# Air Duct Mapping Robot System

Jacob Massaro, Sean Hall, Sebastian De La Cruz

### Department of Computer Science, University of Central Florida, Orlando, Florida, 32816-2450

### *Abstract* — This paper presents the implementation of a robot system designed to aid the University of Central Florida’s Utilities and Energy Services Department in air duct inspection and cartography. The robot features a continuous track drive system on the top and bottom, that couples with a scissor lift to allow for traversal of both horizontal and vertical sections of duct in search of air leaks or damages. The system is designed to create an accurate 2D map exportable into Autodesk Revit using laser scan information from a RPLidar A1 in conjunction with ROS packages to provide additional dimension information.

### *Index Terms* — Computer vision, leak detection, robot programming, simultaneous localization and mapping

### I. Introduction

UCF’s Utilities & Energy Services (UES) department provides utilities and energy management services for over 7.8 million gross square feet of conditioned space [1]. Included in their energy management practices is the evaluation of the HVAC systems across campus. The majority of energy use across UCF’s main campus is HVAC related, and any damages or air leaks present in these systems severely affects their efficiency and as a result, the amount of energy necessary to operate them at normal conditions increases.

Locating leaks within these HVAC systems is a tedious and disruptive process. Within each building, sensors are located at certain checkpoints throughout the system and are used to measure pressure data for determining if there are any significant changes as conditioned air travels from the air handler. When a significant drop-off in pressure is seen, a leak is assumed, and technicians now must locate it within the section of duct located between the two checkpoints where the pressure drop-off was noticed. However, long stretches of duct and inaccurate information about the layout of the duct complicate the issue. Technicians must remove ceiling tiles that follow the duct’s path in order to inspect it, often having them move in and out of various rooms as they attempt to locate the leak for repair. When the ducts undergo changes and repairs, they aren’t always updated in the layout mapping due to a variety of reasons (e.g. unknown presence of pipes or other obstructions in the middle of proposed path require building around them), and over time, especially for older buildings, this can cause the documented layout of the system to differ from the actual layout. This might require technicians to access rooms they had not planned to have entered to inspect the ducts and can cause more delay if these rooms are occupied when they need to access it (e.g. class being held within the room, the room is a office or lab and being used, etc.). Since they cannot be certain about the rooms they need to access or exactly how much duct lies between the two checkpoints, planning an inspection becomes a laborious task.

The proposed solution to help expedite this process is the creation of a remote operated robot, capable of navigating through the ducts to create mapping data that can be imported into Autodesk Revit so that an accurate layout of the system can be reconstructed. This robot must be able to traverse through obstacles with the ducts and be able to climb vertical sections of duct if necessary. It must also be able to withstand the environment conditions within the ducts, including but not limited to: gradually decreasing corridor sizes, dew point temperatures, high humidity, changes in traversable material (stainless steel, galvanized sheet metal, fiberglass insulation, etc.), uneven surfaces, wet surfaces, and wind speeds of up to 1600 feet per minute. In addition to this, the system also comes with an onboard camera system that can be used while controlling the robot or to automatically notify a technician via the control panel that it might have detected a leak using computer vision techniques. The control panel that accompanies the robot is a fast and responsive browser-based application that is independent from the operating system used, which includes control via an Android phone.

### II. Mechanical Design Overview

As the dimensions of the ducts and the materials they are composed of vary throughout the system, possible solutions involving magnetic or suction cup feet became unavailable. As a result of this, the mechanical engineering team designed a continuous track system on the bottom and top of the robot. This gives better traction when moving as opposed to the use of wheels and are more suitable for the types of terrain the robot will be traversing. The design also features a scissor lift system that allows the robot to extend and tension itself against vertical corridors to allow for vertical movement. The greater surface area of the tracks helps to prevent it from puncturing holes or causing damage to the ducts as it is moving upwards. The tracks and scissor lift system are powered by Nema 17 stepper motors that are powerful enough to support the weight of the robot and the amount of force it creates while tensioned out during vertical movement. A central chassis houses the motherboard, the sensors and the control boards used to communicate with the motors. Finally, an LED light strip is attached to the front of the central chassis which will be used when a user is manually controlling the robot to view the ducts through the camera. A tether is also attached to the back of the robot which provides the bot it’s power for movement and an ethernet port for communication with end user’s system. Each motor is controlled by a Pololu Tic T5000 stepper motor controller. These boards communicate with the Arduino boards using the I2C communication protocol in conjunction with the Pololu Arduino which allows for precise velocity and positional control modifications.

