AI-Driven Autonomous Robot Demonstrating NVIDIA Jetson Edge-Computing Platforms

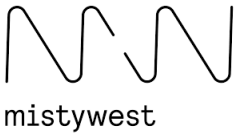
Finck,	Khorram,	Kuwabara,	McConkey,
Glyn	Austin	Masayoshi	Keenan
19759166	22105150	41242158	21337167

Project Sponsor

Dr. Andreas Putz - MistyWest

The University of British Columbia

Engineering Physics Project Laboratory



ENPH 479 - Capstone Project

Project Number: 1958

April 21, 2023

Executive Summary

Background & Project Motivation

The sponsor of this project, MistyWest, is a leading-edge, multi-disciplinary engineering consultancy company based in Vancouver. Many of their clients are innovative industry leaders, and thus it is important for MistyWest to stay up-to-date in the latest technological trends of various fields of engineering, including computing and robotics.

Project Summary

This project aims to demonstrate the potential of new edge-computing platforms in robotics applications by constructing a robot to perform a live demonstration made capable with the computing power of the NVIDIA Jetson TX2 platform. The demonstration will operate fully autonomously and will consist of SLAM-based navigation, visual recognition of empty bottles in a vertical orientation, mechanical retrieval of the bottles, and delivery of the bottles to a specified location marked by a QR code.

Key Objectives

1. **SLAM-Based Navigation**: using an Intel RealSense 3D camera and odometry data from an inertial measurement unit (IMU), the Jetson TX2 generates a map of an initially unknown environment, updates the map continuously, and tracks its own position as it traverses autonomously.
2. **Bottle Recognition**: the Jetson TX2 processes images via a 2D webcam through a pre-trained neural network designed to recognize the image of a vertically oriented bottle.
3. **Bottle Retrieval**: the robot navigates towards bottles it detects and picks them up with a front-mounted claw.
4. **Bottle Delivery**: the robot searches for and navigates towards a QR code which marks the bottle drop-off location, and drops the bottle at that location.

Conclusions

1. The Jetson TX2 allows computationally-expensive tasks, such as running deep neural networks, to be performed on-board, providing a significant advantage over conventional microcomputers
2. While our demonstration is still in the stages of functional but independent modules, over the next few months we will be fine-tuning these modules and integrating them into a single, coherent demonstration for the Project Fair in March

Contents

List of Figures

1 Introduction

1.1 Project Sponsor

Our sponsor, MistyWest, is a leading-edge, multi-disciplinary consultancy company based in Vancouver (6). Many of their clients are industry-leaders in innovation, therefore it is important for them to stay up-to-date on upcoming trends in various fields, including computing and robotics. Dr. Andreas Putz, a computer scientist and project manager at MistyWest, reached out to the Engineering Physics project lab to create a capstone project centered around exploring the potential of new edge-computing platforms in robotics applications. Specifically, Dr. Putz was interested in creating a low-cost autonomous robot to showcase the power of these platforms through a demonstration of artificial intelligence (AI) and computer vision. Dr. Putz was also looking to have more autonomous robots in the MistyWest office, which can be shown to clients and potentially perform useful tasks.

1.2 Background & Significance

This project is based around the concept of edge-computing and its role in robotics. At its simplest, edge-computing refers to the idea of bringing the collection and analysis of data closer to a location where it is used, rather than offloading this process to servers (7). In a robotics context, this means moving traditionally computationally demanding processes such as machine learning and computer vision, to a computer on-board the robot. Several manufacturers have moved to develop integrated platforms specifically meant for edge-computing, such as NVIDIA's Jetson family of embedded computers (8). All systems in this family are System-On-Modules (SOM), meaning they contain all of components of a regular computer on a single board. This allows for high-performance computing in a small, power-efficient footprint, making them well-suited to autonomous robots.

Several robots that take advantage of edge-computing platforms have already been created. Using the Jetson Nano, NVIDIA have created the JetBot (9), an open-source robot that serves a starting point for those interested in AI-driven robotics. Similarly, MistyWest's own Robo-Steve (1) uses a Jetson TX2 to run machine learning and computer vision, and served as a prototype for the our robot, BUDD-E.

Much of the significance of this project is in the value of exploring the the NVIDIA Jetson family of SOMs for our sponsor. This includes open-source documentation of our work with these platforms. Creating interactive demonstrations of the performance of this technology is also important for generating public interest in the field.

