COLORADOSCHOOLOFMINES
EARTH • ENERGY • ENVIRONMENT

CSCI 370 Final Report

Stratom, LLC.

Brady Veltien
Ethan Ko
Lucas Niewohner
Ryan Lopez

Revised June 15, 2023

CSCI 370 Summer 2023

Prof. Donna Bodeau

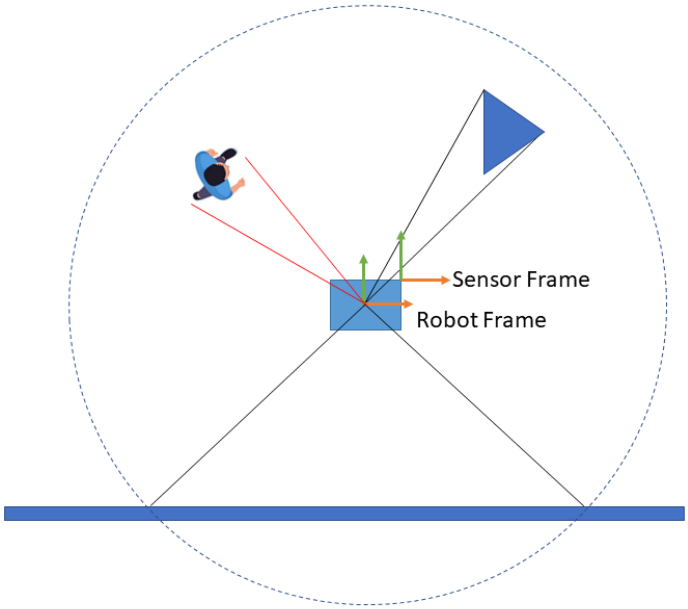
Table 1: Revision history

Revision	Date	Comments
New	May 17, 2023	Completed Sections: I. Introduction II. Functional Requirements III. Non-functional Requirements IV. Risks V. Definition of Done XI. Team Profile Appendix A – Key Terms
Rev-2	May 25, 2023	Completed Sections: VI. System Architecture Revised Sections: Appendix A – Key Terms
Rev-3	June 1, 2023	Completed Sections: VII. Software Test and Quality VIII. Project Ethical Considerations Revised Sections: Appendix A – Key Terms References
Rev-4	Jun 12, 2023	Revised Sections: VII. Software Test and Quality a) Modified intro section and added results to each test. Added Sections IX. Results X. Future Work (started) X. Future Work XI. XI. Lessons Learned
Rev-5	Jun 14, 2023	Revised Sections: The entire document was revised for grammar and clarity. Added Sections VII. Technical Design
Rev-6	Jun 15, 2023	Revised All According to Peer Review

Table of Contents

I. Introduction

Stratom Inc. is an independent contractor that designs, builds, and maintains autonomous systems that assist in the military's logistics network. They are currently developing a suite of autonomous and semi-autonomous robots for transporting, loading, and unloading 10,000-pound pallets in off-road environments. Stratom utilizes LIDAR (Light Detection and Ranging) sensors to allow these vehicles to detect objects near the robot. These sensors return the distances of objects in a 2D plane around the sensor. The goal of this project was to develop a proof-of-concept system that transforms data from these sensors to detect and classify obstacles within a given range around the robot (known as a "safety curtain"). Essentially the robot must be able to scan a given radius in the environment and provide an output of all obstacles within that range. This project was designed to give the team experience creating a crucial piece of software that allows autonomous vehicles to detect the world around them. Figure 1 below shows a visual concept of the project.



Safety Curtain Warning Message Array
- Warning Msg 0
  - Header: Timestamp, FrameID
  - Min Angle: 50°
  - Max Angle: 65°
  - Distance: 8m
  - Dynamic: False
- Warning Msg 1
  - Header: Timestamp, FrameID
  - Min Angle: 117°
  - Max Angle: 140°
  - Distance: 10m
  - Dynamic: True
- Warning Msg 2
  - Header: Timestamp, FrameID
  - Min Angle: 203°
  - Max Angle: 312°
  - Distance: 12m
  - Dynamic: False

Figure 1: Example Safety Curtain Detection (left) and Output (right)

To accomplish this task, the team was given a set of tools to work with. We must write a system in the Robot Operating System, version 2 (ROS2), a middleware suite that connects components of a robotic system together using loosely coupled "topics". We must also use YAML (Yet Another Markup Language), a markup language akin to JSON, to configure our project. Finally, our project must produce useful visualizations to show objects from the robot's perspective and visually confirm that the system is functioning correctly. More specific requirements are listed in sections II and III.

Our work is available online at https://github.com/Hermanoid/StratomCurtain.

II. Functional Requirements

Note: Because our team will be designing our system using ROS, we do not need to know what our inputs and outputs will be connected to, only that we produce our output correctly. Thus, these functional requirements will not contain any information about the external system that we must interface with.

- The system must take in information from a ROS 2 system, including:
  - LIDAR scan data from, at a minimum, one simulated source.

    - LIDAR scan data from, as a stretch goal, multiple simulated or physical sources.
    - Spatial data via ROS 2's TF2 (TransForm System) library. TF2 is the *de facto* system for handling location information in ROS.
- Users of the system must be able to configure aspects of the system using a YAML file, including:
    - Input and output topic names (as listed above).
    - Curtain area polygon shape and size relative to a configurable frame of reference, such as the robot or the sensor itself, using TF.
    - All other parameters that might reasonably need to be changed, such as detection thresholds.
- The system must identify potential collisions with objects up to 20 meters from the vehicle reference frame.
- The system must distinguish between static and dynamic obstacles.
- The system must publish a warning message indicating where in the safety curtain there is an infraction.
    - Array for multiple detections.
    - Min and max angle and closest distance of the object relative to curtain reference frame.
    - Timestamp and TF Frame Id for the detections' reference frame.
    - Indication of static or dynamic obstacles.
- The system must also incorporate a visualization utility, built on ROS 2's RViz2 Visualization system.
- The system must feature Gazebo simulation for testing. Gazebo is a commonly used robot simulation platform that allows systems to be tested before they are put on a physical robot. Stratom recommended we start with the out-of-the-box Turtlebot simulation configuration.