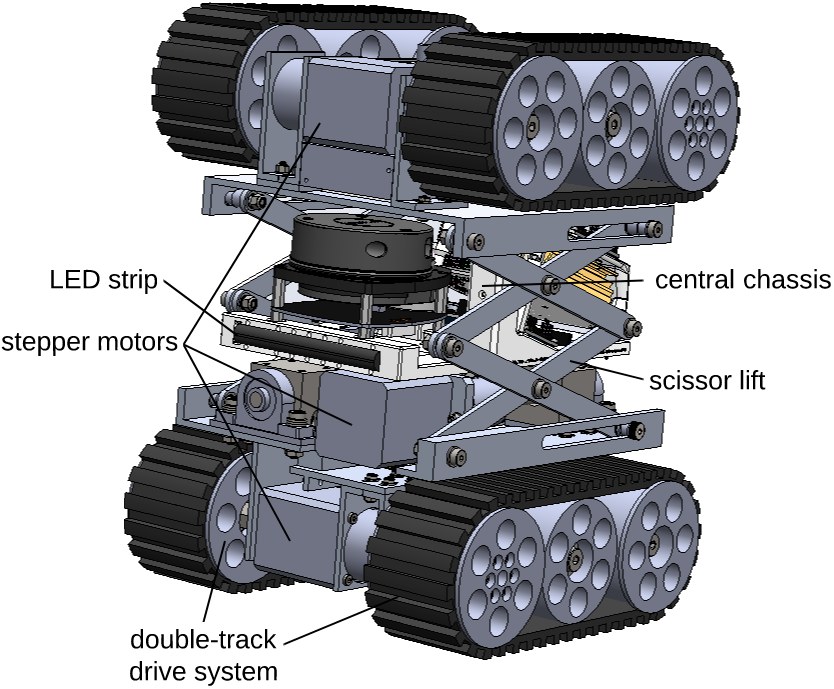


Fig. 1. Finalized robot design with key design features shown.

### III. Hardware

The board chosen for use in this project is the NVIDIA Jetson Nano. This board was chosen due to its tiny form factor, fast processing speeds, and its greater GPU performance compared to similarly priced boards in its price range. The robot will be processing a large amount of data so it is critical that it should be powerful enough for the required tasks and the addition of any future software integrations or while under heavy load. Two Arduino Uno R3 boards are also used to control the stepper motors that drive the track system and scissor lift system accordingly. These boards allow for communication between the Nano and the stepper motors with regards to movement commands. The Arduino Uno R3 was chosen due to its reliability and its large amount of online support articles and documentation. Typically, any issue that we encountered during development with the Arduino already had multiple solutions, workarounds, and tutorials online that we referenced to resolve issues at a moderately fast rate.

An RPLIDAR A1 sensor is also mounted on the central chassis and is used to generate 2D mapping data that is to be exported for map creation. It is a low-cost yet reliable solution for generating point-cloud information about the environment. It can scan points in a 360-degree range and works by emitting infrared laser signals that is reflected off objects. Positional information about the environment is determined by measuring the time it takes for the signal to return to the lidar sensor after being sent.