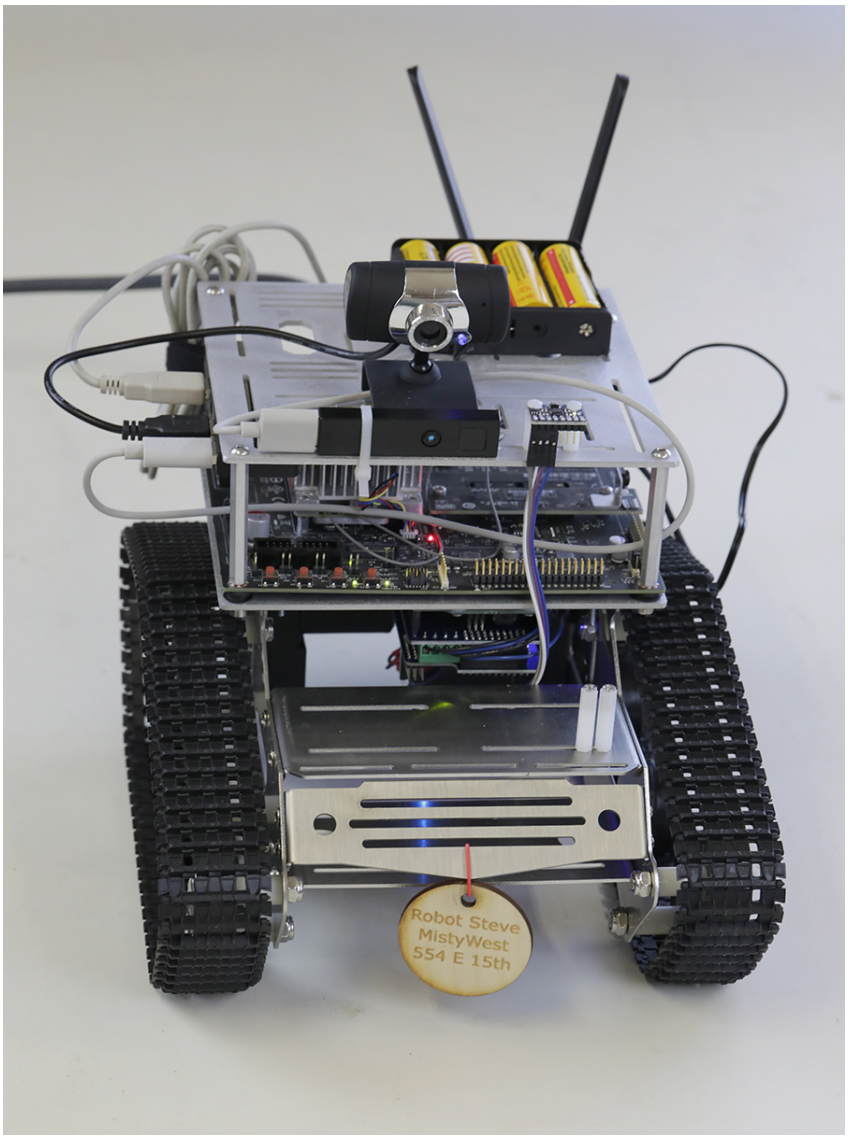


Figure 1: RoboSteve, a NVIDIA Jetson TX2-based robot which served as the prototype for our robot. Mechanically, our robot is the same except for the addition of a robotic claw. The USB camera/depth sensor pair shown in this figure was replaced with a Intel RealSense D435i.

1.3 Project Statement & Objectives

Through discussion with the sponsor, we distilled the project down to a single statement:

> *"Demonstrate the effectiveness of new NVIDIA edge-computing platforms by creating an autonomous robot that performs an interesting and interactive demonstration of AI technology"*

We further broke down the sponsor's priorities into a list of high level objectives:

1. Explore the power of edge-computing platforms in robotics applications
2. Create an exciting demonstration of this new technology for a wide audience
3. Perform machine learning and computer vision "at-the-edge" using the NVIDIA Jetson family of SOMs
4. Have more autonomous robots roaming around the MistyWest office

1.4 Scope & Limitations

We worked with our sponsor to decide on a demonstration for the robot to perform that best fulfilled these objectives. We chose to create a demonstration of a recycling robot (i.e. a "trashbot"), inspired by the Pixar movie, WALL-E. Our robot, BUDD-E, autonomously roams around an office, classifies and collects recyclable objects, and deposits these at a drop-off location. The scope of this demonstration includes the following (*note: this list is not exhaustive*):

1. Implementing machine learning and computer vision to perform on-board recognition and classification of recyclable objects
2. Developing a system for autonomously roaming around an office
3. Designing a method of physically manipulating recyclable objects (i.e. a robot claw)

We also made sure to clearly define the project limitations, as to retain a reasonable level of depth and complexity in the project scope. Using RoboSteve as a base to work with, initial chassis design and any major mechanical modifications were made unnecessary. We also made an effort to use as many pre-existing libraries as possible due to the small time window for the project.

The scope has been redefined in some areas since the kickoff phase of the project. Initially, we discussed having the robot interface with a cloud computing service to offload some of its processing there, but as that would diminish the significance of demonstrating on-board computing, that has

since removed from the scope. Due to the nature of the pre-existing libraries we managed to explore, the recyclable objects classified were limited to only include bottles. Finally, time constraints left us unable to fully investigate hardware-acceleration of machine learning using the Jetson TX2, which was a secondary point of interest for the sponsor.

1.5 Outline of Report

The rest of the report is split into three sections: discussion, conclusions, and recommendations. The discussion section will explain the fundamental concepts of the project and the design choices made throughout its completion. There we also present project results along with the testing used to verify them. The conclusion section will talk about the important takeaways of the project in regards to both original project objectives and sponsor priorities. Finally, the recommendations section will discuss recommended next steps for the project as well as new areas for potential exploration.

2 Discussion

2.1 Introduction to ROS

An in-depth discussion of the different subsystems of this project requires an introduction to the Robot Operating System (ROS) (10). ROS is a software framework for robot development that allows for encapsulation of low-level devices, message passing between devices and processes, and reuse of commonly implemented robot functionalities. ROS implements the various processes running on a robot using a graph architecture. Separate processes occupy individual nodes in the graph, and these nodes communicate through the use of topics (Figure 2). A node can publish messages to a topic, which other nodes can retrieve asynchronously by subscribing to the topic. Types of messages include sensor and camera data, odometry information, or high-level movement commands. The ROS Master provides naming and registration services for all of the nodes in the ROS graph.

