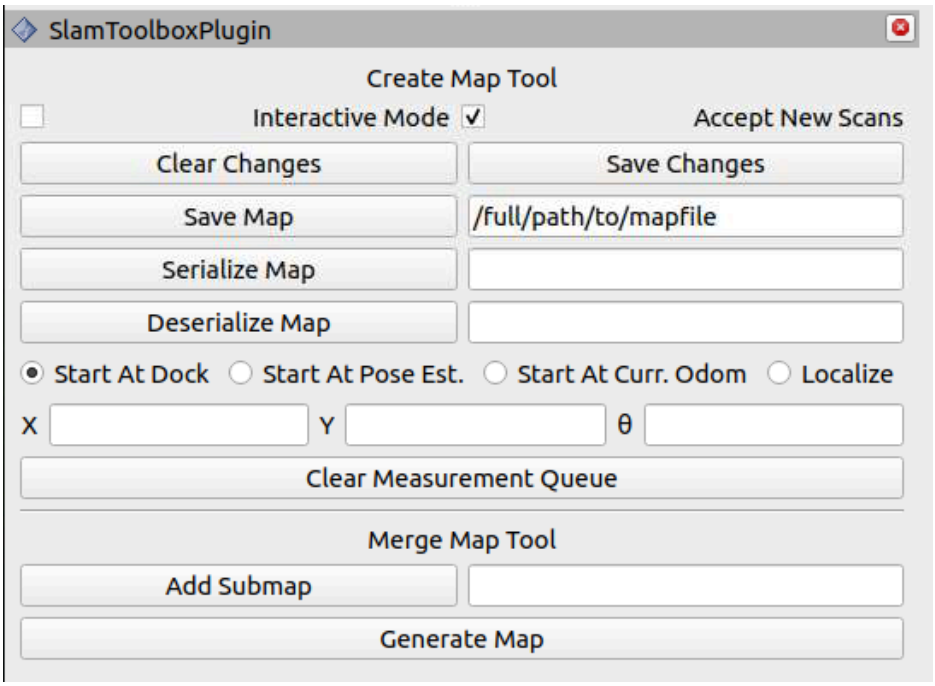
```
$ ros2 launch slam_toolbox online_async_launch.py
```

With this all setup you can open RVIZ and visualize the map topic. You can now begin manually driving to begin building a map of the area. slam_toolbox provides a RVIZ panel that can be added via "Panels -> Add New Panel". From this panel many functions can be performed, but the only one we are interested in is the "Save Map" feature. In the text box next to the button enter the full path and name of the map file you want to create. Once your map is complete click the "Save Map" button. Before closing slam_toolbox, you can verify if the map was successfully saved by opening the created PGM file in an image viewer.

slam_toolkit saves maps with a pair of PGM and YAML files. PGM files are black and white image files and the corresponding YAML file contains information regarding that PGM file. When loading saved maps in Nav2, the path and name of the yaml file is specified.

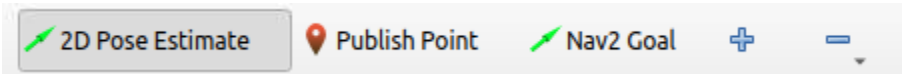
The PGM file can be edited using programs such as GIMP to remove noise or correct any minor errors. It is recommended to manually block off stairwells or entrances to offices that the robot should not have access to. This prevents navigation issues where the robot can attempt to navigate through walls or up and down staircases for example.





Using RVIZ to set poses

It is possible to input destinations and the initial pose using RVIZ. The following panel found towards the top of RVIZ can be used to perform these tasks. First click either “2D Pose Estimate” or “Nav2 Goal”, then the next click on the map will be set as the position. Clicking and dragging on the map can be used to set the direction of the pose and can be seen with a green arrow in RVIZ while holding down the click.



Autonomous Navigation While Mapping

While it is possible to autonomously navigate while generating the map, it tends to be faster to manually drive the scout while building the map. The following commands are necessary to begin autonomous navigation while mapping:

```
# terminal 1
$ ros2 launch nav2_bringup navigation_launch.py

# terminal 2
$ ros2 launch slam_toolbox online_async_launch.py
```

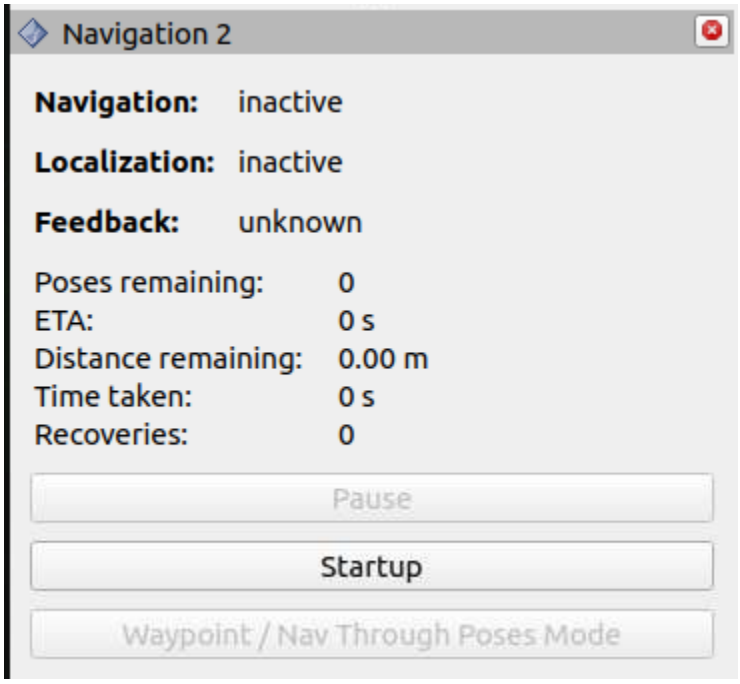
Navigating a saved map file

```
# terminal 1
# change “/path/to/your-map.yaml”
$ ros2 launch nav2_bringup bringup_launch.py use_sim_time:=False autostart:=False
map:=/path/to/your-map.yaml

# terminal 2
$ ros2 run rviz2 rviz2 -d $(ros2 pkg prefix nav2_bringup)/share/nav2_bringup/rviz/nav2_default_view.rviz
```

Running these commands will allow you to begin navigating a map (assuming that you are using the default launch file and the collision_monitor is yet to be added).

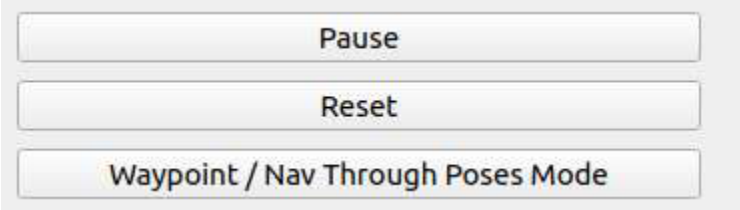
Nav2 RVIZ panel

Assuming RVIZ was launched using a config file provided by nav2, the “Navigation 2” panel should be visible. Assuming that nav2 was not set to start automatically, the startup button should be shown. After clicking startup, the map will be loaded and displayed in RVIZ. You now need to set the initial pose of the robot using the “2D Pose Estimate” tool. This can be done over multiple attempts to get the position of the robot properly initialized. The laserscan can be compared to the map to determine how accurate the initial pose is. It is best to make this initial pose fairly accurate before beginning navigation.