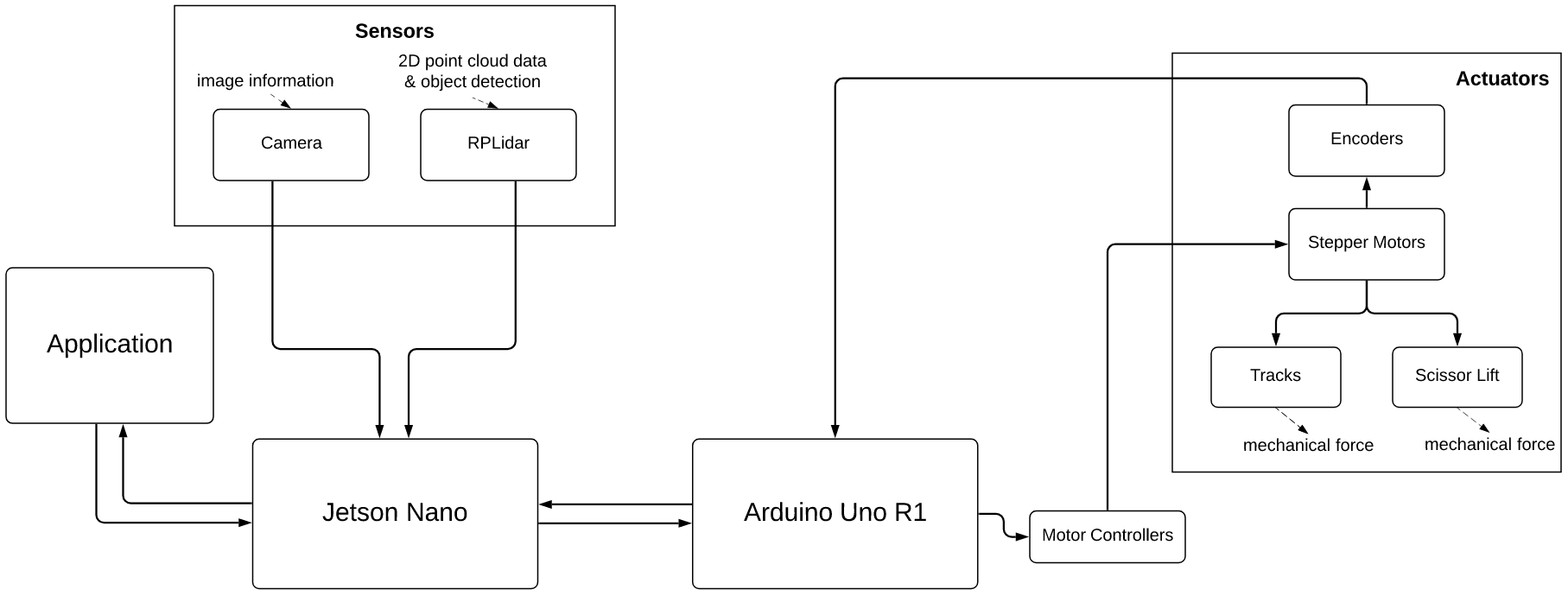
Finally, a camera is attached to the top of the central chassis which will be used when a user requests to view inside the ducts or to detect areas that might possibly be a leak. These hardware components and the system modules they interact with is highlighted in Fig. 2.

### IV. Algorithms

*A. Mapping with SLAM*

In order to create a map representative of the interior of the ducts, two differing types of information are required: distance of visible objects from the robot and the robot’s odometry (positional) data. Fortunately, this information can be obtained and utilized via a SLAM (simultaneous localization and mapping) algorithm. SLAM algorithms utilize various sensor information to continuously generate a map of the surrounding area whilst also keeping track of the robot’s position within that generated map.

A plethora of SLAM algorithms exist that can be used in virtually every use case. In the scope of this project, a 2D SLAM algorithm will be used via the slam\_gmapping package for ROS as the RPLidar A1 produces 2D data. The odometry data provides the algorithm with an accurate estimate of the robot’s position within the environment and allows it to ignore certain extreme data points received by the lidar that could misrepresent the robot’s actual position. This ROS package has a signification amount of documentation and as such is straightforward to set up. The lidar communicates its data to the algorithm via a ROS topic, which is a simply a callable bus that transfers data between ROS nodes. An odometry and laser scan topic are both generated and “transformed” (modified based on the robot’s current position) before being sent to the SLAM algorithm. The algorithm then stores the map as a bag, a filetype recognized by ROS that is typically used for analysis or visualization, which is then taken and created into an .pgm file via the map\_server package in ROS which is then converted to a .png file using the Python Imaging Library, which is a format that can be imported into Autodesk Revit.

  
  
Fig. 2. System overview of the robot system.

The use of a lidar sensor for SLAM algorithms is not necessary, as other types of sensors such as ultrasonic sensors can be utilized to provide positional information of the robot and its environment and was even considered for this project. Unfortunately, problems with accessing the GPIO pins on the Jetson Nano and utilizing its corresponding issue-filled library proved this path to be too difficult to implement.