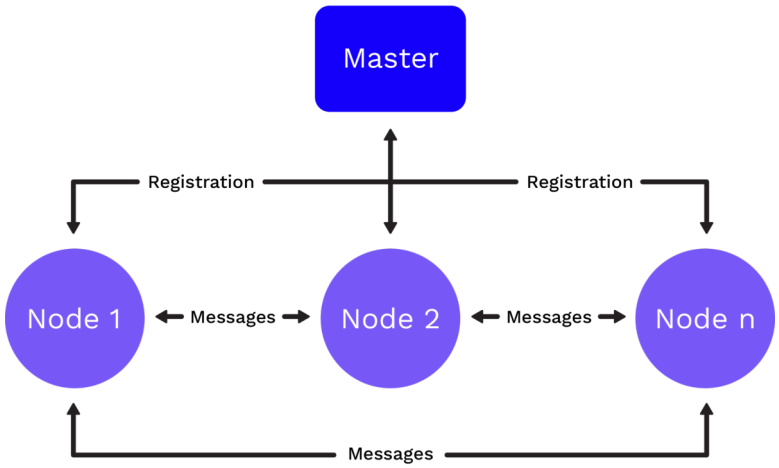


Figure 2: An example ROS graph. Each node in the ROS graph represents a separate process, and nodes are registered to the ROS master. Nodes can communicate by publishing messages to ROS topics, which other nodes can subscribe to (Source: RoboSteve Documentation (1))

2.2 High Level Overview

The concept of a ROS graph provides a convenient framework for showing the interactions between project components. Grouping together the nodes comprising the three high-level subsystems and produces a system level diagram of the software (Figure 3).

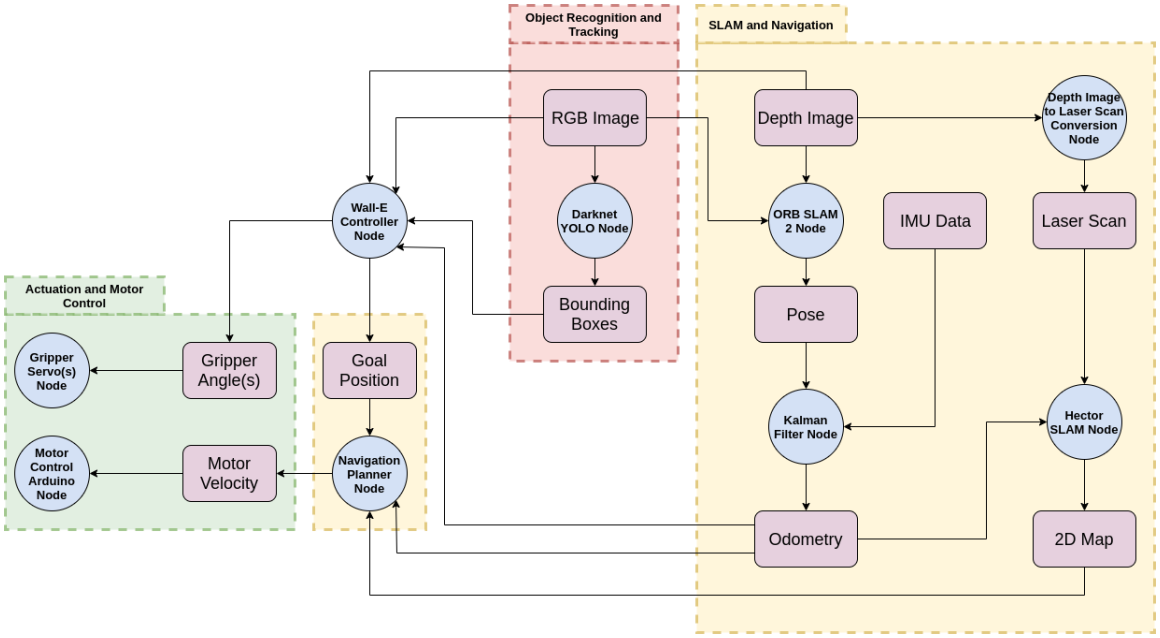


Figure 3: System level diagram of the robot as a ROS graph. The graph is divided into three subsystems: SLAM and navigation; object recognition and tracking; and actuation and motor control. Each subsystem contains ROS nodes and topics whose interactions are shown by edges in the graph.

As shown in the ROS graph, the project is comprised of three main subsystems: SLAM (simultaneous localization and mapping) and navigation; object recognition and tracking; and actuation and motor control. SLAM and navigation includes everything the robot uses to map, localize, and navigate throughout its surroundings. Object recognition and tracking relates to the algorithms used to detect and track bottle objects. Finally, actuation and motor control details the robot's differential track drive system and grabbing claw, as well as the ROS-encapsulated control of these motors. A more detailed explanation of each subsystem is given in Sections Hit Enter to reply 2.3, 2.4, and 2.5.

Each of these subsystems come together in the demonstration the robot performs. We used a finite state machine to encapsulate separate aspects of the demonstration into individual states (Figure 4). The robot then shifts between states when specific conditions are met.

