COLORADO SCHOOL OF MINES
EARTH • ENERGY • ENVIRONMENT

CSCI 370 Final Report

Stratom

Nathan Woo

Adam Bukres

Thuan Nguyen

Revised Dec. 6 2024

CSCI 370 Fall 2024

Prof. Donna Bodeau

Table 1: Revision history

Revision	Date	Comments
New	Sept. 1, 2024	Created initial document
Rev – 2	Oct. 20, 2024	Added software quality working plan
Rev – 3	Nov. 10, 2024	Added the results documents
Rev – 4	Nov. 21, 2024	Finished up final draft of document
Rev – 5	Dec. 6, 2024	Revised paper based on peer reviews

Table of Contents

List of Figures

I. Introduction

In many robotics' applications, including those at Stratom, accurate processing of 3D data such as point clouds is essential for precise object recognition and spatial awareness. Point clouds are basically a collection of data points in three-dimensional that are used to represent 3-dimensional objects and areas. A single point cloud would be a collection of thousands of coordinate tuples, each representing a point in space. When combined, all these points will display the scenario around the camera frame. One of the most common problems in computer visions that uses this idea of point clouds is template matching. Template matching is the process of taking in a scenario point cloud (i.e. a scan of a room) and a template point cloud (i.e. a scan of a chair) and finding the location of that template within the scenario. The practical use of template matching in real world scenarios, however, often requires extensive preprocessing to achieve consistent and timely results, which can be highly dependent on each situation. The challenge is to develop an algorithm that can autonomously match a template to a part of a large point cloud with minimal preprocessing, making it robust and adaptable to varied environments.

This project focuses on creating a robust 3D template matching algorithm for use in real world robotics applications. This algorithm is capable of taking a large point cloud of the world around a robot and accurately matching a template within that cloud. For example, imagine scanning an entire car, having a 3D model of a tire, and having the algorithm identify and segment just the tires from the larger scene without intensive and context-specific preprocessing. Although template alignment algorithms exist, achieving reliable performance in diverse and dynamic settings remains a significant challenge and is a large focus for this success criteria of this algorithm. We have developed an algorithm that accepts a template (either a point cloud, a 3D CAD file, or some other data type that students deem useful) and a scene point cloud, then matches the template to the scene. It should work with real-world and/or simulated data provided to us in ROS2 bags. The algorithm is implemented within a ROS2 node and will be benchmarked on an Nvidia Jetson provided to us for the duration of the project. ROS2 is basically just an open-source framework for building robotic systems and are the key to us implementing code that will work in Stratom's existing code stack. We have also built practical unit tests for their algorithms, a report on the accuracy of their algorithm, the processing time required to handle point cloud scenes, and the algorithm's ability to work in dynamic and varied environments.

Project Objectives:

- Research existing algorithms and tools for 3D template matching
- Develop an algorithm for template matching in point clouds that requires minimal preprocessing
- Implement the algorithm in a ROS2 node and deploy it onto an Nvidia Jetson.
- Test the algorithm on data provided in ROS bags, write practical unit tests, and report on algorithm accuracy and processing times
- Ensure the algorithm works consistently in varied and dynamic environments

II. Functional Requirements

- The algorithm should be structured inside a ROS2 node, with defined inputs and outputs
- System should listen to via ROS subscriber a PointCloud2 message type, containing the point cloud input to the algorithm
- System output should publish a PoseStamped message type representing the location of the target in space. The pose of this output should be in the same reference frame as the input point cloud.
- System output should publish one or more PointCloud2 message types containing information about the processed input point cloud. One of these point cloud outputs should visualize the final determined location of the template in 3D space. The frame of these point cloud(s) should be in the same reference frame as the input point cloud.
- System output should include execution time of the algorithm, either on a ROS topic or easily inspected via command line printouts.
- The system should take in a 3D template to align in the point cloud scene. This may be of any reasonable type (a point cloud, a 3D CAD file, or some other data type that students deem useful)

- The ROS2 node should be configurable via a yaml file, including information about:
    - ROS2 Pointcloud2 topic to listen to
    - Location of the template to load into the system
    - Algorithmic configurations (number of iterations to use in the algorithm, thresholds to use, etc)
- The algorithm should run at a usable time- target execution time is under 1 second.
- The algorithm should require minimal application specific preprocessing.
- Unit tests should be written to test the algorithm. Students may determine the best method to do so. The goal of these tests is to see not only that the code runs without error, but also that is "works" to an acceptable level.