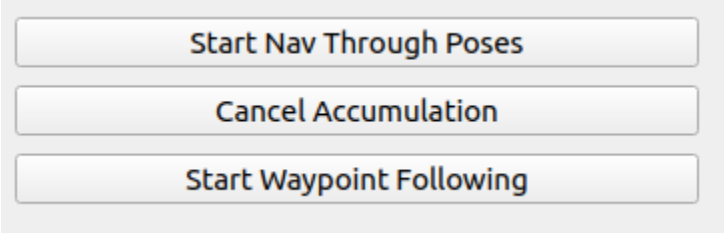


Once the initial position is set, you can begin navigating. Simply use the “Nav2 Goal” tool to set a goal pose and the robot will begin navigating to that point. The Navigation 2 panel can be used to pause or cancel navigation. To begin navigating via a series of waypoints, simply click the “Waypoint / Nav Through Poses Mode” button. While this mode is enabled, multiple goal waypoints can be set, and the robot will navigate to each of those destinations one by one. After all waypoints are set, click the “Start Waypoint Following” button. Click the “Start Nav Through Poses” button to return to the previous behavior.

Nav2 panel after startup:



Nav2 panel in waypoint following mode:



Collision Monitor

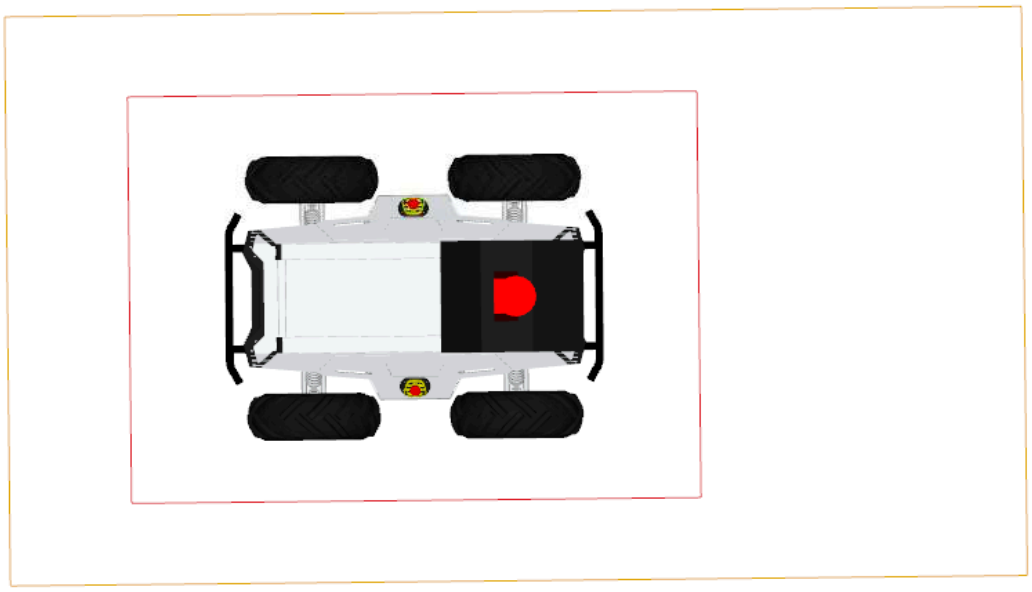
https://docs.nav2.org/tutorials/docs/using_collision_monitor.html

By default there is no collision detection enabled in nav2. It will route around obstacles but if the robot is left with no other choice it will collide with walls, objects, etc.. nav2_collision_monitor can be used to prevent collisions through custom polygons. The collision monitor works by modifying the velocity commands that are eventually given to the robot.

If an object is detected inside the polygon (if enough points from the laserscan are within the polygon), then the specified modifier for that polygon will be performed. It is common to have a “stop” polygon and a “slow down” polygon. If an object is getting too close for comfort to the scout, then the speed will be reduced. If the scout is about to collide with an object, then it will stop completely. The scout will continue to move again if the object that entered the “stop” polygon is removed. If the object that triggered the “stop” polygon is a static object (ie. a wall), then the robot will be permanently stuck without manual intervention. Polygon specifications can be configured through the “collision_monitor_params.yaml” file that can be found in the “/opt/ros/humble/share/nav2_collision_monitor/params” directory (assuming nav2 was installed using apt / apt-get)

As noted in the nav2 tutorial linked above, it is important to ensure that the velocity commands are routed through the collision monitor. These can be done by modifying the remappings in the bringup launch file that is being used. Once the velocity commands have the proper topic remapping, simply launch the nav2_collision_monitor node alongside the bringup launch file.

```
$ ros2 launch nav2_collision_monitor collision_monitor_node.launch.py
```



Known Issues

- If the stop polygon is active for too long, eventually the scout 2.0 will rotate in place to a specific angle. This could lead to the scout colliding with the object triggering the stop polygon. A temporary fix is to use the Scout 2.0 emergency stop button or cancel the navigation with the nav2 RVIZ panel.

Rotation Shim Controller

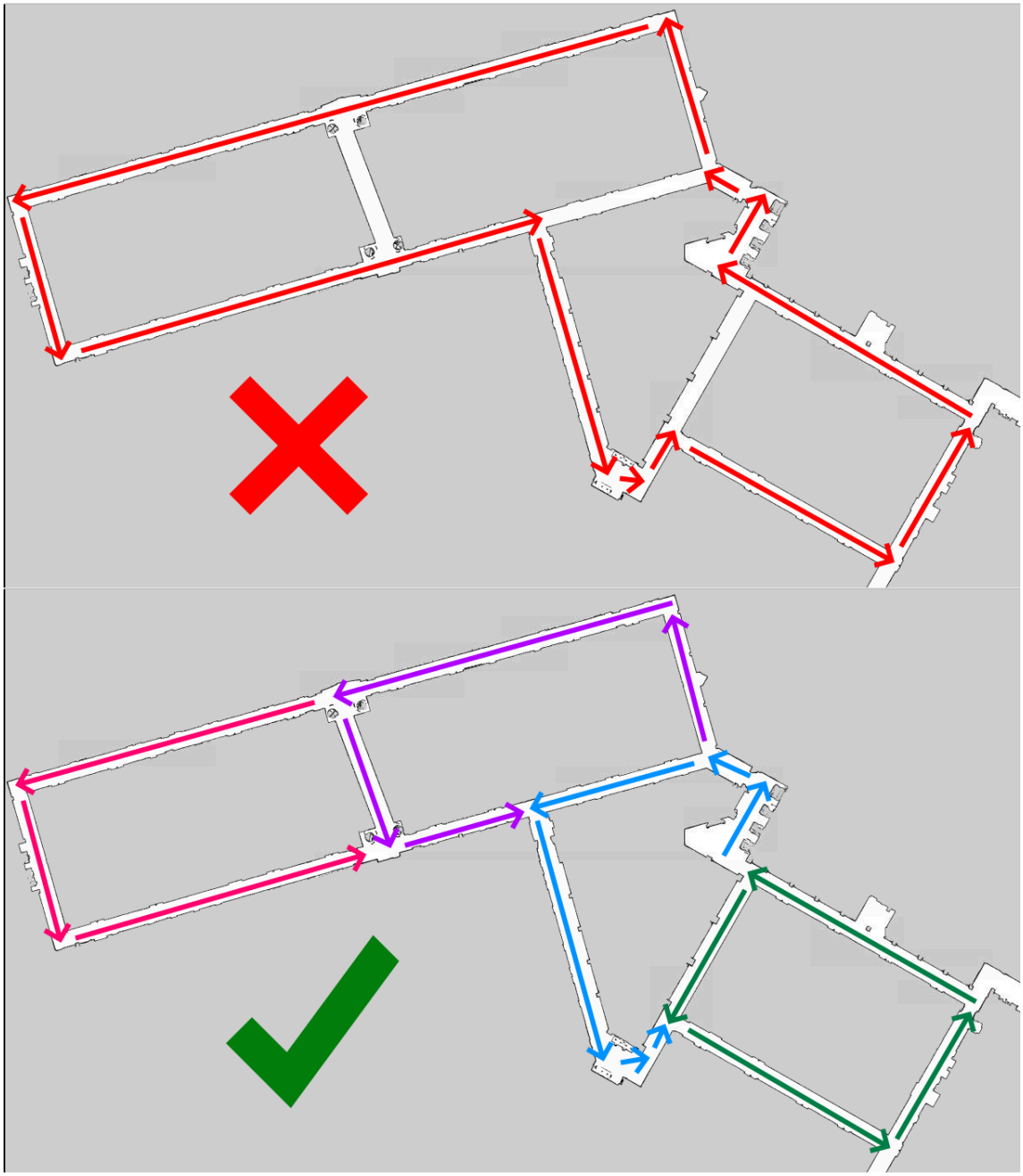
https://docs.nav2.org/tutorials/docs/using_shim_controller.html