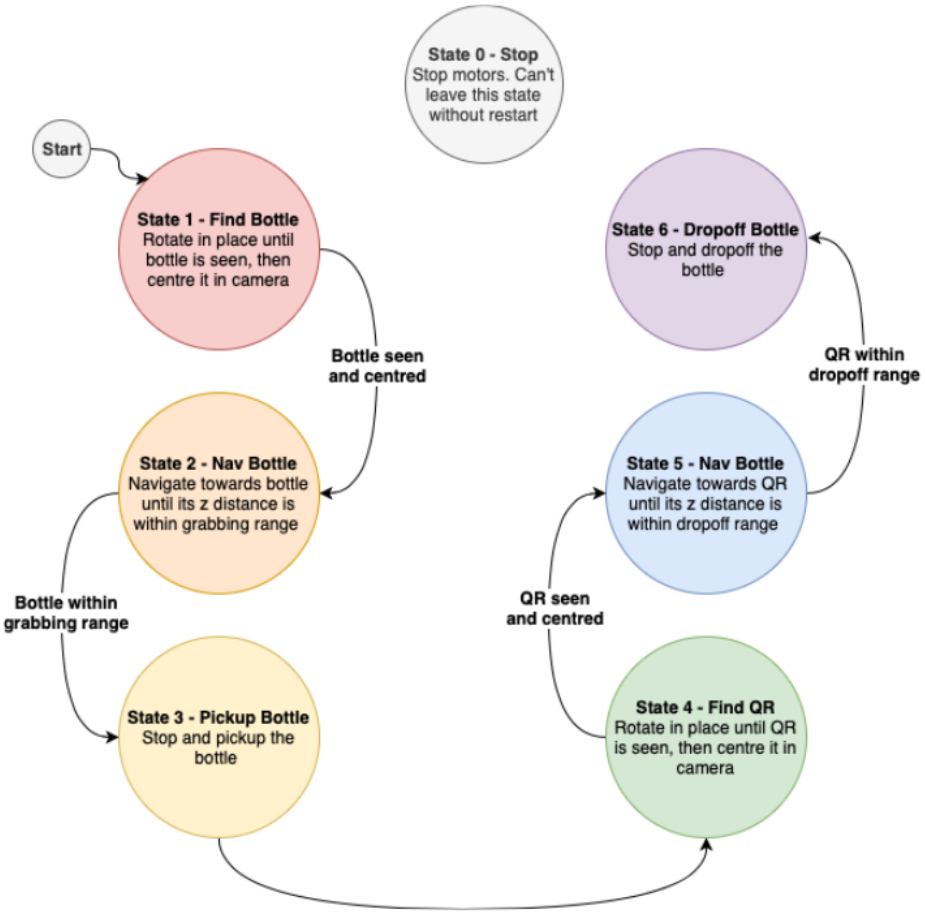


Figure 4: Finite state machine of the robot's bottle-retrieval demonstration. Individual states correspond to a specific set of actions that the robot performs.

.

2.3 SLAM & Navigation

2.3.1 Theory & Fundamentals

Simultaneous Localization and Mapping or SLAM is a process which is used in robotics to procedurally generate and update a map of the local environment while localizing the observer within it. Although SLAM can be implemented using a variety of different techniques, they all share a similar structure. The SLAM problem can be defined mathematically as follows:

Given a series of control and sensor observations, u_t and o_t, over discrete time steps, t, create an estimate for current position, x_t, and the map of the environment, m_t (11). The likelihood of our

estimated position and map are represented as

$$P(m_{t+1}, x_{t+1}|o_{1:t+1}, u_{1:t})$$

We can then use Bayes' rule to give us an equation for the likelihood of a particular current position given a map by the following (11)

$$P(x_t|o_{1:t}, u_{1:t}, m_t) = \sum_{m_{t-1}} P(o_t|x_t, m_t, u_{1:t}) \sum_{x_{t-1}} P(x_t|x_{t-1}) P(x_{t-1}|m_t, o_{1:t-1}, u_{1:t})/Z$$

We can also get an equation for likelihood of a particular map at by using Bayes' rule (11)

$$P(m_t|x_t, o_{1:t}, u_{1:t}) = \sum_{x_t} \sum_{m_t} P(m_t|x_t, m_{t-1}, o_t, u_{1:t}) P(m_{t-1}, x_t|o_{1:t-1}, m_{t-1}, u_{1:t})$$

From these mathematical definitions, various techniques exist to calculate approximations of the above equations and generate maps and locations of maximal probabilities given the history of observations.

An example implementation of SLAM uses key-points from images (RGB and/or Depth) as sensor observations, (o_t), and generates a 3D point-cloud of key-points, (m_t), as the map of the environment. An example SLAM algorithm is Oriented FAST and Rotated BRIEF or "ORB" for short (12). Such a SLAM algorithm must use a key-point detector to locate unique patches of pixels to be used as key-points. In the example of ORB, the FAST key-point detector shown in Figure 5 is used (12).

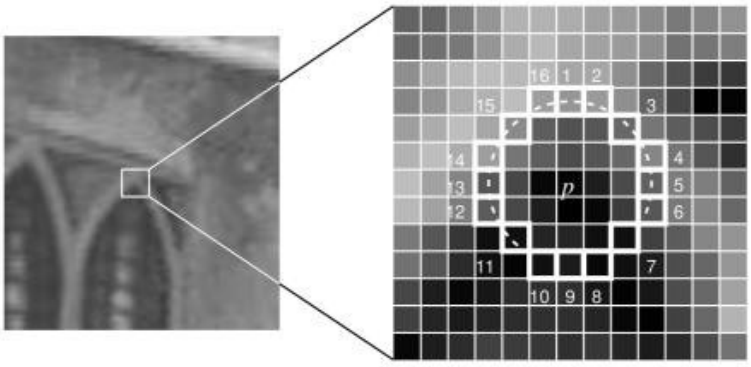


Figure 5: 12 point segment test corner detection in an image patch. The highlighted squares are the pixels used in the corner detection. The pixel at p is the center of a candidate corner. The arc is indicated by the dashed line passes through 12 contiguous pixels which are brighter than p by more than the threshold (2).

The next step of the algorithm is to compute a descriptor for each key-point. A descriptor encodes unique information about a patch of pixels into a series of numbers. This acts as a numerical

identification that is used to differentiate one feature from another, while also being used to identify matching key-points between consecutive images. ORB uses the BRIEF algorithm (13) to accomplish this.

Finally, pairs of matched key-points can be used to determine their 3D location in space to generate a sparse point cloud and determine the camera's pose change using linear algebra in an algorithm like the Eight-point algorithm (14).

Another example implementation of SLAM is Hector SLAM (15) which takes in subsequent laser scans and computes a 2D occupancy map. A laser scan is just a 1-dimensional depth vector that is computed via Time of Flight (TOF). Navigation and path planning algorithms used are outlined in the the navigation ROS Wiki (16) and specifically the paper (17).

2.3.2 Design Approach

To provide the robot with SLAM we centered our design around the particular sensors (RealSense d435i (18)) we had, going for the best approach given their associative advantages and limitations. The sensors we had available to us were a 1080p RGB camera (at 30 FPS), a 720p depth camera (at 90 FPS), and an IMU (200/400Hz for gyro, 63/250Hz for accelerometer). We decided that we would use a combination of all three sensors to facilitate our SLAM.