III. Non-Functional Requirements

- The system should be written in C++ or Python
- The system should be written using ROS2 Humble
- Code should be maintained and shared via GitHub
- The system should be able to run in a Docker container based on a Stratom provided image
- The system should be deployed and benchmarked on an Nvidia Jetson, provided by Stratom

IV. Definition of Done

Our definition of being done is a product that meets the specified requirements of running in an efficient time frame and with minimal preprocessing. We knew our product was complete when these requirements were met, and we have successfully created the following deliverables:

- Report on existing algorithms and tools for 3D template matching. In this report, a few promising, easily stood up algorithms shall be tested against our problem set to determine if they are feasible for use. If not, call outs as to their insufficiencies should be included in the report.
- Development of a custom algorithm in a ROS2 node, as outlined above. This algorithm may make use of open-source libraries and tools, but their licenses must allow for alteration and/or use in commercial applications.
- System demonstration throughout the semester
- System Code, delivered via GitHub
- System documentation including architecture, bring up instructions, and dependencies

V. System Architecture

The input of our system is a point cloud template, and a larger point cloud scene. The point cloud template will be a .pcd file, which is the standard for any point cloud. The input of the larger point cloud scene will be from a rosbag, which contains various larger point cloud scenes at different time values. The rosbag may contain quite a few larger point cloud scenes. Our architecture will contain the following steps, also visualized in a flowchart below.