The rotation shim controller is a plugin that adds support for rotate-in-place with the robot during navigation (assuming your UGV is capable of rotating in place. Supported on scout 2.0). When inputting a new path with this plugin, the robot will rotate in place to face the newly generated path before moving. This can be useful in tight hallways where there is not enough room to turn around without rotating in place. As noted in the tutorial linked above, modification of the nav2_params.yaml file is necessary to enable this plugin. This parameters file can be found in the directory “/opt/ros/humble/share/nav2_bringup/params” (assuming that nav2 was installed using a package manager).

General SLAM map building tips

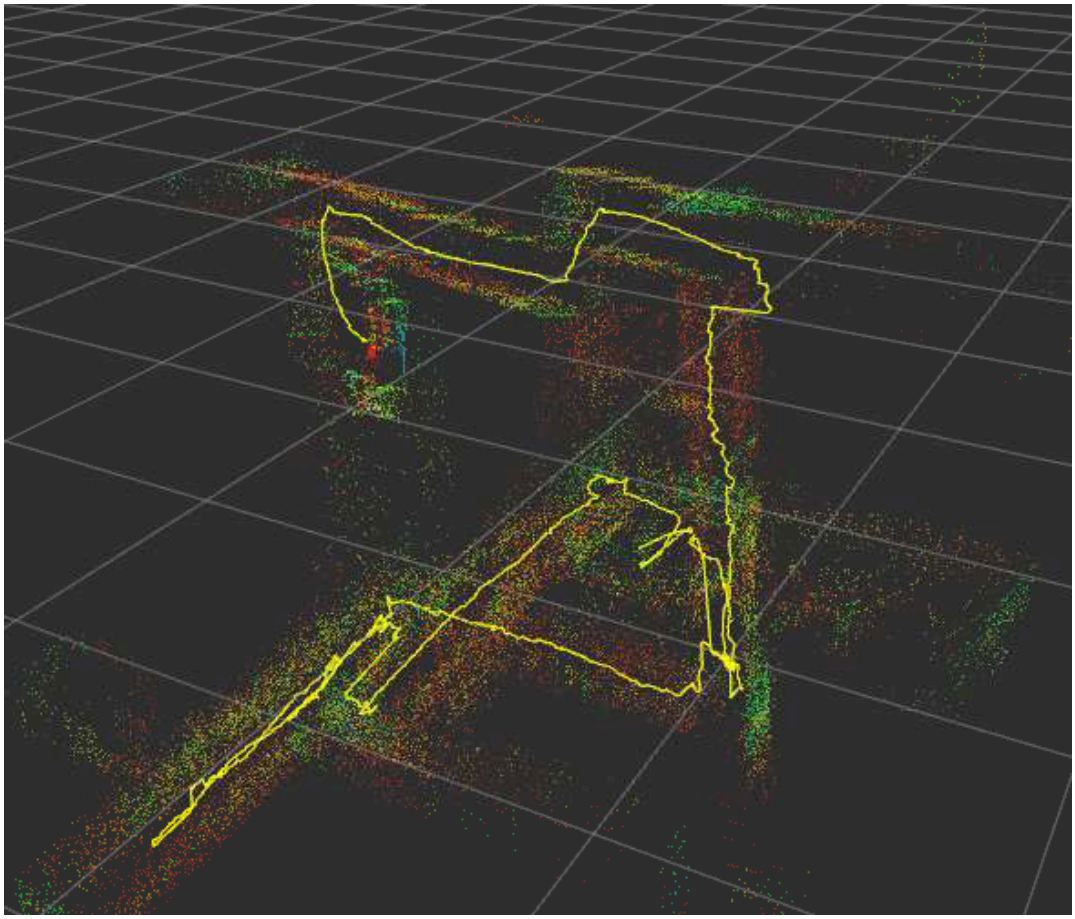
Loop Closure

When you are mapping out a large area, it is best to map out the area in small loops. This allows your slam algorithm to automatically correct minor errors in alignment. If you attempt to map out a large area without looping back until the very end, there will be alignment issues that cannot be automatically corrected (hallways/walls clipping into each other, etc). See the image below for an example route.



Cornering hallways

Depending on your slam package, turning around hallway corners can easily lead to alignment issues (especially when trying to build a 3D map. This was not as much of an issue with slam-toolbox). Having a 360 degree LiDAR also helps maintain alignment. When mapping and standing close to the scout, avoid moving while the scout is turning a corner. This avoids an issue where the slam algorithm attempts to use you as a point of reference (while both you and the sensor are moving independently) which can lead to unintended results.



Jetson Flashing guide using VirtualBox (VM)

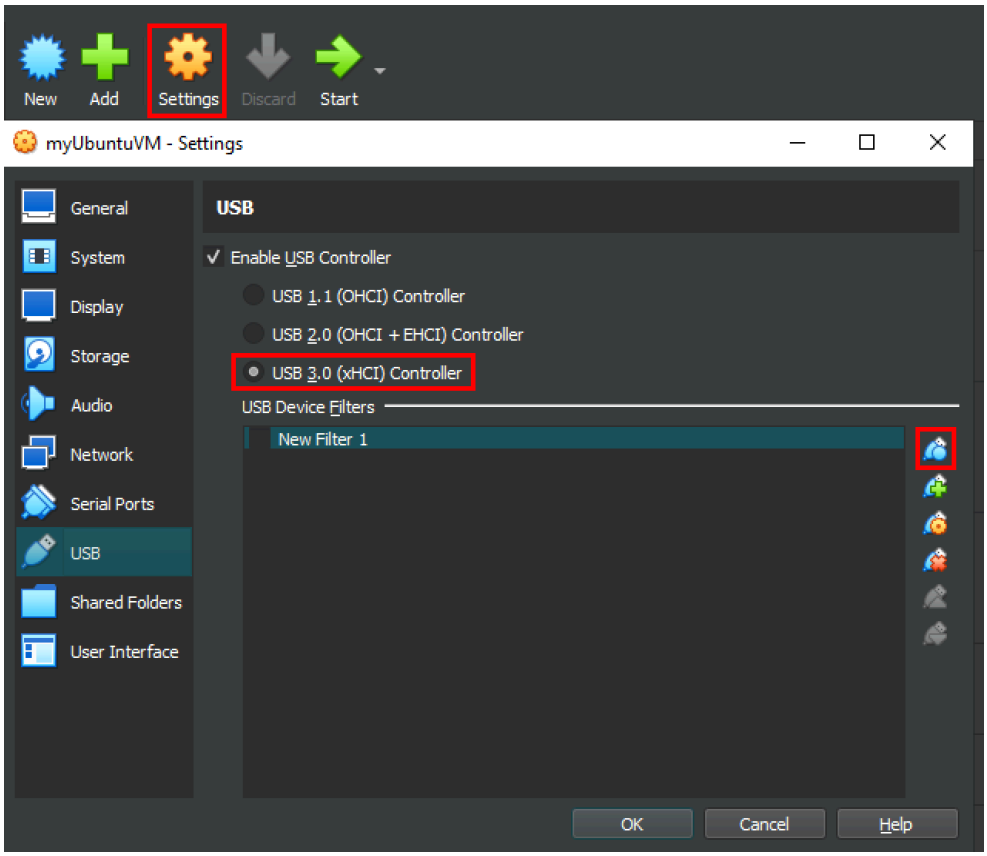
VirtualBox and Ubuntu

Install VirtualBox:
https://www.virtualbox.org/

Download an Ubuntu .iso compatible with the Linux for Tegra flash tools (22.04 or 20.04 as of r36.3) and set up your virtual machine. (22.04 was used)
https://releases.ubuntu.com/jammy/