A major focus of ours was that we wanted to provide the robot with a good source of odometry given the sensors we had. Odometry is an essential aspect for SLAM as a bad odometry source can cause major instabilities with regard to localization and in-turn, accuracy of mapping. We ultimately reasoned that we should take advantage of the resolution available in both the RGB and depth cameras by using some sort of key-point tracking SLAM. Additionally, we decided that it would be beneficial to take advantage of the on-board IMU for providing orientation data. Using a Kalman filter (19) we could take orientation data from both the key-point tracking algorithm and the IMU to produce an accurate estimation of our robot's orientation with respect to its initial pose.

After determining our approach, we had to source examples of existing projects that were compatible with ROS. For key-point tracking we found a RGB-D (RGB-Depth) key-point tracking SLAM ROS node called ORB SLAM 2 (20). The algorithm would track key-points between subsequent frames in a point-cloud map, simultaneously localizing the robot within the generated map.

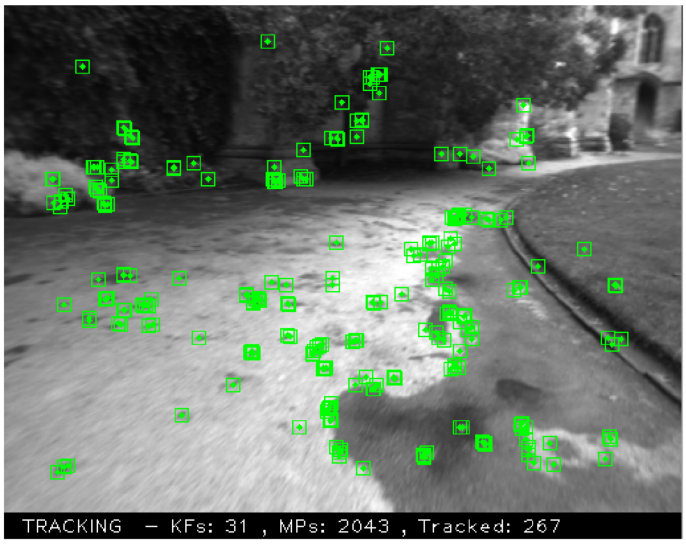


Figure 6: ORB SLAM 2 Tracking for one frame where key-points are shown as green squares in the image. This image was taken from the article in reference (3).

After completing this portion of the SLAM we needed to integrate it with the ROS navigation stack. As the ROS navigation stack requires a 2D occupancy map (Figure 7) we decided to implement a secondary SLAM algorithm to map the environment using a laser scan generated by our depth image. We chose to use Hector SLAM as it would allow us to generate a map using the odometry provided by our key-point tracking SLAM. As well, Hector SLAM has an available ROS distribution (15) allowing for us to easily adapt the node for our project.

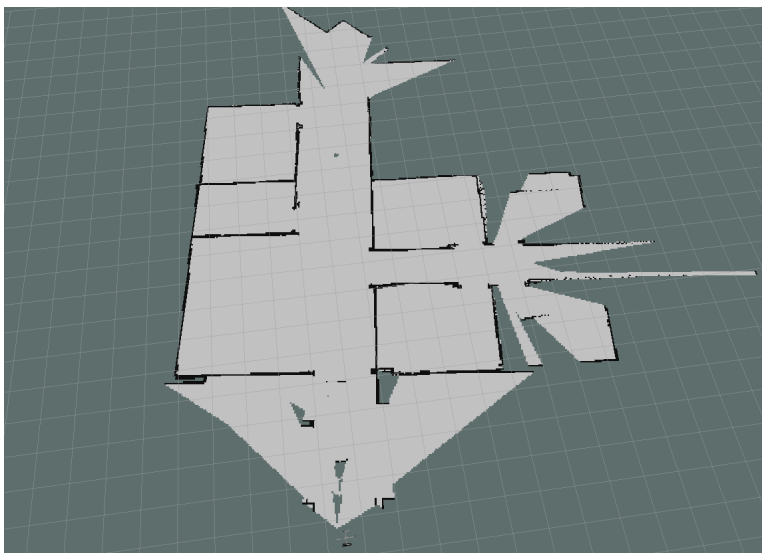


Figure 7: The figure depicts an example 2D occupancy map for ROS (4)

The next step was to fine tune the ROS navigation stack by adjusting various parameters in the standard configuration files. These files are outlined in the ROS navigation stack setup tutorial along with their associative descriptions (21).

2.3.3 Results

Our sponsors main objectives with regard to SLAM and Navigation were to use existing software available for ROS to give our robot the ability to localize, map and navigate an optimal environment. We defined an optimal environment to be an uncluttered office floor limited to an area of roughly 4 meters by 4 meters.

We were able integrate the ORB SLAM 2 ROS node as well as the Hector Mapping node with our ROS project to produce an accurate 2D occupancy map but were not able to localize our robot with enough stability to navigate throughout the map. Our qualitative/quantitative results are shown in the following section.

2.3.4 Testing & Validation

For testing our SLAM we wanted to initially start out by testing a very simple case and gradually test more complicated scenarios. Our first test was to map an area with a flat wall head of the camera, having no obstacles in the way (Figure 8).