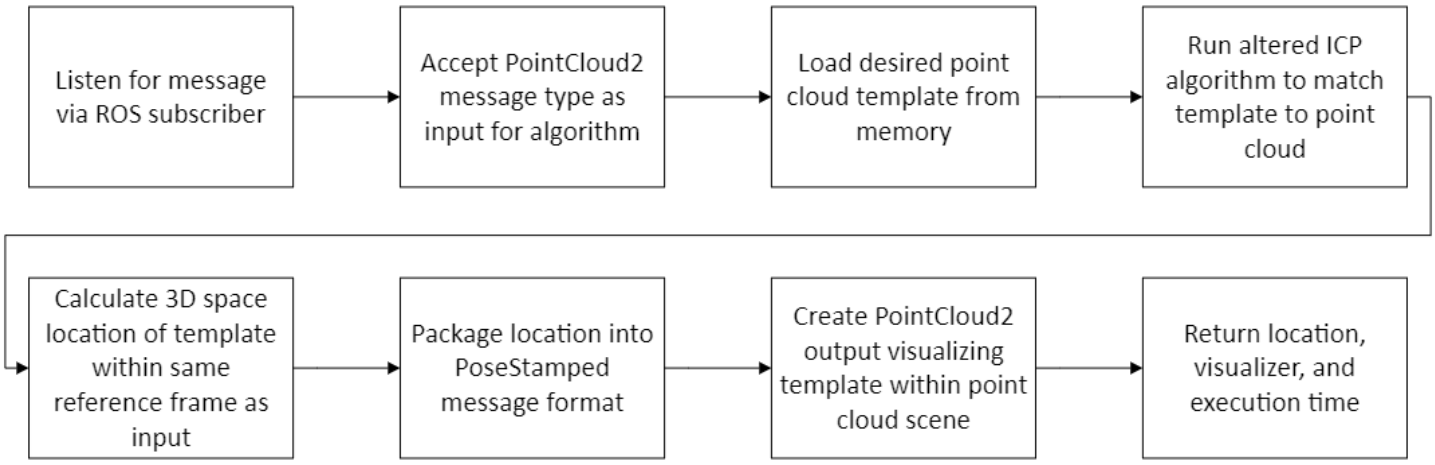


Figure 1: Process Flowchart

1. Read in the template point cloud that we need to find in a larger scene. Since this we know to be a .pcd file, we use a python library called open3D to read in point clouds and store them as objects.
2. Read in the scene point cloud from the rosbag. We do this with a ros node that subscribes to the topic the point clouds are published to. So, each time a point cloud is published to that topic, our node reads that point cloud and performs the template matching on tat input scene.
3. Run template matching algorithm to match our template to the larger point cloud scene. We were testing multiple algorithms, outlined below.
    a. Iterative closest point (ICP). ICP is a generic algorithm that usually matches point clouds of relatively same size. ICP iteratively finds closer and closer transformations by minimizing errors between closest points. Example of an ICP algorithm to match point clouds:

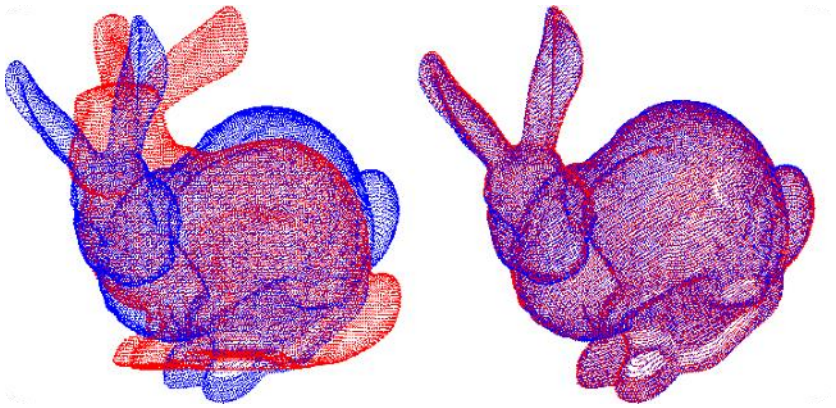


Figure 2: Iterative Closest Point (ICP) Example

    b. Modified iterative closest point. This modified ICP algorithm we created uses the directions normal to the points to have extra metrics to match to. We calculate the normal vector, which is the direction perpendicular to the surface of the points. Then we minimize errors on the normal vectors and the distances between points, so we use extra information to also match to.
    c. Feature extraction and matching. This algorithm calculates interesting points, or features on both the template and the scene point clouds. Then, we find the location that has the most matched features in the region, using RANSAC, a feature matching algorithm. This finds the transformation that best matches the features extracted earlier.
4. Calculate the 3D transformation, or pose, of the template within the much larger point cloud. Each algorithm does this slightly differently, so we need to reformat the 3D pose output of the algorithm we use.

5. Package the 3D transformation into a standardized format. In this case we use a standard homography transformation matrix, which we save a NumPy matrix that can be sent and used by other algorithms.
6. Write the homography matrix to a rostopic, so a different algorithm can read this homography matrix and know where the template is in the larger scene.