Basic tutorial from ubuntu on setting up a VM (make sure to allocate enough RAM and storage to the VM. 100GB of storage is plenty, 8 GB of RAM is preferable)
https://ubuntu.com/tutorials/how-to-run-ubuntu-desktop-on-a-virtual-machine-using-virtualbox

After creating the VM, open the settings for the machine and navigate to the usb tab. Add a new filter with all the fields set to blank, this connects any usb device connected to your physical machine to the VM. Ensure the USB is set to 3.0



Boot your Jetson device into recovery mode
https://developer.nvidia.com/embedded/learn/jetson-agx-orin-devkit-user-guide/howto.html#force-recovery-mode
Jetson AGX orin developer kit example:

1. Begin with the power disconnected
2. hold down the force recovery button
3. Connect the power cord
4. Release the force recovery button after the power LED turns on (press power button if it doesn't automatically turn on)

To ensure your Jetson is in recovery mode, connect the jetson usb port for flashing to your pc and type the following command into the terminal on your VM. An NVIDIA device should appear on the list of devices.

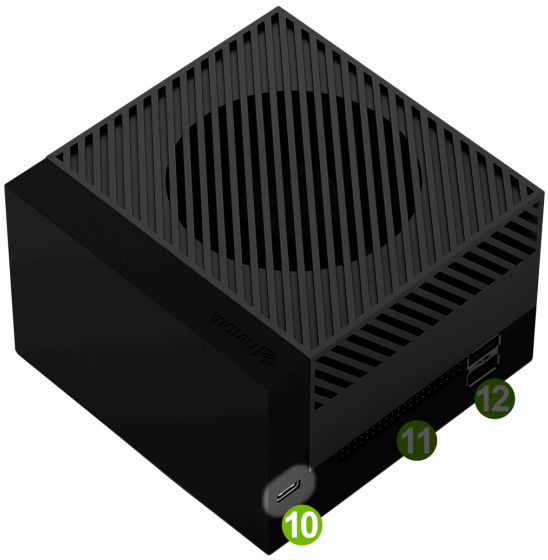
```
$ lsusb
```

Example device when jetson is in recovery mode:

```
Bus 001 Device 003: ID 0955:7023 NVIDIA Corp. APX
```

Determine whether your devkit is in force recovery mode:
https://docs.nvidia.com/jetson/archives/r36.3/DeveloperGuide/IN/QuickStart.html#to-determine-whether-the-developer-kit-is-in-force-recovery-mode

Jetson agx orin usb port for flashing (connect to host PC):
https://developer.nvidia.com/embedded/learn/jetson-agx-orin-devkit-user-guide/developer_kit_layout.html



Linux_for_Tegra

Follow this guide to set up the environment needed to flash the jetson. Do not attempt to flash the jetson at this point.
https://docs.nvidia.com/jetson/archives/r36.3/DeveloperGuide/IN/QuickStart.html

The list of Jetson Linux versions can be found in the following link. The sample root file system and release package can be downloaded by selecting the desired version. Ensure that you are consulting the correct documentation for the Jetson Linux version you are trying to flash.

https://developer.nvidia.com/embedded/jetson-linux-archive

Commands used:

```
$ export L4T_RELEASE_PACKAGE=Jetson_Linux_R36.3.0_aarch64.tbz2
$ export SAMPLE_FS_PACKAGE=Tegra_Linux_Sample-Root-Filesystem_R36.3.0_aarch64.tbz2
$ export BOARD=jetson-agx-orin-devkit


$ tar xf ${L4T_RELEASE_PACKAGE}
$ sudo tar xpf ${SAMPLE_FS_PACKAGE} -C Linux_for_Tegra/rootfs/
$ cd Linux_for_Tegra/
$ sudo ./tools/l4t_flash_prerequisites.sh
$ sudo ./apply_binaries.sh
```

Python Code

https://forums.developer.nvidia.com/t/lets-make-flashing-from-virtual-machines-reliable/232682/4
Using the information from this forum post, you must modify the tegraflash_internal.py which can be found at ./bootloader/tegraflash_internal.py

Navigate to the tegraflash_reboot function and replace it with the following code:

```
def tegraflash_reboot(args):
    if args[0] == 'coldboot':
        info_print('Coldbooting the device')
    elif args[0] == 'recovery':
        info_print('Rebooting to recovery mode **** [ Within the next 30sec, manually
plug out the USB cable for flashing and put the cable back in ] ****')
    else:
        raise tegraflash_exception(args[0] + " is not supported")

    if int(values['--chip'], 0) == 0x21 or int(values['--chip'], 0) == 0x18:
        try:
            command = exec_file('tegradevflash')
            command.extend(['--reboot', args[0]])
            run_command(command)
            time.sleep(2)
        except:
            tegraflash_tboot_reset(args)
    else:
        if check_ismb2():
            tegraflash_tboot_reset(args)
            info_print('Rebooting to recovery mode ========================')
            time.sleep(30)
        else:
            command = exec_file('tegradevflash')
            command.extend(['--reboot', args[0]])
            run_command(command)
```

What these changes do is add a 30 second delay to the program when the Jetson reboots. During this 30 second window you must physically disconnect and reconnect the USB from the jetson to your PC.

Flashing

You must have windows sounds enabled on your PC and you must listen for the device disconnect notification sound during the entire flashing process. Whenever you hear the device disconnect notification, you must disconnect and reconnect the Jetson usb within 30 seconds. This will happen multiple times during the flashing process and constant attention is required. The duration of this flashing process will vary depending on your host PC's hardware. When running on a 5600 rpm HDD, the process can take over an hour.

Once the code has been modified, the command to initiate the flashing process for your target hardware can be run.

To flash jetson AGX Orin Devkit internal eMMC:

```
$ sudo ./flash.sh jetson-agx-orin-devkit internal
```

To flash jetson AGX Orin Devkit NVMe SSD (if it is installed):

```
$ sudo ./tools/kernel_flash/l4t_initrd_flash.sh --external-device nvme0n1p1 \
  -c tools/kernel_flash/flash_l4t_t234_nvme.xml \
  --showlogs --network usb0 jetson-agx-orin-devkit external
```

Missing gs_usb kernel module solution

Currently Jetson Linux version 36.x does not include the gs_usb module that is needed by the can2usb for the scout 2.0 robot. If the jetson has already been flashed, this module can be added without the need for a reflash. If the jetson has not been flashed yet, this customized kernel can be used to flashed the jetson.

Compiling the kernel module (without reflashing)

A compatible linux host system can be used to compile the needed kernel module. Download and extract the “Driver Package (BSP)” that matches the version that was used to flash the jetson. The following kernel customization guide can be used to download the kernel sources and required toolkit. Do not begin compiling until the configs have been modified to compile the desired module.