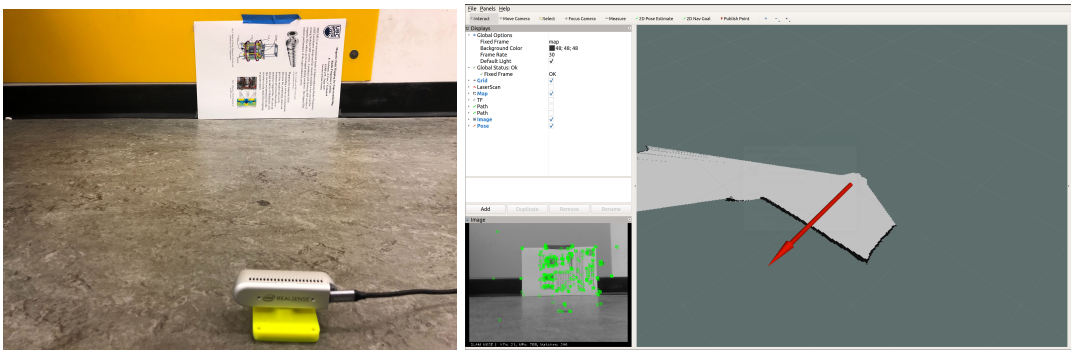


Figure 8: Image of scenario 1 for our SLAM testing (Left) and output from RVIZ (Right)

Our second test used the same area as the prior test but with the introduction of rectangular box as an obstacle (Figure 9). The success of this test shows that the system is able to detect non-uniform features (as opposed to uniform features; a blank wall) and map them accurately.

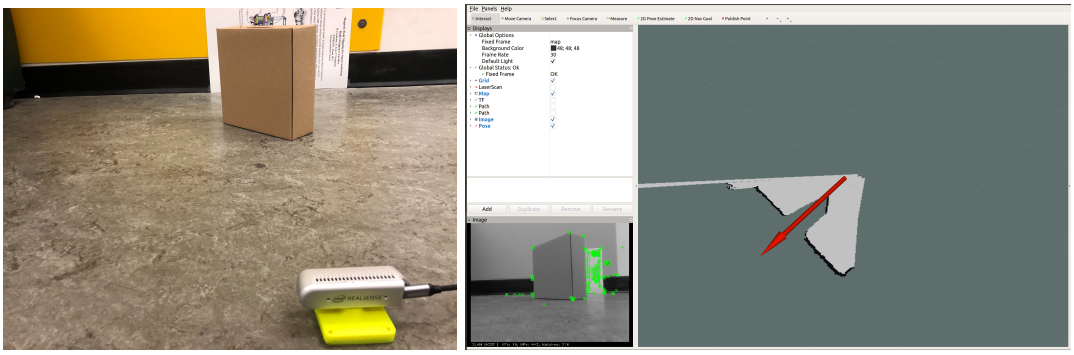


Figure 9: Image of scenario 2 for our SLAM testing (Left) and output from RVIZ (Right)

Our third test was to map a larger area with more complex features, seeing if our implementation was able to keep track of where it was (Figure 10). From this test we noticed that the odometry of the robot would start shaking rapidly after attempting to map a more complex area by rotating the camera 180 degrees. We reasoned that this could be due to conflicting odometry from multiple sources.

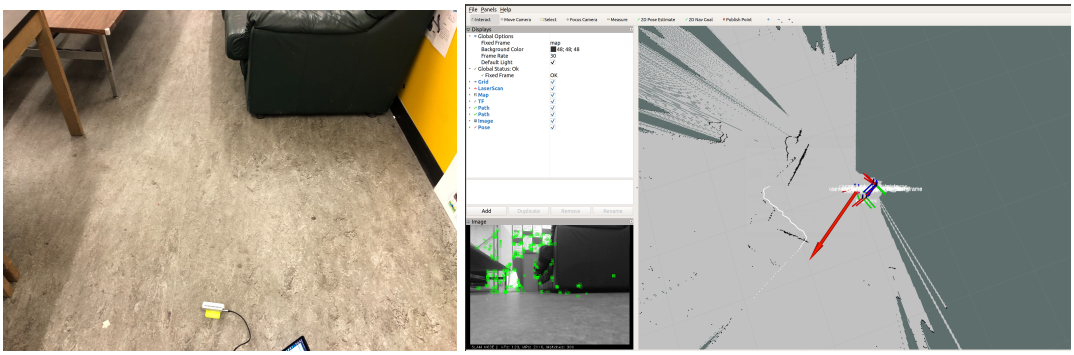


Figure 10: Image of scenario 3 for our SLAM testing (Left) and output from RVIZ (Right)

2.4 Object Detection and Tracking

2.4.1 Theory & Fundamentals

Object detection refers to the image processing task of identifying known objects in a given image. Traditionally, this is done by detecting and extracting key-points and descriptors in an image using algorithms as discussed in section 2.3 for SLAM. The descriptors can be compared known descriptors for key-points of the object being searched for to identify matches. More recently, machine learning has been adapted to accomplish the same thing, using convolutional neural networks (or CNNs). Convolutional neural networks use kernels and down-sampling to reduce the image to a more manageable number of inputs (as opposed to all the pixels) to finally feed into a neural net-

work for final classification. The use of kernels and convolution serves a similar purpose as key-point detection in traditional algorithms and the neural network can be thought of as being trained to match those key-points to known objects.

Object tracking refers to the image processing task of continuously locating a moving object across multiple chronological frames. This can crudely be achieved by performing object detection repeatedly, (extracting features and matching them to the known features or the features from the previous frame). Alternatively, one can attempt to match the patch containing the image in one frame to patches of the subsequent frame, accounting for occlusions or warping the in a variety of ways (such as a homography) to account for potential changes to orientation. Compared to repeated detection, this has several benefits including increased speed due to making use of existing information about the location and appearance of the object from the previous frame (**?**).

2.4.2 Design Approach

We opted to use the YOLO (you only look once) system, an open source pre-trained CNN for object detection as the system was already capable of detecting bottles and outputting their bounding box in pixel coordinates. Adopting an existing system eliminated the issues of implementing object detection algorithms ourselves and gathering training data, both of which would be time-consuming tasks. Our initial tests attempted to use the YOLO object detection system to track as discussed in the theory and fundamentals above, but the computation proved too lengthy to precisely control movement of the robot, especially at close proximity to the bottle. A solution was found by passing the bounding box provided by YOLO to a Minimum Output Sum of Squared Error or MOSSE tracker to interpolate between YOLO detections to provide smoother tracking which translated into smoother control.