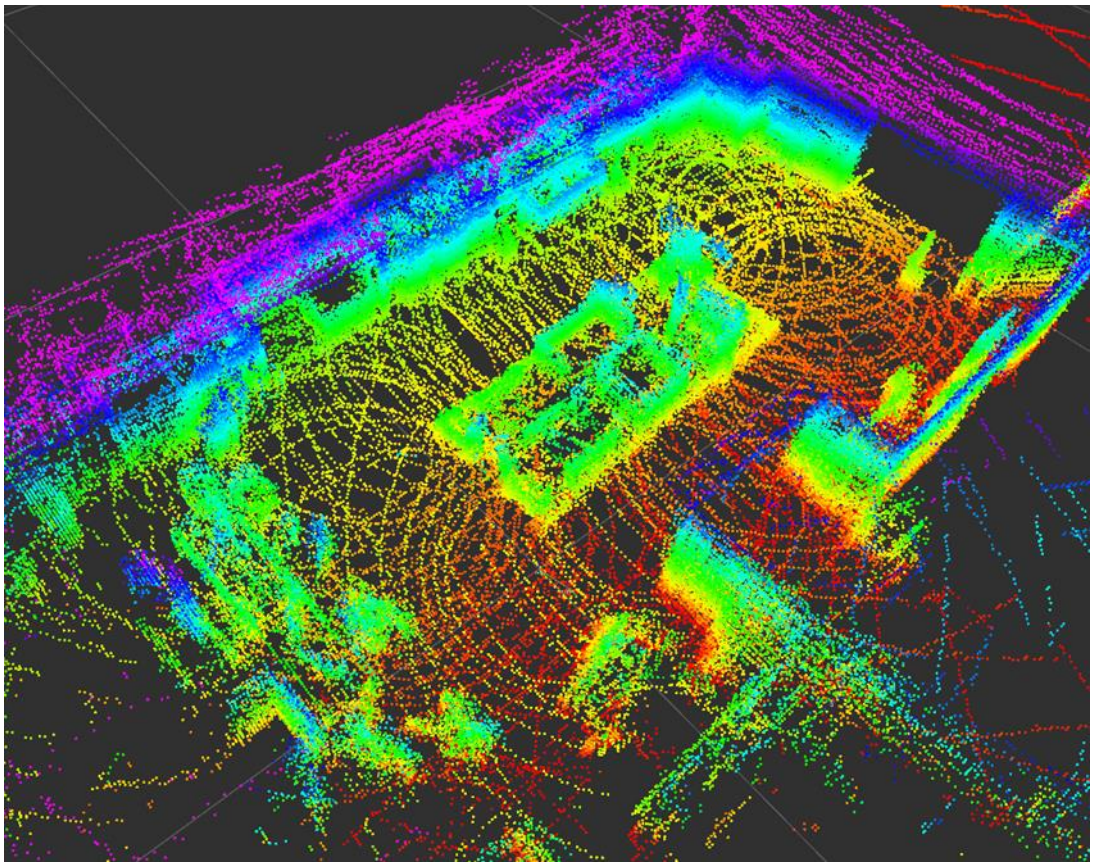


Figure 3: Representation of Point Cloud

VI. Technical Design

Our project mainly consists of just an algorithm that matches the point cloud of a template given to us and a point cloud of the scenario we are trying to match it into. The system designed around this algorithm is a modular ROS2 node, ensuring that we are able adhere to modern robotics software practices. This node processes input point clouds, perform our matching algorithm, and publish the results from that algorithm as a ROS2 topic, so a robotics system can read the information given and perform actions around that when needed. We should also include a docker file inside of our system to ensure it runs consistently across many different environments.

Other than just our system design, we needed to focus more on the design of our algorithm itself. The algorithm itself is composed of different parts that run sequentially to each other to ensure a more accurate result. There are two vital aspects of our design, the first global positioning algorithm and the second local positioning algorithm. Global positioning is used to determine the approximate location of our target in the scene, returning an initial transformation matrix that is applied. In our specific design, we used a Python RANSAC implementation to locate a template of the fuel tank in the scene. RANSAC, like many global positioning algorithms, uses a form of feature matching. This provided better results as the fuel tank is much larger than the cap and avoided getting stuck in local optima as there are less features similar to a rounded tank than to a flat gas cap. The RANSAC algorithm returned the following positions, on a whole truck and on a smaller scene with just the tank:

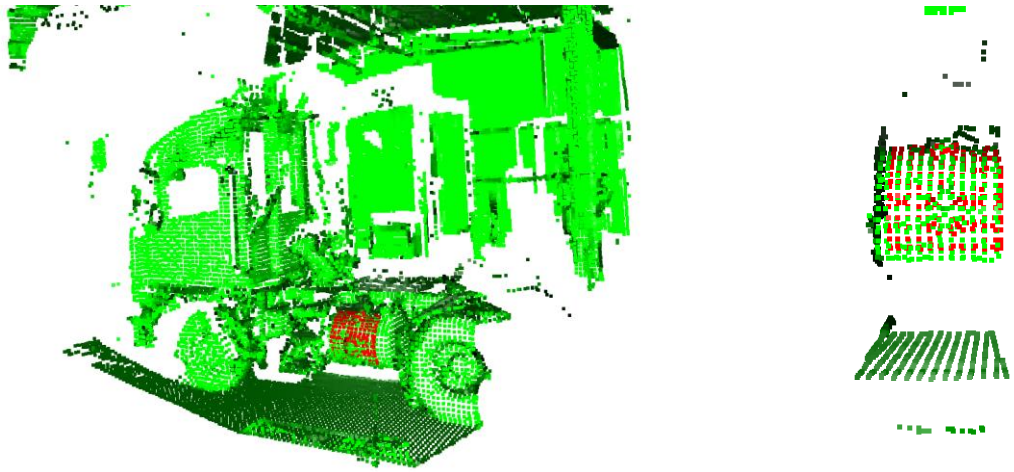


Figure 4: RANSAC Example

With these returns, we now have a general idea of where in the scene the gas cap is located, providing us with an accurate initial guess to pass to our ICP Registration algorithm, the local positioning algorithm. ICP calculates point-to-point correspondences, essentially mapping out all of the closest points in the template to those in the scene. With an accurate initial guess, ICP runs significantly faster and more accurately, making RANSAC an essential aspect of the design, but ICP does the accurate matching. Our program then takes the gas cap template and applies the transformation from RANSAC to bring it closer to the target. Then, ICP is run on the cap and the scene to provide an accurate final alignment, as shown below:

Figure 5: Final Result of Algorithm

VII. Software Test and Quality

Our software testing consists of unit testing, integration testing, code metrics, and code reviews. The following list describes how we performed testing of each of these types, and the value they provide for the quality of our software.

- Unit testing: We created unit tests to test the individual section of our code. We also wrote unit tests to test reading in data as a form of a point cloud, test the template matching algorithm, and test the output write pose.
- Integration testing: We ensure that the code we wrote can be integrated with our client Stratom's existing code stack. Stratom uses ROS2 to manage code in their stack, and we have tests to ensure that our code integrates with Stratom's ROS2 code stack seamlessly.
- Code metrics: The runtime of our algorithm needs to be evaluated as the algorithm must work real time, so we need tests/metrics to ensure the algorithm fits within an acceptable runtime.

- Code reviews: We had code reviews between us as a team to ensure high quality code is presented to the client, who may also do code reviews to ensure the code fits within Stratom's expected code standards.

The unit test that we have consists of two different types. These include the datasets that Stratom has already given to us and a custom set of tests that we have developed ourselves. The custom set of tests consists of much simpler tests that will challenge individual aspects of the algorithm. This should be much different than the point clouds received from Stratom due to its much smaller size. We can pinpoint the issues that our algorithm may be having with these tests and with the much bigger point cloud scenes given to us by Stratom, we can run as our final tests. These large clouds also come with the result position that their current algorithm is getting as an answer, meaning that we can just directly compare our answer to Stratom's already existing answer to test the accuracy.

These simple test cases consisted of just adding various amounts of noise, walls, curved surfaces, and moving the template around. The algorithm itself had a very hard time with the flat surfaces because of how flat the template itself was, it started to think it was perfectly matched even though it was just in the wall. After we resolved the issue of getting stuck in walls with a good initial guess, we could move onto our final test cases.

These final tests were just the tests given to us by Stratom because these are the real-world situations our algorithm will come against. Passing these tests given to us from Stratom will confirm that our algorithm has a possibility to work on their refueling arms because it will match the results of their already existing algorithm.

VIII. Project Ethical Considerations

Our software may not have many consequences for real ethical dilemmas since it is a template matching algorithm for automated machines, but the ethical considerations we should be considering is based on our professionalism and that we deliver a viable and presentable final product. The ACM/IEEE Principles that we learned about don't just focus on impact of social issues but focuses on the impact of design and development issues. I think the ACM Principles that are most related to our project are:

- 2.1: Strive to achieve high quality in both the processes and products of professional work.
- 2.2: Maintain high standards of professional competence, conduct, and ethical practice.
- 2.4 Accept and provide appropriate professional review.