*B. Leak Detection*

Another one of the main algorithms used in our design is the leak detection algorithm which to estimates when a hole or leak might be present within the current camera frame. When the robot is in its remote operated state, it grabs a frame from the camera at when prompted in search of any possible leaks. There are two implementations of this method: one that requires the usage of a thermal camera, and one that utilizes a normal camera in conjunctive with the LED light strip. In the case a thermal camera is used, pixels are color-coded based on the temperature at that location. In the presence of an air leak, the temperature of the air surrounding the leak will be noticeably higher, and as such can be detected by edge detection algorithms by checking for the differences in color. In the event a non-thermal camera is used, the algorithm will look for differences in brightness (assuming the inner ceiling has light that can leak into the duct via a hole) to determine possible leaks. Fig. 3 details this process and the output of the algorithm.

We implemented a Canny edge detection algorithm with a double threshold that will detect the areas mentioned previously in either case. When the algorithm is run, a Gaussian filter is applied to smooth out any potential noise present that could skew results and provide false positives. When a frame has been calculated to contain a leak, the image is sent to another algorithm that highlights the suspected leak area on the image with a red outline. Next, the initial algorithm either pushes a notification to the user via the control application, or flag the image as a leak area, storing the image along with the positional information of the robot in a locally hosted database for a user to inspect after the current run is complete. In either case, the output will either result in an image with the potential leak area highlighted being stored on the system or the image will be deleted in the event the algorithm does not detect a leak.

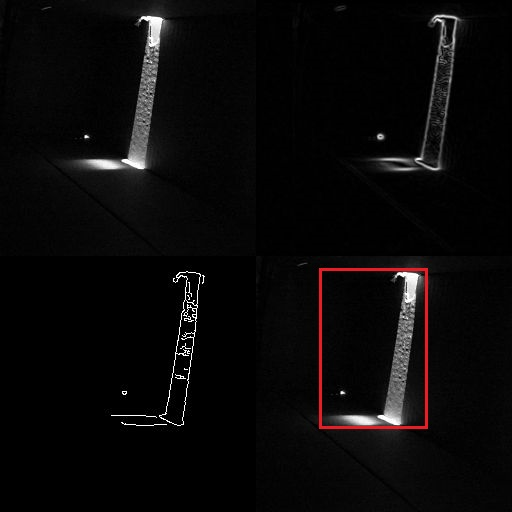


Fig. 3. Leak detection algorithm run in a sample environment where a non-thermal camera is used and changes in light are detected. The algorithm first takes a frame from the camera (top left) and then applies a Gaussian filter to filter out noise and highlight edges where changes in light occur (top right). The image is then passed though another filter (bottom left) before it is determined if it is a possible leak or not. If it is, the algorithm will return the original image with a red box highlighting the suspected area (bottom right).

### V. Control Application

For the end user to control the robot, a lightweight web application is used and is hosted locally on the Jetson Nano. A web application is used as it is fast, OS independent, and simple to develop using HTML, CSS, and JavaScript. In order to interact with the motors via the control application, commands received by the control application must be ROS-readable. To do this, we incorporated the rosbridge\_suite package to allow for ROS functionality for our non-ROS based control application. This package creates JSON based commands out of input received from the control application that is readable by ROS.

When the control application is launched, a startup button can be clicked on by the end user to launch a shell script on the motherboard that creates the ROS environments necessary for the sensors and motors to be usable via the appropriate ROS node. Included in this list are the rplidar\_ros, rosbridge\_suite, rosserial\_python, and slam\_gmapping packages.

The application contains an intuitive and easy to use GUI so end users can quickly become familiar with the system. It features a bar that indicates the current connection status between the application and the robot, a camera feed, and a series of control panels and settings that allow for track movement, image capturing, scissor lift control, map viewing access, and a panel to display what each key binding does is implemented to allow to more precise control to the user.

Since the control application is hosted locally on the Jetson Nano, a method to access it remotely via the end user’s computer is required. To accomplish this, we used the NoMachine software program, which allows us to gain remote access into our Jetson Nano from virtually any device, including Android (iOS is excluded due to disk access limitations by Apple). NoMachine also allows us to facilitate rapid file transfers between the remote host (robot) and the connection device (end user machine).

The main drawback to using NoMachine is that it requires downloading the software on the end user’s machine, where the user then must create a static IP address in order to gain access to the robot via the ethernet tether. Two of the main advantages in using NoMachine include the ability for it to allow any type of laptop or phone to work with our robot and the ability to have on-the-go fixes. NoMachine gives the user the freedom to fix the robot if it breaks down in the ducts without having to pull it all the way out as the end user will have access to the desktop environment onboard the Jetson Nano. This also made testing easier as we did not have to constantly take the robot out of the duct and hook it up to a monitor to see what had gone wrong.