2.4.3 Results

Using YOLOv3 to detect bottles proved to be very reliable, with an estimated recall rate in the range of 80-90% in the context of our robot. However, YOLOv3 ran at a framerate of 3-4 FPS and our initial attempts to track using PID control solely with the bounding boxes provided by YOLOv3 proved to be unstable due to the large delay in feedback causing the robot to overcorrect and lose sight of the bottle. The implementation of the MOSSE tracker increased the framerate to above 60 FPS, allowing for smooth close-range control and allowed the robot to successfully approach and pick up a bottle.

2.4.4 Testing & Validation

Testing done for YOLOv3 was mostly done through qualitative tests using various bottle types to see how accurate the classification was (Figure 11). We found that the classification could accurately classify almost any plastic bottle in which we tested with above 80% accuracy. We decided that from these qualitative test the algorithm performed well enough to be used by other software components.

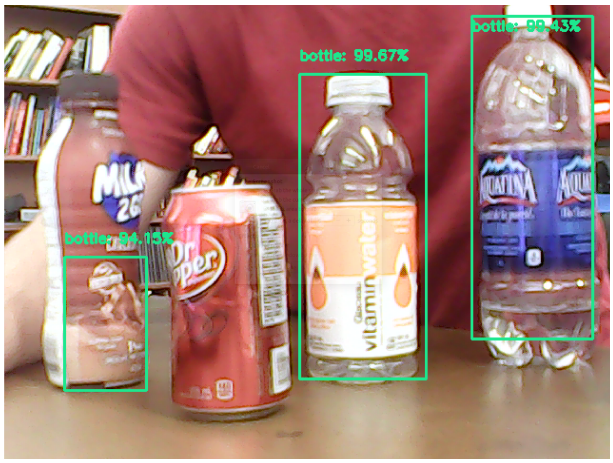


Figure 11: Successful bottle detection using YOLOv3

We also performed tests regarding our custom object tracking node to ensure that it would accurately track a bottle when moved rapidly side-to-side across it's view (Figure 12). We determined that this would be a good test as this was where the tracking algorithm was most likely to lose track of the bottle. The test shows that the object tracking is able to accurately keep up with movement of the bottle. Additionally, the tracking node was tested in conjunction with the PID control and with the correct PID settings, was sufficiently accurate to guide the robot to pick up a bottle.

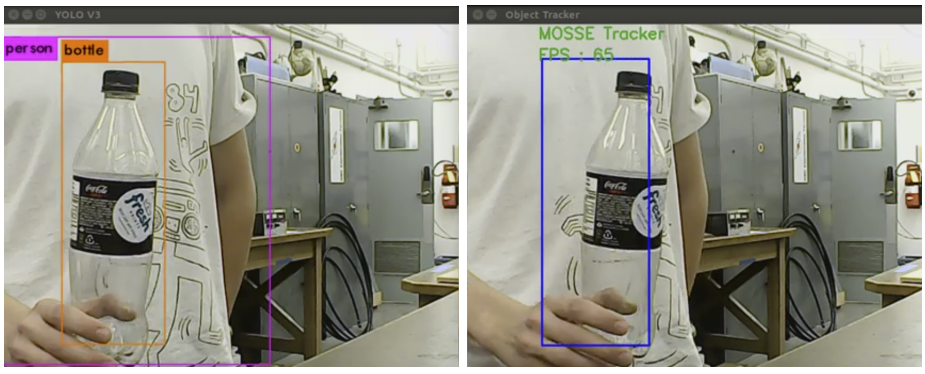


Figure 12: Comparison of tracking using only YOLOv3 (Left) and a combination of the MOSSE tracker interpolated between subsequent YOLOv3 frames (5)

2.5 Mechanical Actuation Drive System

2.5.1 Theory & Fundamentals

The requirements of mechanical actuation within the project scope were simply to design a front-mounted claw capable of picking up and holding a single empty bottle. Therefore, we needed to consider the weight of the bottle, weight and center of mass of the claw, the driving/stalling torque of the servo motors, and claw grip properties (i.e. material, geometry, etc.). These factors affected design choices such as material composition and hardware/component selection, as well as the design complexity. The chassis of the robot was actuated by a differential drive system consisting of two DC motors driving plastic tracks. These motors, along with the servo motors for actuating the claw, were controlled serially by an Arduino mounted on the chassis.

2.5.2 Design Approach

Our design approach for the claw was to allow for as much mechanical compensation for inconsistent or inaccurate software tracking while maintaining a reliable performance through a simple design. This meant designing the claw opening to be significantly larger than the diameter of a standard 500ml bottle to account for any errors in bottle localization. 0.25" Acetal (Delrin) was chosen as the material for the components of the claw due to its high material strength, reliable threading, and ease of cutting on the laser-cutter. The 2-bar design shown in Fig.13 allows a consistent gripping force to be evenly applied for a wide variance of bottle diameters.

2.5.3 Results

The robot achieved a success rate of over 80% for picking up bottles that he had to navigate to. Often when the robot failed it was due to inaccuracy of the tracking algorithm and the motor inconsistency, with the open claw not having enough space to compensate for said inaccuracy. There is opportunity for both improvement of the claw design, and optimization of the tracking algorithms and motor control to increase this success rate.