The IEEE Principles that are most related to our project are:

- *3.* PRODUCT – Software engineers shall ensure that their products and related modifications meet the highest professional standards possible.
- 5. MANAGEMENT – Software engineering managers and leaders shall subscribe to and promote an ethical approach to the management of software development and maintenance.

These principles that are the most related to our project are also the principles we are most in danger of violating. They are the principles we need to refer to as we continue to develop our software but also can easily be broken by our project. The consequence of breaking these principles is a lower quality final product that may not even cover all our constraints and requirements. It will also result in a final product that doesn't conform to professional standards.

The two tests we have chosen to analyze on our project are the legality test and the reversibility test. When considering the legality of the choices we make when developing this product, we must decide if each aspect of our project is violating a current law or policy. In regard to our template matching algorithm, there is no law being broken within its development. All steps of development and the final product fit into the realm of legality and do not call anything into question. Our algorithm is simply designed to locate a given template object within a 3D space, a fully legal and ethical design. The purpose we are using this for is primarily to locate gas caps for automated refueling, a goal not in violation of any given policy.

Our second test for our design and product is the test of reversibility; defined as would this product still look good if I were on the side of those affected by it. When the effects and resulting reversibility of our product are

analyzed, we find that it does not inflict any negative repercussions upon those affected. This algorithm will simply act as an abstraction for the true purpose of the final product, in our case a robotic refueling arm. The existence of automated refueling will create a positive impact on all, automating a redundant day-to-day task. In all applications of our template matching algorithm, there are vast benefits. Locating objects by computer will allow for vast improvements in the lives of laborers, taking the risk away from dangerous tasks.

IX. Results

The goal of our project was to create a 3D template matching algorithm that could be deployed into Stratom's already existing code stack. We have achieved almost perfect alignment of the template within any point cloud. This generic algorithm can find a template anywhere within a scene. We have successful matches on multiple input cases, and our algorithm clearly finds where the template is in the larger scene. With every bag given to us to test with from Stratom, our algorithm finds the correct final location in each one.

Figure 6: Final Result of Algorithm #2

Furthermore, we have generated a test suite of transformations of our template to accurately test our algorithm in an ablation style study. The test suite comprises of many tests generated from transforming the template given to us by Stratom and adding our own points. We created different test cases to simulate different environments, and to test which environments our algorithms had the most trouble with.

This is to ensure our algorithm can work on a variety of inputs and is a method of benchmarking our algorithm. We have increasingly challenging scenes to find the template in, including with a wall, a curved wall, and with noise. Each of these are designed to challenge and determine where the algorithm breaks on these challenging scenes. Our algorithm performs well on each of these scenes and gives a transformation with minimal error on each of these scenes.

We also present this ablation test suite to Stratom, our client, for their use. In the event they want to update and change the algorithm, any new algorithm can be tested easily with this ablation test suite.

X. Future Work

When we complete our project and it meets all the functional and non-functional requirements, it can always be improved. That is already what our original project was in the first place. Stratom, the company that we are working with, already has developed a 3D template matching algorithm that is currently deployed on their robotic refueling units, making our project just an improved version. The accuracy should not be able to be improved much, since all the results must be good enough for the refueling unit to work properly. The main improvements could be made in the timing aspect of the algorithm. Point clouds could be very dense and hold lots of data, so the preprocessing of an algorithm may take a long time, and this is where the improvements in timing can be made. If Stratom wanted to

improve the algorithm even more, they would just have to remove more preprocessing and create an even faster algorithm.

If they did not want to just improve the algorithm itself, there is always room to develop the test cases. Developing even more unique tests cases that test even stranger edges cases can be very useful for developing future algorithms. Our algorithm may also get stuck when it cannot find an exact answer, and to improve upon this fact, they can make it so if the algorithm cannot find an exact answer then it will give an error message instead if outputting an incorrect solution. Overall, future work for a project like this will never be complete because at some point there will be improvements in the timing of the algorithm, whether it is from software improvements or hardware improvements.