Kernel Customization guide:
https://docs.nvidia.com/jetson/archives/r36.3/DeveloperGuide/SD/Kernel/KernelCustomization.html

Once the kernel source has been set up, the defconfig file for the target hardware must be modified. The jetson agx orin devkit uses the arm64 instruction set so the accompanying defconfig must be modified. The path of the defconfig will vary depending on the location of your Linux_for_Tegra and the linux version you are compiling. The following is an example of the path for the desired defconfig file:

/home/me/Desktop/Linux_for_Tegra/source/kernel/kernel-jammy-src/arch/arm64/configs

Open the defconfig file and add the following line to the file:

```
CONFIG_CAN_GS_USB=m
```

You can now return to the kernel customization guide and complete the “Building the Jetson Linux Kernel” and “Building the NVIDIA Out-of-Tree Modules” sections. Once the compilation is complete, the desired gs_usb.ko file can be found. The following is an example of where the file can be found.

/home/me/Desktop/Linux_for_Tegra/source/kernel/kernel-jammy-src/drivers/net/can/usb

Getting the module on the Jetson

This file must then be transferred to the Jetson via usb, cloud, scp, etc.. Once the file is on the jetson, copy it to the following path (path name will vary depending on your Jetson Linux version):

/lib/modules/5.15.136-tegra/kernel/net/can

Once the gs_usb.ko file is in the proper folder run the following command to probe all modules for their dependencies. This will ensure that the kernel detects the newly added module.

```
$ sudo depmod -a
```

Blacklist conflicting CAN modules

The NVIDIA jetson agx orin devkit features a 40 pin connector for GPIO and various other connections. Among these connections are 2 CAN controllers that are supported by their own kernel modules. These CAN modules prevent the gs_usb module needed by the CAN 2 USB adapter from functioning.

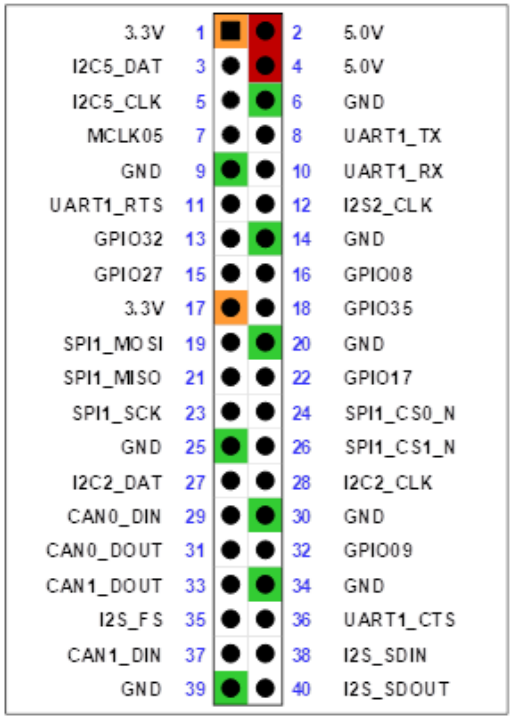
By blacklisting the mttcan module used by the 40 pin connector the gs_usb module will be able to function as expected. This solution assumes that the CAN connections on the jetson are not needed. If these connections are needed then another solution will need to be found.

To blacklist the mttcan module in ubuntu, add the following line to the /etc/modprobe.d/blacklist.conf:

```
blacklist mttcan
```

Restart the jetson to make sure the changes take effect and the can2usb connection should now be functioning. The steps present in the readme for the scout SDK can now be followed to successfully control the scout.

Documentation regarding the Jetson's CAN controllers:
https://docs.nvidia.com/jetson/archives/r36.3/DeveloperGuide/HR/ControllerAreaNetworkCan.html



Misc kernel module info

```
config CAN_GS_USB

     tristate "Geschwister Schneider UG interfaces"

     help

       This driver supports the Geschwister Schneider and bytewerk.org

       candleLight USB CAN interfaces USB/CAN devices

       If unsure choose N,

       choose Y for built in support,

       M to compile as module (module will be named: gs_usb).
```

Jetson Ubuntu Setup

Setting up Ubuntu from a clean Jetson flash is a relatively simple process. This can be done with a GUI via the displayport or the command line via serial USB. More information can be found in the provided links. Be sure to update the packages with the following commands:

```
$ sudo apt update
$ sudo apt dist-upgrade
$ sudo reboot
$ sudo apt install nvidia-jetpack
```

When using the setup via command line, setting up a wifi connection may fail. The setup can be completed without an internet connection, and programs such as “nmtui” can later be executed to connect to the desired wifi connection.

“Getting Started with Jetson AGX Orin Developer Kit” oem-config
https://developer.nvidia.com/embedded/learn/get-started-jetson-agx-orin-devkit

Jetson AGX Orin Devkit How to
https://developer.nvidia.com/embedded/learn/jetson-agx-orin-devkit-user-guide/howto.html

Helpful Linux commands

```
# print L4T version and other information
$ cat /etc/nv_tegra_release
```

```
# monitor memory, CPU, and GPU usage/temperatures
$ sudo tegrastats
```

```
# change power mode with command line (GUI is recommended)
$ sudo /usr/sbin/nvpmodel -h
```

Tight VNC

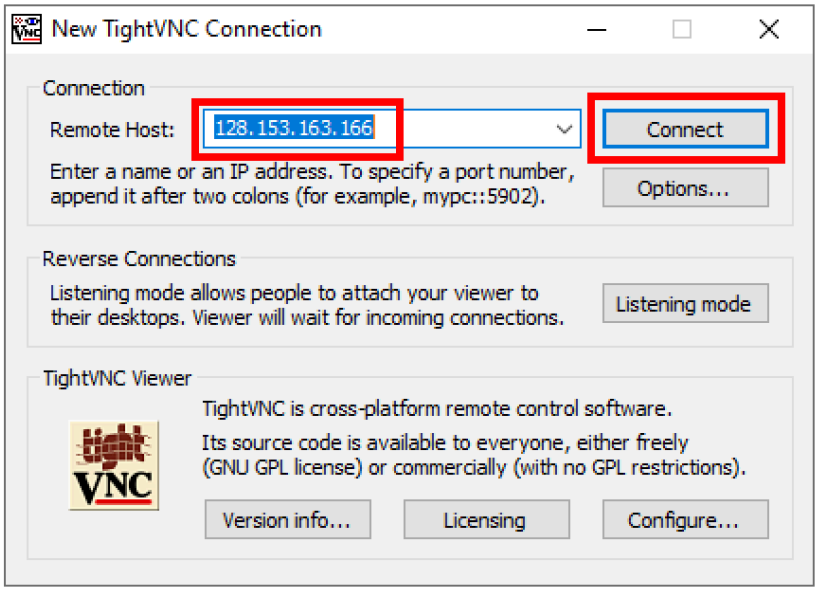
Download TightVNC for the PC you would like to use to remotely access the jetson
https://www.tightvnc.com/

TightVNC install guide: (disable VNC server since we don't need it on our PCs)
https://itservices.cas.unt.edu/services/labs/articles/install-tightvnc-viewer

Setting up VNC on jetson (ignore the linked VNC viewer: it's not free)
https://developer.nvidia.com/embedded/learn/tutorials/vnc-setup

Once TightVNC is installed on your pc:

1. launch TightVNC Viewer
2. input the IP address of the jetson and click connect
3. You will be prompted to enter a password (configured in the jetson vnc set up) (same as admin password)



Accessories purchased

SSD

Jetson SSD setup guide
https://nvidia-isaac-ros.github.io/getting_started/hardware_setup/compute/jetson_storage.html

Jetson AGX Orin devkit info:
M.2 M-Key slot (J1)
Supports 2280 size card, with PCIe x4 interface. No SATA support.