### VI. Testing

During development, testing was split into two parts: simulated and physical testing. This was necessary as the designed robot was not machined and assembled until the later stages of development.

A simulated testing environment was created using Gazebo, a 3D simulator that recreates realistic testing conditions using their extensive library of robot models and environments, sensors, algorithms, etc. One of the major benefits of using Gazebo is how easy it is to translate simulated ROS code into ROS code that is used by the physical design using the gazebo\_ros package. ROS files that work in Gazebo also worked during physical testing without any major changes which reduces the amount of time spent debugging. The major flaw that came with using Gazebo, however, was the amount of time necessary to learn the software as it is complex and time-consuming to recreate the various conditions this project is designed to function under. Some aspects of the finalized design such as the scissor lift system is not supported by Gazebo and as such, we could not test all functionality within the software.

Our simulation test consisted of a simple two-wheeled robot with the same lidar sensor used in the finalized design mounted on top as shown in Fig. 4. The simulation served as a method to test movement commands and laser scan information processing when placed within an air duct. Publishing the odometry and laser scan data was accomplished during simulated testing in addition to cardinal directional movement and turning.

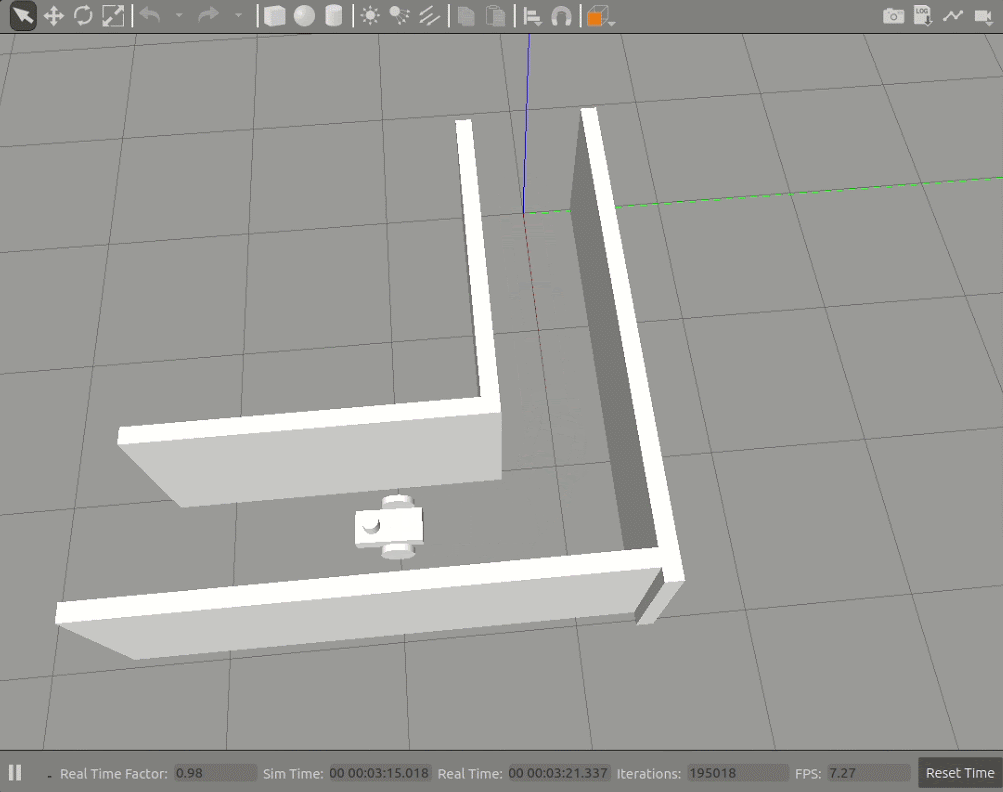


Fig. 4. Test robot in a Gazebo simulation successfully having turned around a corner.