Some elements that are notably **excluded** from the Functional Requirements:

- Handling of failure cases like power failure or sensor data loss. It can be assumed that the code is being run on an ideal operating robot.
- Exact particulars of how the detection and tracking algorithm should operate. The project is given some liberty to decide what should happen when, for example, a dynamic object is occluded and, when it comes back into view, it has stopped moving. These situations should be navigated using two goals:
    - Classifications should be made based on what would be most helpful to a robot attempting to use this information to avoid collisions.
    - Whatever it does, it should be highly configurable.
- What hardware requirements the system should meet. The team was given no processing power constraints except that the algorithm must be able to run on our laptops. One notable requirement is that no cloud processing may be used.
- An interface for interacting with the system. The system is allowed to be relatively rudimentary in the sense that it needs only to perform its job once launched, with no real-time configurability or interactivity.

III. Non-Functional Requirements

- Code shall be built on the Humble distribution of ROS2
- Code shall be written in C++ or Python
- System shall safety curtain detections at a minimum of 10Hz
    - No specific requirements are placed on processing performance, except that it must be able to reasonably run on a developer's laptop.
- Code shall be maintained and shared via GitHub.
- The system must be thoroughly documented such that a new user can easily make use of it.
- The system must incorporate a test plan that thoroughly exhibits and proves the system's functionality in all reasonable cases and edge cases.
    - The system must, as a minimum, be able to pass all these tests in a Gazebo simulation.

- The system may, as a stretch goal, be able to pass all these tests using sensor data from a physical robot.

IV. Risks

At the beginning of this project, one general risk was the limited level of experience the team had with ROS (Robot Operating System) and robotics in general. Three out of four team members were completely new to ROS. Only one team member had extensive experience with it. Because of this, it was possible that the team would fail to predict the level of difficulty and time requirements of the project due to their general inexperience. To mitigate this risk, the team leveraged pre-existing solutions where available to aid in software creation within the required time frame. Additionally, the team relied on the guidance of the one team member who had previous experience creating ROS projects.

Currently the project faces risks related to object detection in the context of vehicle automation. The software is designed to aid in self-driving navigation, and any failure of the software to detect obstacles or properly respond to detected obstacles will put people and property around the vehicle in danger. Stratom's robotic vehicles are large military vehicles designed to transport tens of thousands of pounds of cargo. It is paramount to minimize the risk of these vehicles running into anything. To mitigate this major risk, the team designed a test plan (see section VII) to ensure the basic functionalities of the software are working correctly. The team was unable to fully test the software because of the short duration of this project, but it's important that future developers working on this project continue to test it to minimize this risk.

V. Definition of Done

The system will be considered "done" when:

- A test plan has been produced and documented which will demonstrate all functional and non-functional aspects of the project.
- All elements of the test plan can be proven to pass. At a minimum, they will pass in simulation, though as a stretch goal Stratom would like to test them on a physical robot as well.
- Stratom is provided with a repository of all code written for the project and a Dockerfile that, once built, can easily run the system and replicate shown results.
- Stratom is provided with documentation overviewing the system, detailing all interfaces and how to use it.

VI. System Architecture

The system architecture for this project is straightforward because it is a standalone project, with no need to integrate with any existing system except for the input of a standardized ROS2 LIDAR topic. The goal of the system is to take in this LIDAR scan information, from either a physical robot or simulated source, and manipulate it to be able to classify objects within the safety curtain. The two potential solutions the team came up with are described below.

The first solution the team discussed leveraged the expansive Point Cloud Library (PCL). PCL is segmented into a series of modular libraries that are used in 2D, 3D, or even hyper-dimensional image and point cloud processing. From a very high level, PCL offers many complex algorithms and classes that can segment points into objects that would allow the team to track and publish obstacles as desired. The diagram below provides a high-level overview of the ROS2 nodes and topics that would be implemented in this solution.