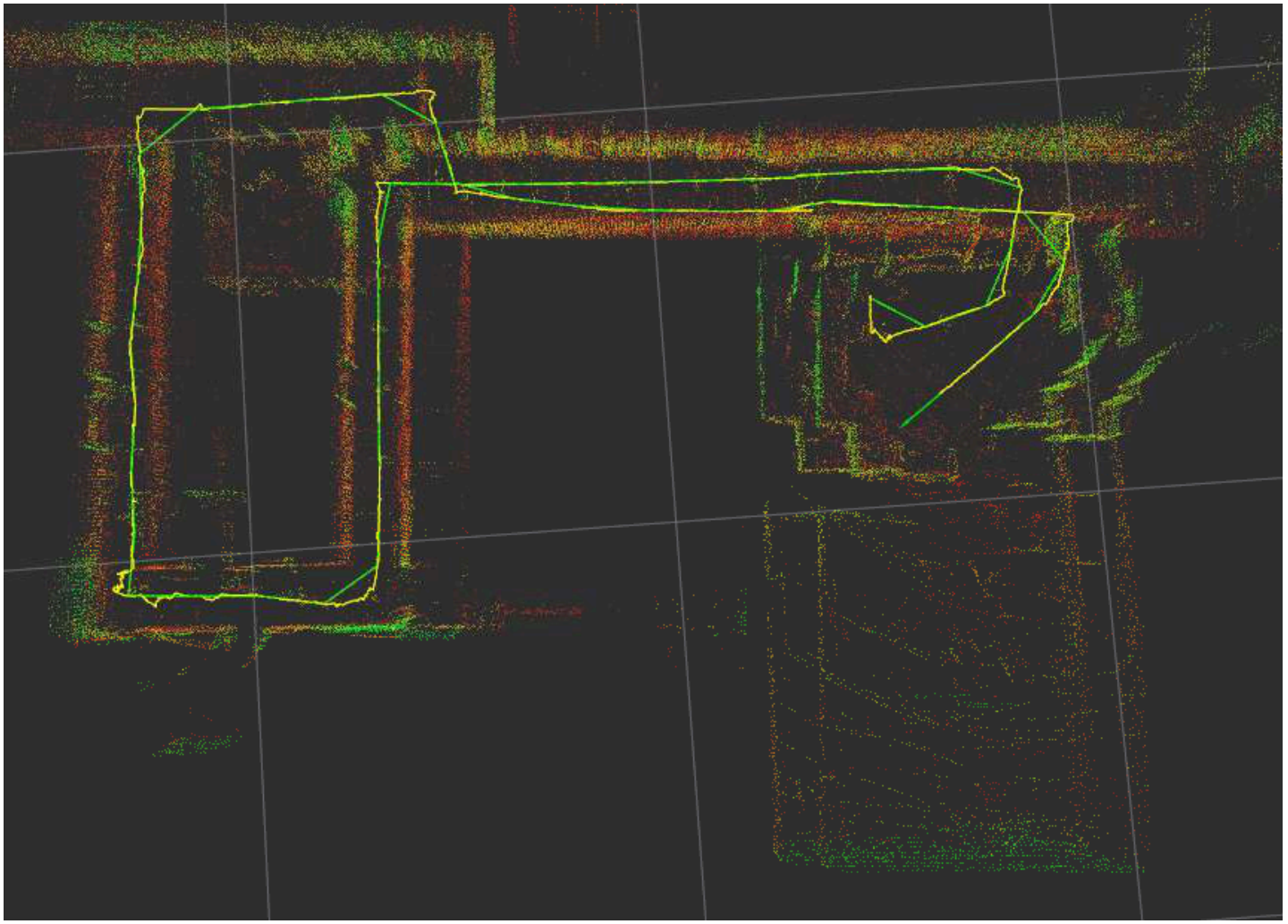
Crucial M.2 SSD (1 TB version)
https://www.amazon.com/Crucial-500GB-PCIe-NAND-3500MB/dp/B0B25LQQPC/?th=1

Video Cables

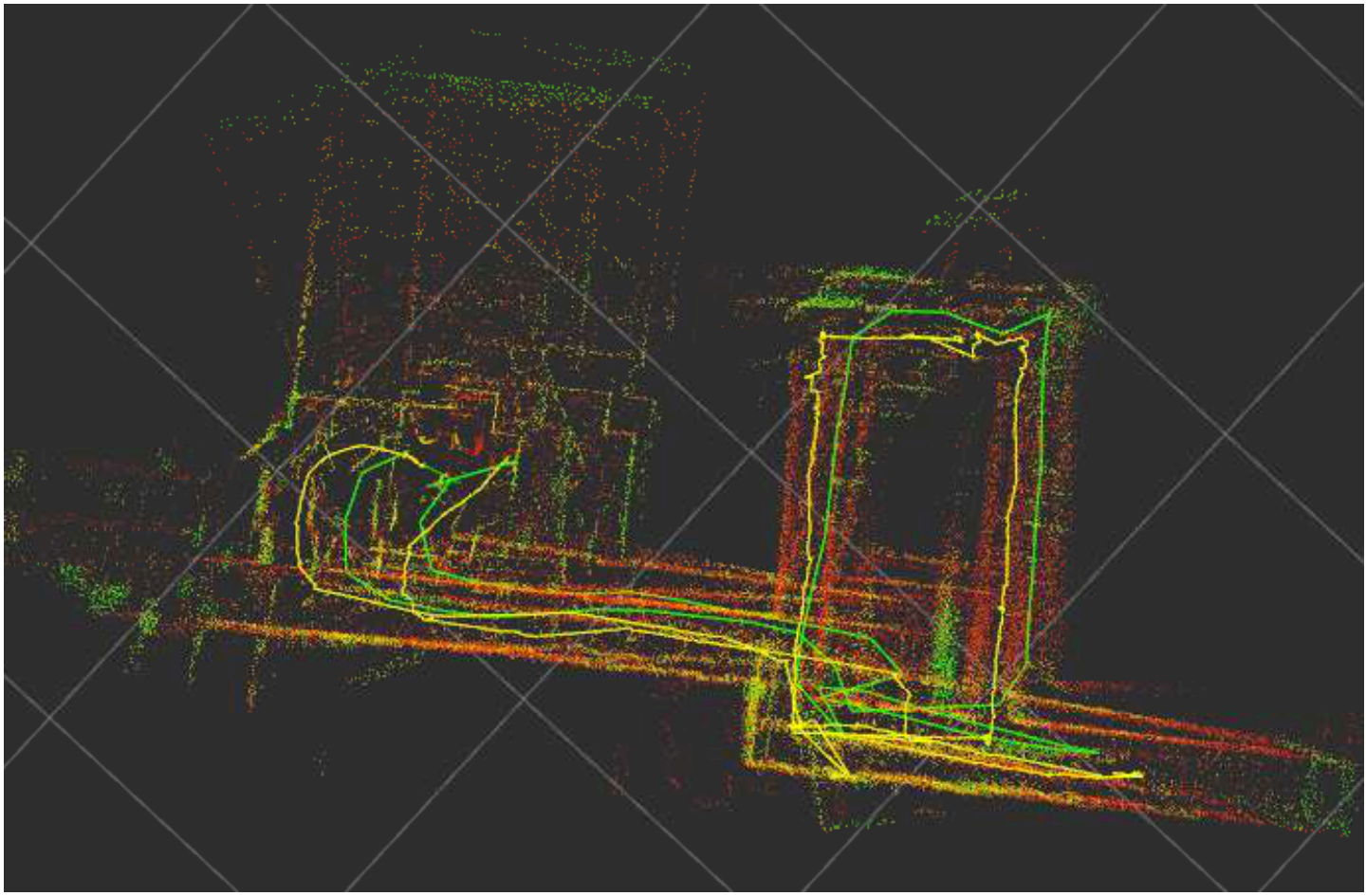
Amazon DP to mini HDMI
https://www.amazon.com/DisplayPort-Uni-Directional-Supports-Portable-Monitor/dp/B0CR3MB2GW/