Physical testing was split into two phases: connection testing and integration testing. For connection testing, a small robot was constructed using a variety of spare parts along with the Jetson Nano and an Arduino board. This test robot featured the same differential drive system that the finalized design would use via the continuous track system, albeit with different motors. This early stage of physical testing was designed to establish communication between sensor information, and the two boards for movement. This stage also gave us insight into how the wiring connection between the boards and breadboard (used for prototyping electronic circuits) would look like using jumper wires.

Once the finalized robot was assembled, we were able to begin complete physical testing and integration with the addition of a sample air duct corner that the mechanical engineering team created. This stage included testing the scissor lift system, tether attachment, vertical movement, and horizontal corner traversal in addition to the tests completed with both the simulation and smaller test robot.

We were able to use the test air duct provided by the mechanical team to test various parts of our system such as in Fig. 6. The addition of the test air duct was vital to our testing phase and it provided us with the necessary environment to test the mapping and the motor controls. We were also able to test vertical movement within this duct. Initially, we tensioned out the system and then flipped the duct to see if it held, then once we succeeded there, we drove into the duct, tensioned out, flipped the duct, and tested driving up and down vertically. After we succeeded in all these steps, we moved on to testing the transition from horizontal to vertical. This phase of testing proved to be the most difficult as the test duct we used had a vertical entry height just a couple cm taller than that of our robot when the scissor lift system is compressed to its lowest allowable height. This resulted in the robot experiencing slippage as it attempted to tension out during its transition from the horizontal to the vertical shaft of the duct. With some fine-tuning and orientation adjustments, we were able to complete the transition with human interference. Although this test required us to adjust the robot’s position manually, this should not have been the case with a slightly larger air duct. we should have been able to complete the transition more seamlessly.