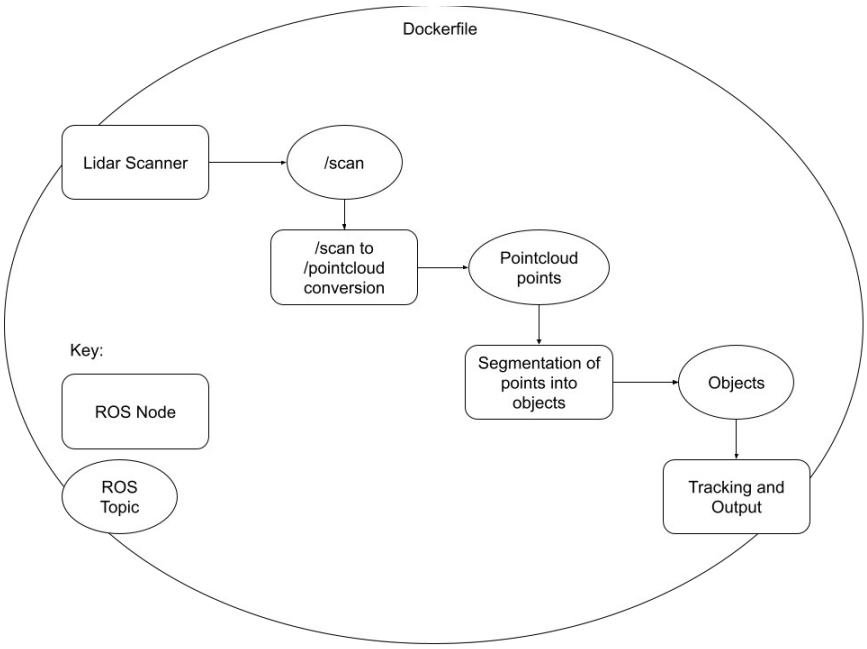


Figure 2: PCL System Architecture

The second solution the team discussed was to use existing utilities from the Nav2 stack to generate a Costmap from the LIDAR scan. A Costmap represents the world as a raster image, where each pixel represents a few square centimeters of space. Using this approach, a Costmap Converter algorithm can then be applied to detect objects inside of the safety curtain. Some challenges to this approach include an out-of-date codebase, greater complexity, and higher processing requirements. However, the Costmap Converter package includes many applicable algorithms that had already been designed, tested, and debugged. The system architecture flow using the Costmap and Costmap Converter is shown in Figure 3 below.

Figure 3: Costmap System Architecture

The team worked through both approaches in parallel to determine which would generate the most accurate results while being relatively easy to implement. In conclusion, the team chose to implement the Costmap solution. Some of the specifics of this solution are discussed in the following section.

VII. Technical Design

Nodes and Topics

Since this system with is built in ROS, it is built on nodes which take in data from and publish data to "topics". These topics represent streams of information such as LIDAR data which can be freely subscribed to and published to by nodes. This loosely coupled communication system affords us a unique amount of modularity which can't be found in most data processing software. Since the nodes can work independently of each other and transmit data in a node-agnostic way, our system allows for components to be swapped in and out without affecting the overarching design.

For example, if a node exists that can turn a LIDAR's input into a polygon, the system's final node can still be perfectly functional as it is built only to take a set of polygons to form the centroid tracking algorithm. This has also allowed the team to work on nodes concurrently, since we can work on one node using "faked" input data until the intended source of that data is completed. This allowed for py_tracker to be developed concurrently with the costmap converter node, for example. Thanks to this modularity, the system can remain airtight if a more optimal way to perform a specific task is found. The straightforward nature of this is demonstrated in Figure 4 which shows how this data is seamlessly transitioned downwards as each node performs its task.

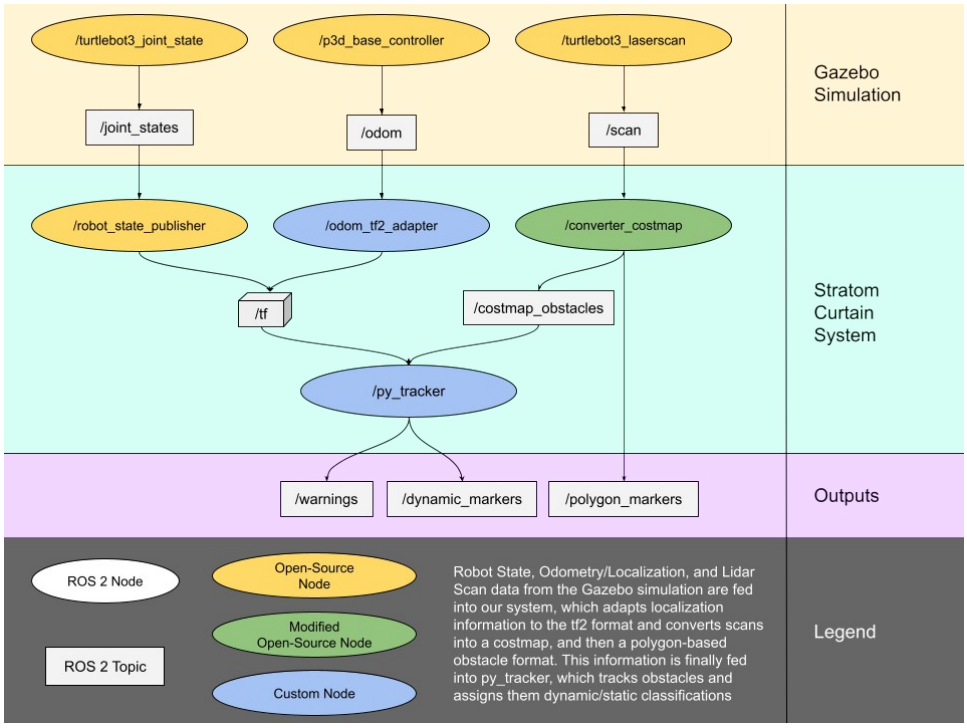


Figure 4: Topic Structure

Centroid and Tracking

For tracking individual objects, the team has selected an algorithm which calculates the centroids of objects and applies what is known as the Hungarian algorithm. To track object polygons between frames, the Hungarian algorithm is applied to match polygon centroids between the current frame and the previous frame. This algorithm is depicted in (Figure 5).