Gallery

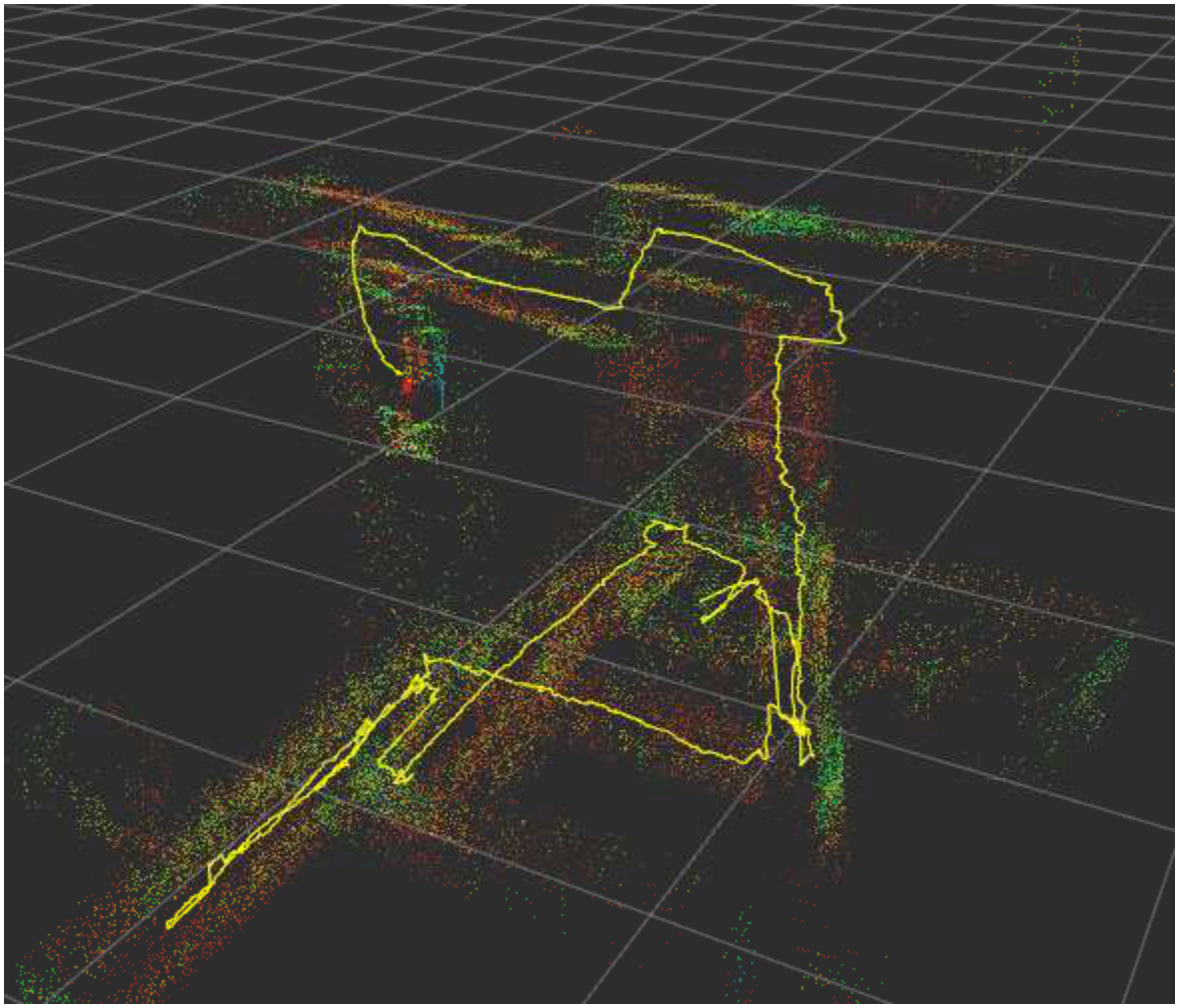
Buggy hallway map with misalignment



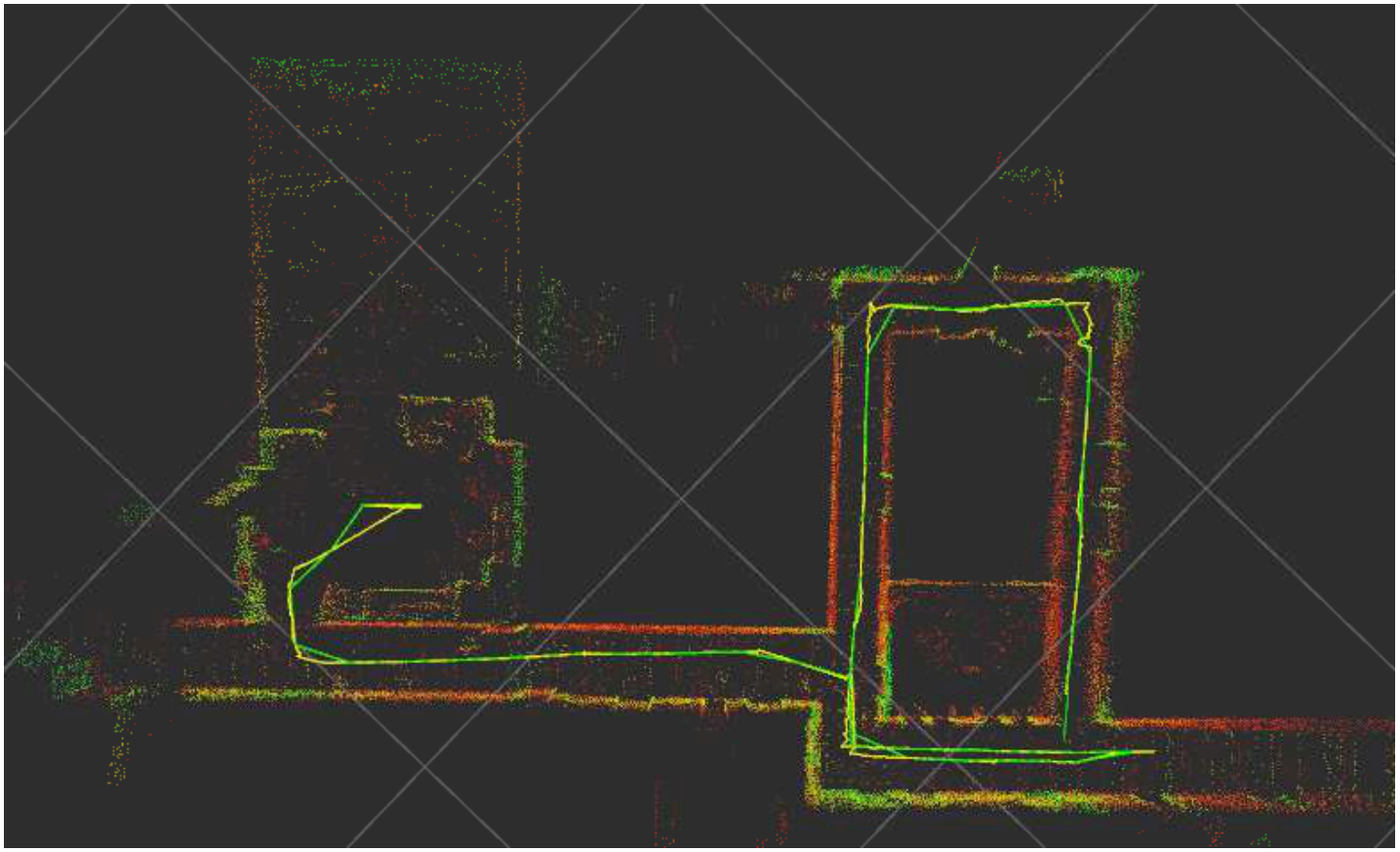
Map attempt 2



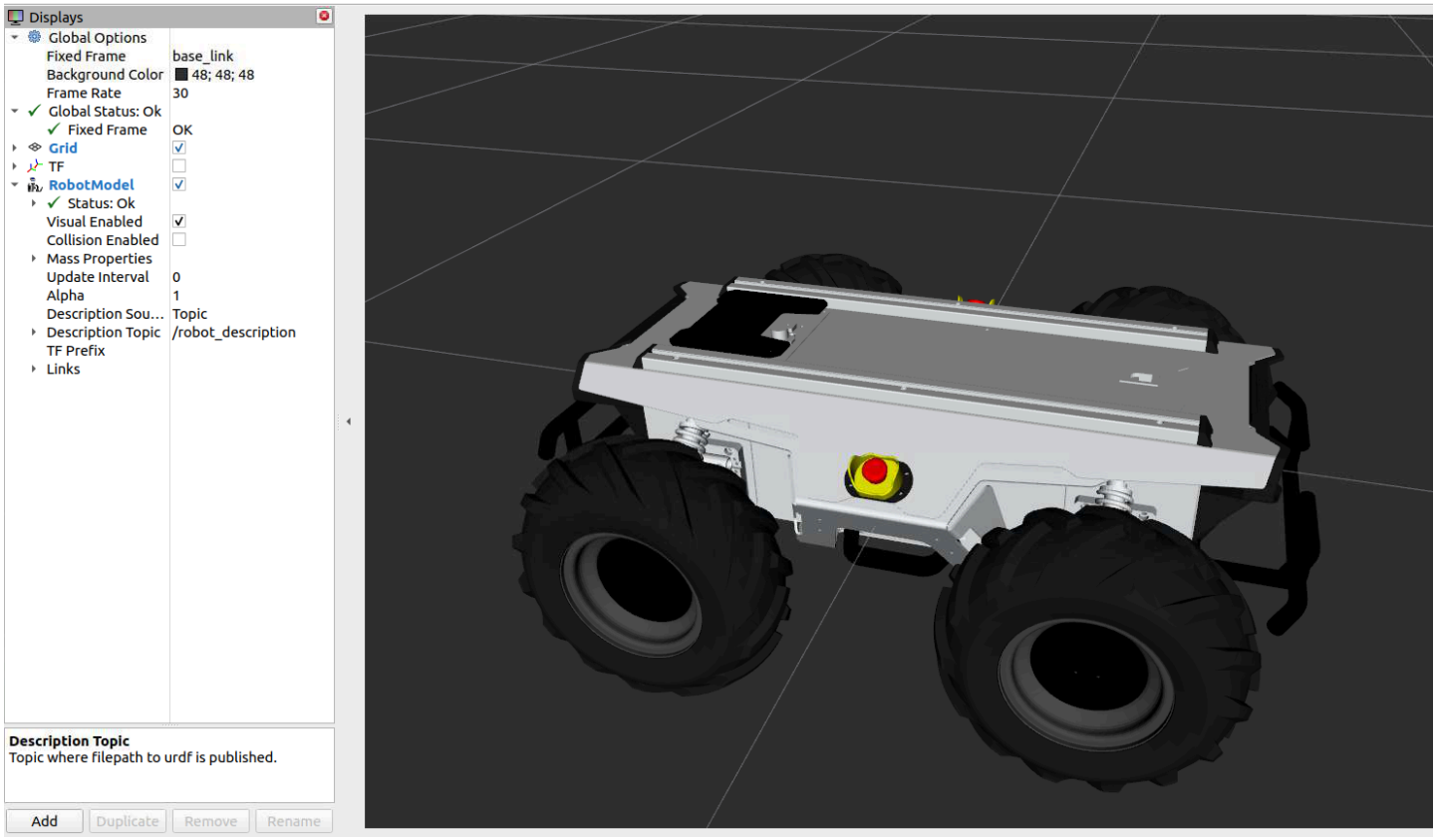