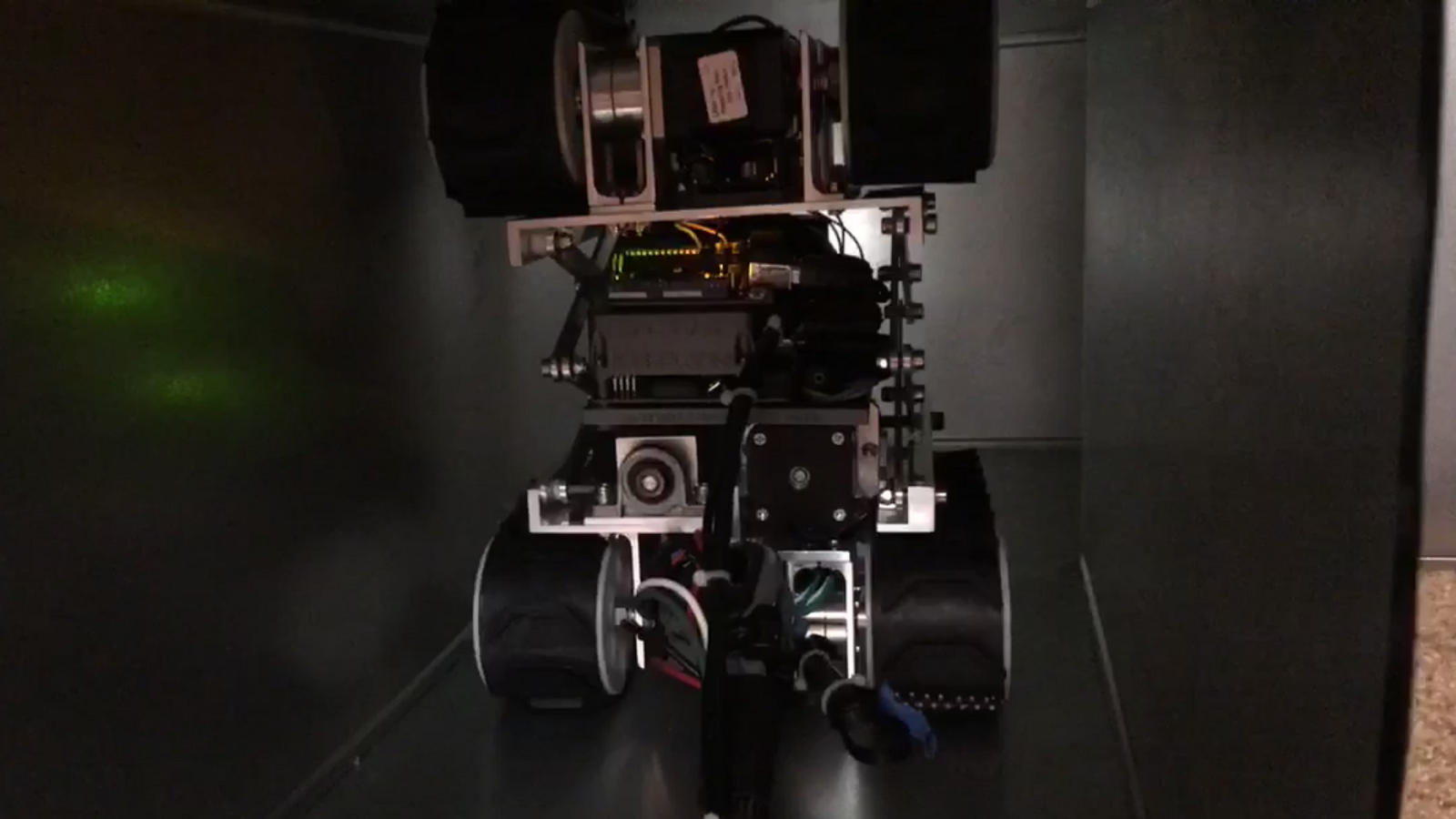


Fig. 6. Finalized robot design undergoing testing within the test duct. In this test, the robot prepares to navigate around a corner as it maps the duct.

To ensure the map generated by the laser scans produced an accurate map, we used RViz, a 3D visualization tool for ROS that allowed us to view the generated map in real-time, which is shown in Fig. 7. This tool allowed us to view how our mapping algorithms were working and their level of accuracy. It was also used during integration testing to ensure the lidar scanner was scanning within the constrained degree range (we constrained the range of view of the lidar to only map the area in front of it rather than behind or directly to its sides as the presence of the scissor lift system and central chassis behind the lidar sensor would have impacted the results of the scan produced).

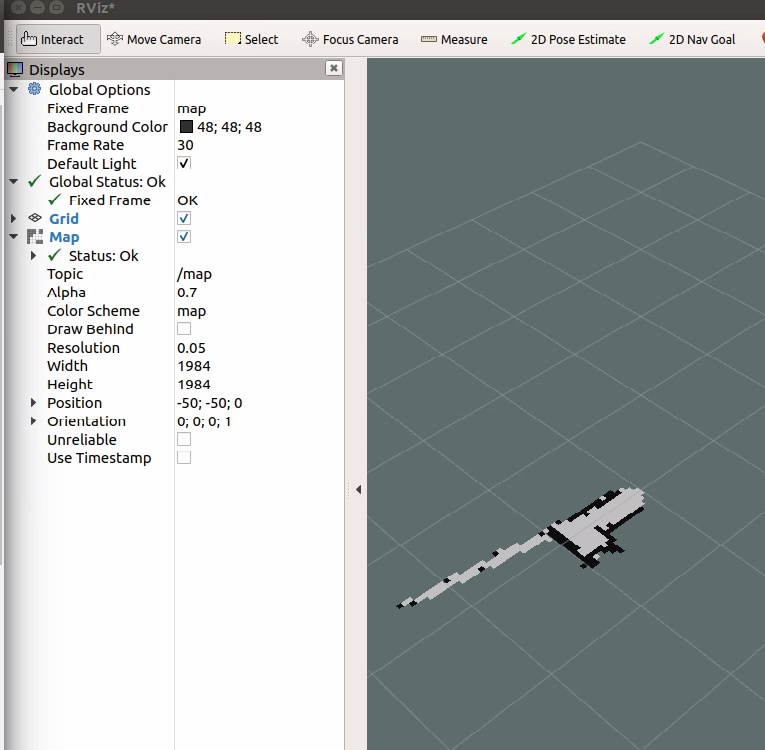


Fig. 7. A 3D visualization of our test duct environment within RViz. The narrow portion of the map protruding from the corner indicates that a tiny hole has been picked up by the laser scan. The presence of these can also be used to assist in the search for any leaks found in the system.

VII. Conclusion

Overall, we were able to successfully create a remote operated robot capable of generating and saving a map of its environment. During testing we created a map representative of the space within the test air duct, saved the image within a directory, and were able to travel horizontally and vertically within the same environment.

### References

[1] University of Central Florida. “Our Team.” Our Team – Utilities & Energy Services, energy.ucf.edu/about/our-team/