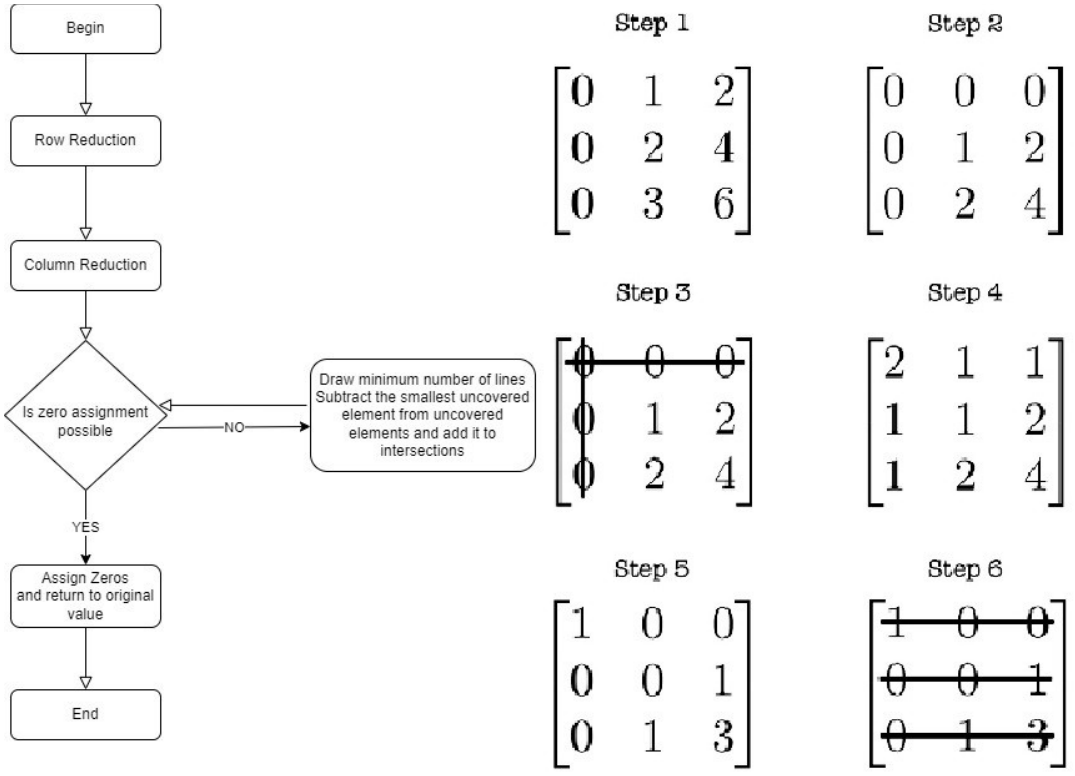


Figure 5: Hungarian Algorithm

In our specific case this algorithm checks the locations of every centroid in the current frame and compares each one to every centroid in the previous frame. Via the Hungarian Algorithm (Figure 5), each centroid is correlated between frames. These correlations can then be used to transfer Object IDs between frames, creating a form of tracking. Once these correlations have been established, the distance between each centroid is divided by the time between frames to calculate the current velocity of that object.

Because the LIDAR sensor has a natural amount of roughly-gaussian error, the centroid points calculated tend to "shake" quite a bit as the perceived outline of objects moves around randomly. This necessitates the application of several filters. The bouncing-around of centroids is largely centered on the actual location of the object, so by taking a rolling average of the velocity vector (including direction), much of this shaking can be cancelled out. Next, a time-based filter is applied to ignore brief movements which are usually associated with movements of the robot itself and errors in localization. Finally, a speed threshold is applied to filter out objects that appear to be moving very slowly, and dynamic/static classifications are applied.

VIII. Software Test and Quality

Virtual Testing Environment

Each test described below was conducted in a simulated environment created in Gazebo. The tests in simulation were completely ideal and served as a sandbox to build our software in an optimal environment while disregarding many realistic variables. For simplicity, the objects being detected were cubes, cylinders, spheres, and other unrealistic whole shapes.

In Person Testing Environment

The tests detailed below were also recreated in a real world setting at Stratom's headquarters. ROS bags were used to save the data collected from a real LIDAR sensor that was attached to a teleoperated vehicle. The main detection objects were humans and rolling trashcans.

The real-world testing environment presented new challenges that the team didn't experience in simulation. The biggest challenge stemmed from the assumption that the robot would always be operating on a perfectly flat surface. The team noticed that when the robot was on uneven ground in a real-world environment, it would detect the ground as an object.

Secondly, let it be noted that the team was unable to test a full 360-degree field around the robot. Instead, a 180-degree field directly in front of the robot was tested. However, our system for ingesting LIDAR information is based on a battle-tested open-source Costmap implementation that is designed to adapt to many input situations. It is thus the team's assumption that the software would behave the same when operating in a 360-degree field as it did in the 180-degree field.

Lastly, the LIDAR sensor used was extremely accurate, and its detections were much more realistic than could be recreated in simulations. This created some additional problems which are discussed in specific tests below.

Test 1 – Base Case

Test Description

The purpose of this test is to verify that the basic requirements of the problem are met. A dynamic object and static object are detected and categorized correctly.

Procedure/Needed Objects

- One dynamic object moving perpendicular to the robot starting within the safety curtain.
- One static object placed within the safety curtain at the beginning of the test.

Expected Output

- Safety curtain message array containing 2 objects.
- One correctly detected static object with corresponding information.
- One correctly detected dynamic object with corresponding information.

Diagram

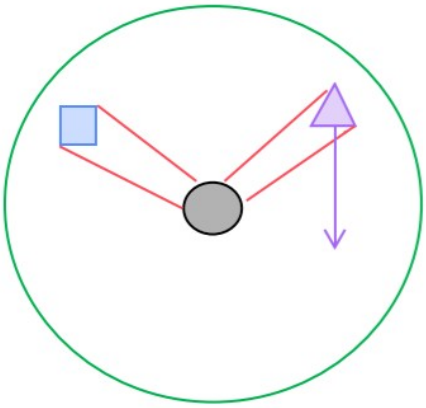


Figure 6: Test Case 1 – Base Case

Simulation Results

a) Stationary Robot: Test passed as expected.
b) Moving Robot: Test passed as expected.