Map attempt 3 (with added lidar blind spot)



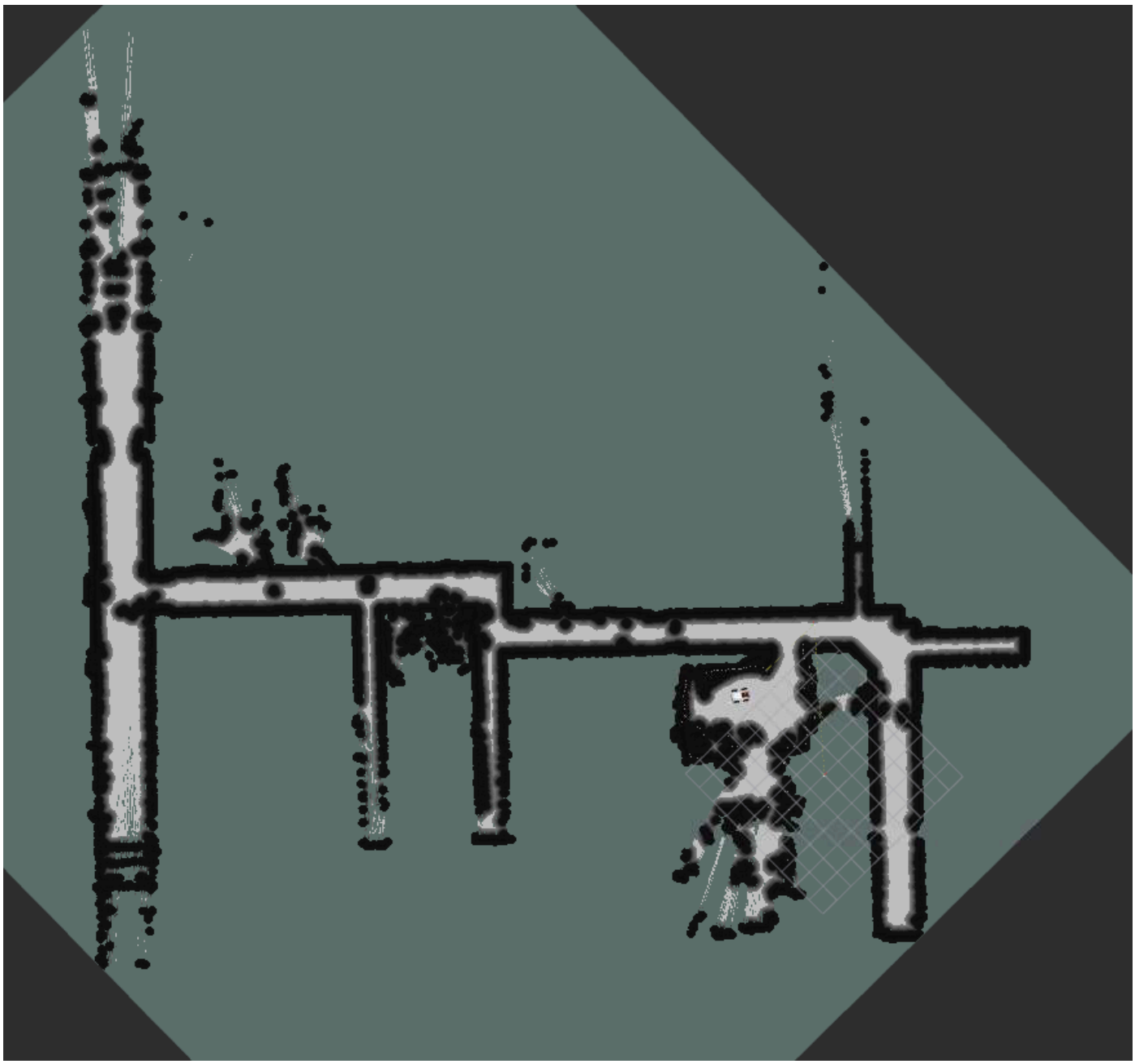
Improved v1



Scout 2.0 3D model visualized in RVIZ



Slam-toolkit map



Scout_mini