XI. Lessons Learned

This project was extremely difficult, but due to the difficulty we learned a lot more than we expected. We learned about many things like different types of software, project development strategies, and research methods. Some of the most important lessons we learned:

- Scrum methods are extremely powerful in both learning and developing software projects. Breaking down a complex project into smaller pieces and completing them one by one makes a huge project much easier to track and complete. The sprint length of around 2 to 3 weeks seems to be enough time for progress, but not long enough that we may get stuck and cannot work on other aspects of the project.
- Robot Operating System 2, or ROS2, is an open-source framework to build robotic applications. This was the main software tool that we had to learn to create our project. Though extremely complicated initially, you start to understand more as you work with it and see how powerful it can really be. It is used across a variety of robotic applications, including autonomous vehicles, service robots, industrial automation, and research in robotics.
- Visualizing a point cloud can be a lot more complicated than we initially thought. We needed to use the Jetson they provided us to run our algorithm, but unless we were at the location of our Jetson, we couldn't see the visualization created. We tried to use remote desktop software, but none were compatible with the Jetson. We finally were told that we could transfer the ROS2 topic from the Jetson and visualize it through our local devices.
- Taking references and using open-source software is extremely helpful in both learning and trying to implement your own software. There is a lot of information online that others have already learned, so we should use their research to teach ourselves that same information. 3D template matching is also a common problem in robot systems, so there are already many implementations that have already been developed. These implementations could be used to see if they solve our problem or as references to try to create our own implementation.

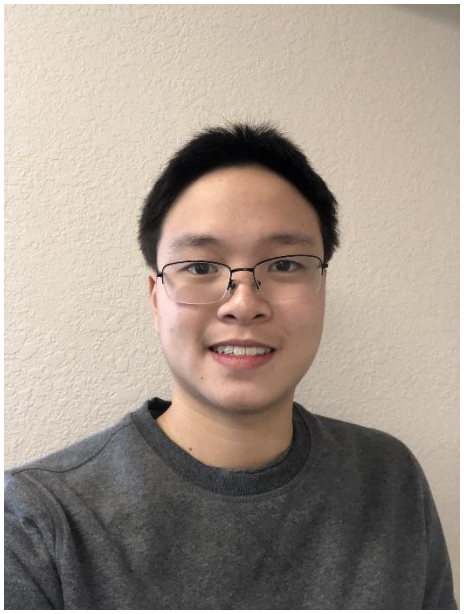
XII. Acknowledgments

We would like to acknowledge Kristin Farris at Stratom for her hard work in assuring us we were successful this semester. She was incredibly helpful to us throughout the term, answering all our questions and providing us with all the tools we needed to succeed. We would also like to acknowledge Donna Bodeau, our advisor, for doing an excellent job in guiding us this semester. She was incredibly helpful and accommodating for any help we needed, even letting us set up our hardware in her office when we needed to.

XIII. Team Profile

My name is Thuan Nguyen and I am a third-year undergraduate student here at the Colorado School of Mines. I was born and raised here in Colorado, and just chose to go to a school near me. I have decided to major in Computer Science, but have not specified a track I'm interested in.

My name is Adam Bukres and I am a fourth-year undergraduate student at Colorado School of Mines. I am originally from Oregon but moved here for school. My track is general Computer Science, but my focus and work are centered around computer networking



My name is Nathan Woo. I'm a combined M.S. + B.S. computer science senior at Colorado School of Mines, graduating in May 2025. My track is general computer science, but I have been focusing on robotics and autonomous systems.

Appendix A – Key Terms

Include descriptions of technical terms, abbreviations and acronyms

Term	Definition
ROS2	*An open-source framework designed for the development of robotic systems.*
point cloud	*A collection of data points in three-dimensional space, often representing an object, environment, or scene.*
Nvidia Jetson	*A very compact computer that consists of very powerful GPUs, CPUs, and specialized hardware to enable real-time processing and decision making.*