In Person Results

This test was conducted in a mostly empty parking lot, with a stationary trashcan representing the static object and a rolling trashcan as a dynamic object.

c) Stationary Robot: Test passed. Obstacles in the background had a considerable amount of noise that caused the algorithm to incorrectly identify them as dynamic, but obstacles within the safety curtain were correctly identified.
d) Moving Robot: Test passed. Just as with the stationary robot, background objects were noisy, but in-curtain objects were classified as expected.

Test 2 – Dynamic Object in front of Static Object

Test Description

The purpose of this test is to ensure that when a dynamic object is passing in front of a static object that the static object maintains the same object ID without interference from the dynamic object in front of it.

Procedure/Needed Objects

- One larger static object (such as a wall) is behind a smaller dynamic object. Both objects are within the safety curtain range.
- The dynamic object will move in-between the static object and the LIDAR

Expected Output

- Safety curtain message array containing 2 objects
- One correctly identified static object with corresponding information
- One correctly identified dynamic object with corresponding information

Diagram

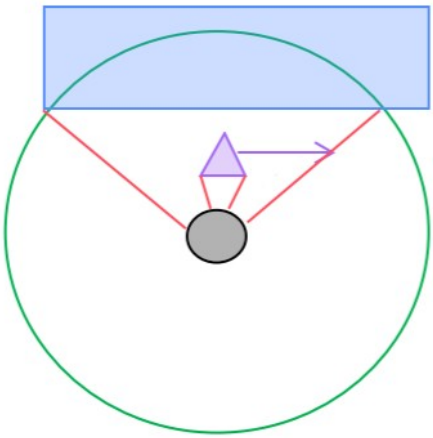


Figure 7: Test Case 2 – Dynamic in front of Static

Simulation Results

a) Stationary Robot: Test failed. The wall was recognized but misclassified. The centroid tracking algorithm constantly found a new center point on the wall because the scan wasn't consistent, so the wall appeared to "move" and was therefore misclassified as dynamic.
b) Moving Robot: Same as above.

In Person Results

The robot was set up perpendicular to a wall and a trashcan was rolled in between the wall and the robot. There was an elevation difference between the robot and the trashcan.

c) Stationary Robot: Test failed. Just as in the simulation test, the background objects have significant noise, and the "shadows" cast by the foreground object occluding the background object only contribute to that noise, causing the system to identify the wall as dynamic.
d) Moving Robot: Same as above.

Test 3 – Dynamic Objects on Intersecting Paths

Test Description

The purpose of this test is to ensure that when two dynamic objects cross paths both objects are correctly identified before they cross each other, and then correctly re-identified after.

Procedure/Needed Objects

- Two dynamic objects within the safety curtain that take intersecting paths, but do not collide with each other.

Expected Output

- Safety curtain message array containing 2 objects with corresponding information.
- Both correctly identified as dynamic objects before their paths cross
- Both correctly re-identified as dynamic object after their paths cross

Diagram

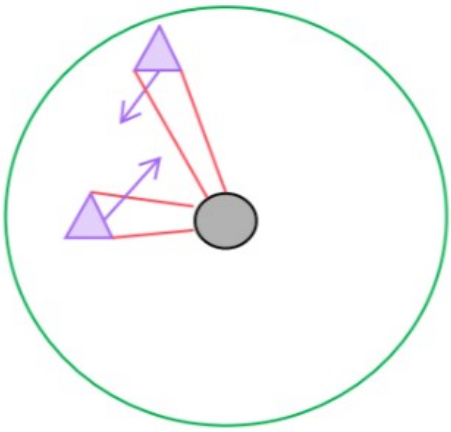


Figure 8: Test Case 3 – Intersecting Paths

Simulation Results

a) Stationary Robot: Test passed as expected.
b) Moving Robot: Test passed as expected.

In Person Results

This test was conducted in the parking lot. Two rolling trashcans would cross paths in front of the sensor.

c) Stationary Robot: Test passed, with some caveats. The tests were performed by physically pushing a large object (a trash can) in front of the robot. The LIDAR was extremely sensitive and picked out human legs as they walked, causing the system to identify them as objects when in view. This resulted in inconsistent detection, where the number of tracked objects varied with time.
d) Moving Robot: Same as above

Test 4 – Boxed Robot with a Dynamic Object

Test Description

The purpose of this test is to ensure that the LIDAR scanner within a boxed environment (such as a room) can detect the walls around it and a dynamic object within that area.

Procedure/Needed Object

- 4 long static objects that create a box around the robot, all within the safety curtain
- 1 dynamic object inside the box, also within the safety curtain

Expected Output

- Safety curtain message array containing 5 objects (depending on implementation)
- Four correctly identified as static objects with corresponding information
- One correctly identified dynamic object with corresponding information

Diagram

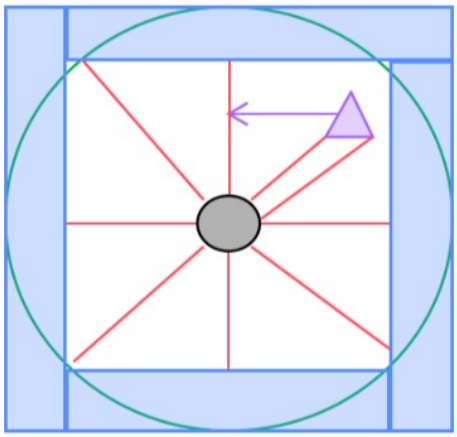


Figure 9: Test Case 4 – Boxed Dynamic Object

Simulation Results

a) Stationary Robot: Test Failed. The dynamic object is detected, but the four walls were not identified correctly.
b) Moving Robot: Same as above.

In Person Results

This test was impossible to recreate given our testing environment. There was not a small enough room available.

c) Stationary Robot: NA
d) Moving Robot: NA

Test 5 – Objects Changing State

Test Description

The purpose of this is to explore the behavior when a dynamic object moves into the safety curtain and then becomes static, and when a static object within the safety curtain becomes dynamic.

Procedure/Needed Object

- 1 dynamic object that begins outside of the safety curtain range

    - The dynamic object will move into the safety curtain
    - The dynamic object will then become static
- 1 static object that begins with the safety curtain range
    - The static object will become dynamic and move out of the safety curtain range

Expected Output

- Safety curtain message array containing 2 objects with corresponding information
- A static and a dynamic object are correctly identified at the beginning of the test
- When each object changes states, the safety curtain message will update identifying both objects as dynamic.

Diagram

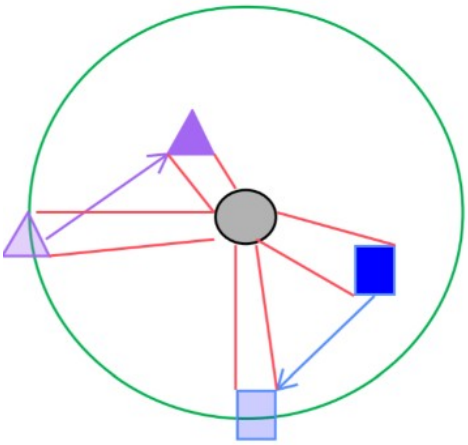


Figure 10: Test Case 5 – Objects Changing State

Simulation Results

a) Stationary Robot: Test passed as expected.
b) Moving Robot: Test passed as expected.

In Person Results

This test was conducted inside the warehouse and in the parking lot. The team sporadically walked in and out of the curtain, moving and standing still to mimic objects changing state.

c) Stationary Robot: Test passed, although the warnings returned were a bit sporadic with detections of the team's legs as the objects were pushed.
d) Moving Robot: Test passed with some caveats. The poor localization information available to the robot caused all objects around the robot, including static objects, to appear to move. This same movement caused the object that stopped moving to be still classified as dynamic until the robot itself stopped moving.

IIX. Project Ethical Considerations

The software has relatively limited ethical considerations. The most predominant ethical consideration related to the project has to do with ensuring that the system works as expected when it is deployed. This is especially true if the detection software is built upon by another system to create a collision avoidance system for a vehicle. Section 1.2 of the ACM Code of Ethics and Professional Conduct states to avoid harm. Harm in this context can be identified as injury or destruction of people and property brought about by the system. An example of this would be if the detection software fails to identify an object, and the vehicle it is attached to runs into somebody or something. As the creators of this software, it is our job to take special care to ensure that our product works as intended before deploying it to situations in which it could cause harm.

An additional ethical consideration of this project relates to the complications related to automated technology replacing human workers. This detection software can be used to directly replace ground crew members working alongside large vehicles. Because of this, the team needs to consider ACM 3.1, which states that engineers must always ensure that the public good is the focal point of development. Therefore, the system must bring about more benefits to the public good than the alternative. In conclusion, the solution should be able to quantifiably justify replacing human workers for it to be ethically supported.

IX. Results

It has been determined that the software has met the basic requirements provided by Stratom. The team conducted a series of tests, both virtually and in person, to determine that the solution can produce the desired output of warning messages of detected objects that are inside of the safety curtain. Please see Section VII for further information on specific testing results. It is noteworthy that while the solution does meet the minimum requirements, many edge cases and scenarios exist that cause the software to not have 100% accuracy. This is specifically true when testing in more realistic scenarios when more obstacles are present. Several tweaks need to be made to the existing solution to get more accurate results. The team was unable to implement the stretch goal of implementing multiple 2D LIDAR scans, but the foundation has been set to ensure the software can easily be modified and built upon to achieve this goal.

For those interested in our work, it is available online at https://github.com/Hermanoid/StratomCurtain.

X. Future Work

If future work is to be done on this project to meet the stretch goal of multiple LIDAR sensors or to refine the existing program, multiple steps can be taken to ensure it will be an easier process. To get multiple LIDAR sensors the existing costmap converter node has its own configuration file which can be edited to ensure that it takes a specific amount of LIDAR inputs and creates a map that correlates to those inputs with the curtain still anchored in the center of them. Additionally, if the tracking algorithm wants to be refined or the costmap approach is not to the client's liking, the py_tracker node which contains most of the tracking and dynamic classification of objects is built to take in a list of polygons. This means that if those who wish to work on this project would like a different approach it can be easily changed. If that approach produces polygons, the software should still generate all objects without needing to rewrite another node in addition to the costmap converter. The py_tracker node will only require a change to what topic it grabs those polygons from, and it should work cohesively. This will still require an understanding of ROS and its message type

but does not require an understanding of tracking. All that is required is a method or way to convert a frame into polygon data for the node to use.

The astute eye may have noticed that there were a few failing tests amid the test suite. We did not address these tests because to do so would require a dramatic modification of the tracking node. Thus, future work could include this dramatic rewrite. The team learned many lessons in designing and testing the centroid-based tracking algorithm, and while it performed quite well in many scenarios, these lessons could apply to a smarter tracking system in the future. For the benefit of anyone working on this project in the future, we used these lessons to lay out a potential modification plan below.

First, the system could be considerably improved by modifying the costmap_converter node to produce more, larger polygons. The converter repository does contain a ready-made plugin for constructing concave representations of objects. The team did not have luck with this plugin because, by tweaking the available parameters, we could only attain two undesirable results. When the concavity parameter was increased, the polygons hugged the data too closely and tended to oscillate heavily with error. With lower concavity, the algorithm would throw out small objects. Because the team did not have much time for this project, we were forced to use the convex polygon algorithm with a relatively low maximum object size to shrink the massive polygons that would otherwise be formed around walls. A modification to this algorithm, however, could get the best of both worlds.

Alternatively, another instance of the costmap_converter node could be implemented to leverage the RANSAC-based line-finding plugin to find and filter out long, linear sections of walls from the normal convex-polygon handling system. These walls were the cause of most of our test failures. Regardless of what approach is used to improve polygon detections, an improved polygon-detection system should make centroids less sensitive to error out-of-the-box by removing the long connecting segments that result from wrapping objects with a convex polygon. This may make dynamic/static classification better out-of-the-box.

In addition to improving the costmap_converter algorithm, the object-detection system could likely be made considerably more robust with a rewrite of the tracking algorithm. To do this, the team proposes throwing out the centroid-tracking system and following one of two routes:

1. Track objects using their full polygonal representation, rather than just their centroid.
2. Track objects using the entire Costmap, to have access to more map information.

The team believes that the polygon tracking approach, while not as robust as the polygon approach, would still be considerably more robust to common edge cases than the centroid approach. An algorithm that tracked using polygon overlap could intelligently ignore the growing-and-shrinking that generally results from Lidar error, while still responding to the movement of the entire polygon, where the size of the polygon remains roughly constant.

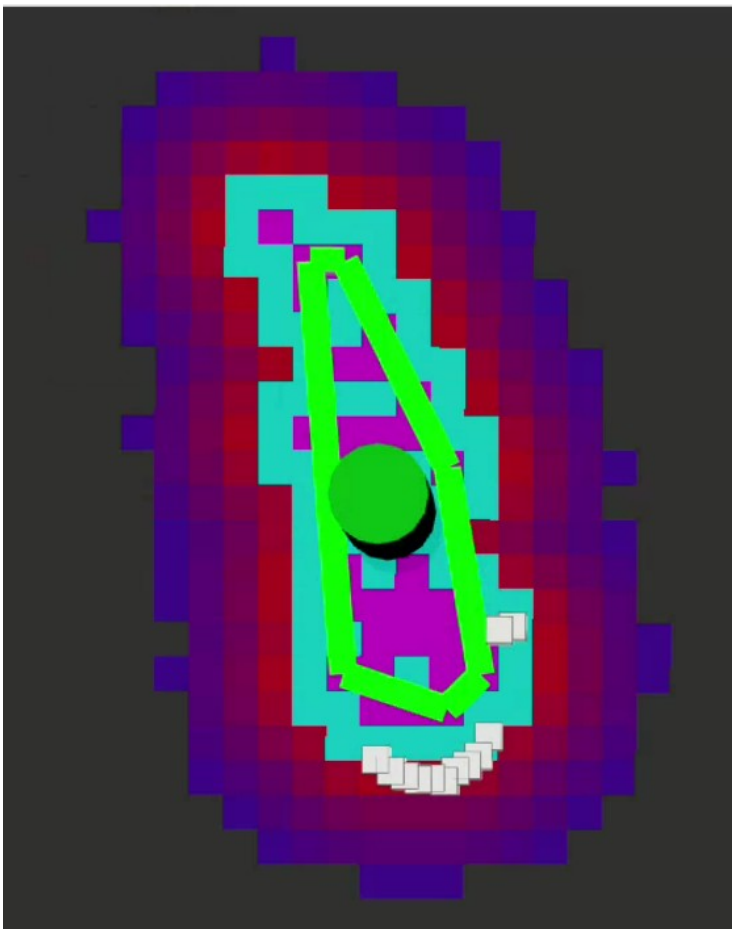


Figure 11 - An object moving downwards, towards the LIDAR sensor. This represents a difficult edge case for determining if an object is moving.

To express the utility of Costmap tracking, it's necessary to introduce a new edge case. For effective obstacle detection, it's necessary to map the entirety of an object, not just the portion that the robot can actively see. However, there is a downside to mapping. Any mapping algorithm, including costmap_2d, can only confidently mark an area of the screen as "cleared" of all objects the robot can definitively see that that area is clear. In this edge case, shown in (Figure 11), an object is moving downwards towards an offscreen LIDAR sensor. The area where the object is entering is being marked as "filled", but because the object blocks the LIDAR from seeing behind it, the mapping algorithm cannot confidently mark the area behind the object as "clear". Thus, the object appears to grow in length when it is not actually growing at all. The polygon approach would not understand this situation and would throw it out, interpreting that the robot is only seeing sides of an object that it couldn't see before. For example, the shape shown in Figure 10 could have easily come about if the LIDAR sensor were moved around the side of an oblong object.

To get around this edge case, the team believes it's necessary to use a second option; to perform image processing on the entire Costmap, rather than to track using only polygons. This option, while more difficult than polygon tracking, also gives a tracking algorithm access to an extra "state" that space can take. With polygon tracking, only two states are available: "filled", for regions contained by polygons, and "empty", for all other space. With only these two states, it's not possible to differentiate between growth caused by seeing a new side of an object and growth caused by an object moving out of an area that the robot cannot see. However, Costmaps contain three states: "filled", "empty", and "unknown", where unknown represents any space that hasn't been seen before.

A Costmap-based tracking algorithm could use image processing to look for changes in state between two frames of the Costmap image. Should a filled area – and thus, an object polygon – grow into an area that was previously unknown, the algorithm would know to ignore this movement. However, if an area that was previously empty becomes

filled, or an area that was previously filled becomes empty, that would be a definitive sign of object movement. Thus, an algorithm could intelligently handle the edge case portrayed in (Figure 11).

XI. Lessons Learned

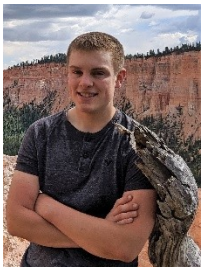
- Robot Operating Systems is a powerful open-source middleware that allows for the efficient transfer of data through topics. Topics are an anonymous publisher/subscriber system that makes it easy to add additional processes to a program.
- LIDAR sensors can map out an environment, but they can struggle with noise. Dust and rain are two big hazards that can create a lot of noise that will need to get filtered out. High end LIDAR sensors are capable of detecting objects very accurately, and the simulation could not replicate the real-world environment accurately.
- Costmaps are a great tool for mapping and remembering an environment. Nav2 and costmap_converter are two open-source libraries that implement them for robot operating systems.
- There are many different approaches to tracking detected objects from point clouds. Several different algorithms were considered before deciding to implement centroid tracking via the Hungarian algorithm.
- Point Cloud Library has a lot to offer with many ways to process point clouds. It can transform point clouds into other objects such as clusters and polygons. It also has great documentation which makes it easy to use.
- Whiteboards are an excellent tool for coding. They helped with the agile method as a calendar tool and Todo list. Whiteboards are also great for planning the implementation of code and visualizing how potential solutions may work.
- Pair programming is one of the many advantages of working in a group. It helps solve problems faster with collaboration and increases the quality of the code.

XII. Acknowledgements

Many thanks to Kristin Farris and Charles Best from Stratom, our technical advisor Donna Bodeau, and the team behind costmap_converter Christoph Rösmann, Rainer Kümmerle, David Dudas, and Albers Franz.

XIII. Team Profile

Brady Veltien
Senior
Computer Science / M.S Engineering & Technology Management
Hometown: Longmont, Colorado
Work Experience: Information and Security intern at ABC Fitness, experience in Android application development
Activities: Mines Varsity Baseball
Hobbies: Playing & Watching Sports, Skiing, Reading, Video Games



Lucas Niewohner
Junior
Computer Science – Robotics and Data Science
Hometown: Herman, Nebraska
Work Experience: Robotics Engineer, IoT Systems Developer, Website Developer, and App Developer
Clubs: Robotics Club (MATE Underwater Robotics Competition), Marching Band (Percussionist)
Hobbies: Skiing, Biking, Rock Climbing/Bouldering, Board Games, and Studying Abroad.

I love solving complex engineering and interpersonal problems to create systems that help people in their everyday lives. I also love a good, social game of Minecraft.

Ethan Ko
Senior
Computer Science
Hometown: Fort Collins, Colorado
Work Experience: Mines Service Center Consultant, IT Specialist
Hobbies: Running, Biking, Disc golf, Video Games, and Rafting

Ryan Lopez
Senior
Computer Science
Hometown: Centennial, Colorado
Work Experience: Mines Service Center Consultant, Lighting Designer
Hobbies: Reading, Video Games, Drawing and Snowboarding

References

ACM. (1992). ACM Code of Ethics and Professional Conduct. Retrieved from https://www.acm.org/code-of-ethics

Rösmann, C., Kümmerle, R., Dudas, D., & Franz, A. (2020). costmap_converter. GitHub. Retreaved from https://github.com/rst-tu-dortmund/costmap_converter

Appendix A – Key Terms

Include descriptions of technical terms, abbreviations and acronyms

Term	Definition
ROS 2	*The Robotic Operating System. Not actually an operating system, ROS provides a system for sending data between the many subsystems/processes ("nodes") of a robot or system of robots networked together. ROS2 is the second iteration of this platform, and Humble is the most recent distribution of it (as of mid-2023)*
ROS Node	*A single process in a ROS network. A node can send and receive data via ROS's topic, service, and action channels, and is able to be configured by ROS's parameter system. A node typically accomplishes one task for a robot, like processing LIDAR information.*
ROS Bags	*Sets of data taken from real world robots that can be replayed on machines remotely and run through software without the need to run the software in real time. Essentially a snapshot of data inputs for testing purposes.*
YAML	*Yet Another Markup Language. Provides a human-readable format for serializing data. It is a superset of the more-popular JSON.*
RViz	*ROS's go-to tool for visualizing 2D or 3D data relating to the robot in real time.*
Docker	*A tool for creating and managing self-contained, isolated "containers". These containers encapsulate a program and all its OS dependencies so that it can be easily deployed without configuration or installation troubles.*
Dockerfile	*A file containing instructions for installing and building all the resources needed for an application. Once built, Dockerfiles become Docker Images, which can be executed to create a Docker Container as described above.*
LIDAR	*Light Detection And Ranging. Technically referring to a range of sensors that use lasers to measure distances, in this project it refers to a spinning implementation that returns distances of objects in a 2D plane around the sensors.*
TF/TF2	*ROS's built-in system for managing position and rotation information (TransFormations) in multiple reference frames. It allows multiple ROS nodes to share transformations in a distributed manner and provides utilities for accessing and manipulating this information.*
Costmap	*A created grid around a sensor that uses nearby information to construct a map of detected objects in the surrounding area*
PCL	*The point cloud library is an existing large scale open-source library of methods dedicated to image processing consisting of different libraries dedicated to different subsections of image organization including clustering and tracking*
ACM	*Association for Computing Machinery*