2.5.4 Testing & Validation

Despite prior torque calculations suggesting that the servo motors would be powerful enough, the claw initially struggled to lift a bottle once it had been gripped. This was explained as the result of friction. This left us with a few choices to consider: Reduce the weight, upgrade the servo motors, or verify the operational parameters of the current servo were met (i.e. power requirement

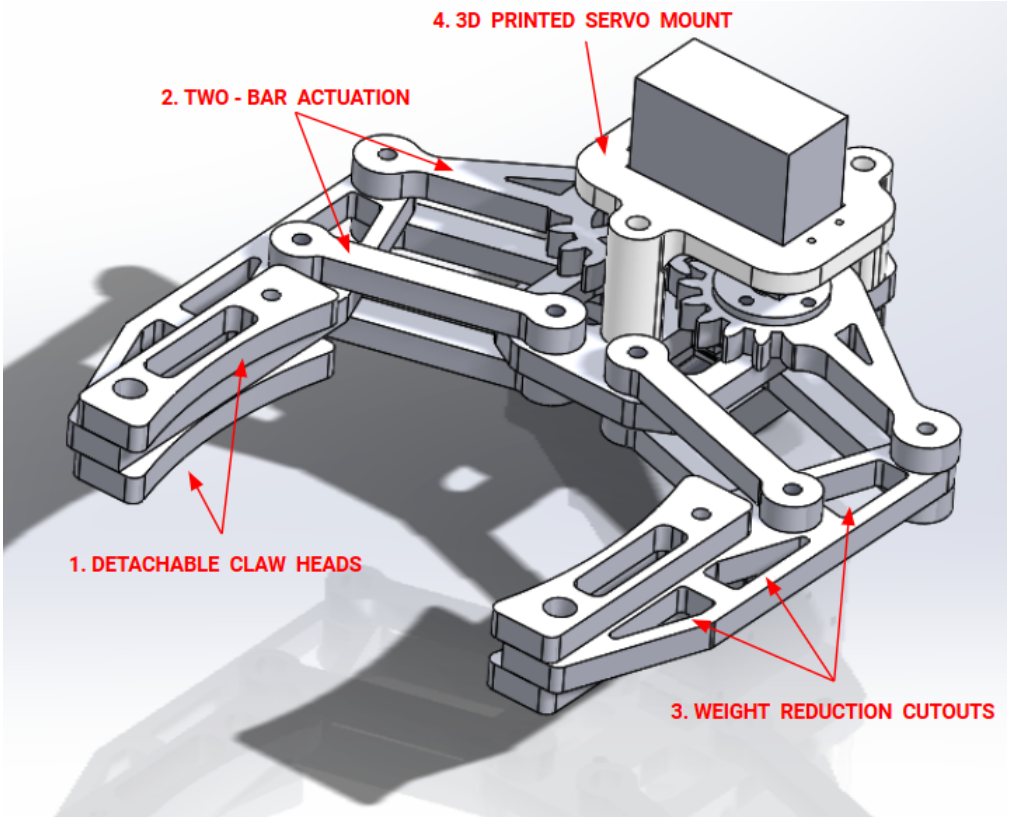


Figure 13: 3D CAD model of the claw design. (1) The detachable claw heads shown allow us to test the performance of different shaped heads. (2) The two-bar design allows for the claw heads to be parallel regardless of the distance between them, applying a consistent clamping force for any bottle diameter. (3) Weight reduction cutouts were added, as initially the claw was too heavy. (4) All parts were laser-cut from 0.25" Delrin with the exception of the 3D printed servo mount labelled on the image.

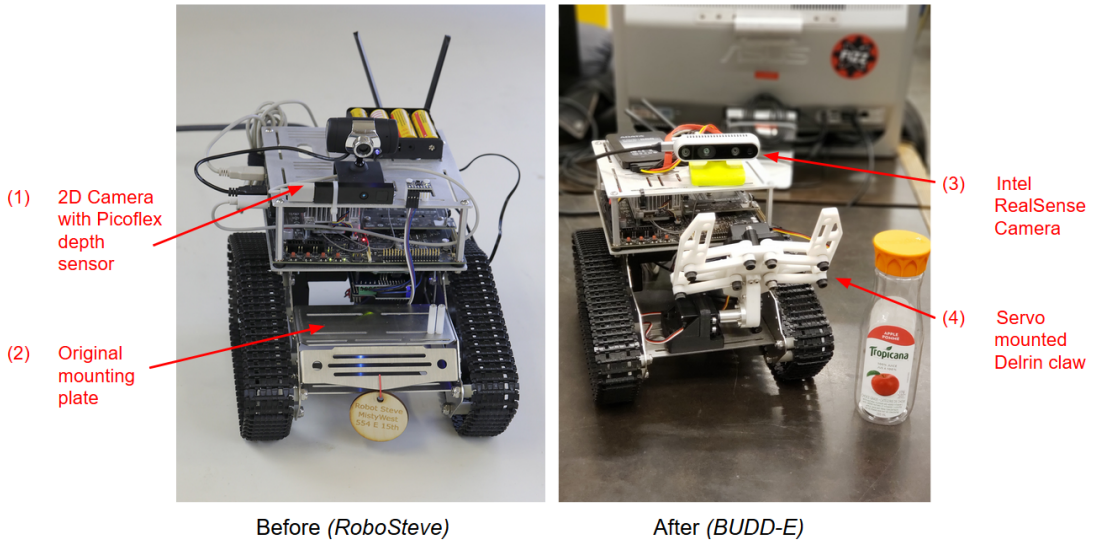


Figure 14: A before and after comparison of RoboSteve/BUDD-E. Shown in the image is: (1) the original setup with a 2D web camera and Picoflex depth sensor, (2) the original mounting plate was removed to mount the claw, (3) The Intel RealSense camera which is used for bottle recognition and SLAM navigation , (4) Servo-mounted Delrin claw for picking up bottles

validation, proper motor control, etc.). After validating the power requirements were met for the servo and checking that it was fully operational, we reduced the mass of the claw by removing the claw heads shown in Fig.13. The claw was then able to lift the bottle with ease.

Another issue with stability is present in the joint between the claw and the base servo (not shown in Fig.13). The joint is not very strong and may result in unreliable performance in the future. As this was not an immediate concern, it remains unresolved. We will revisit the claw design to resolve this issue in the coming months.

3 Conclusions

3.1 Power of Edge-Computing in Robotics

The power of the Jetson TX2 allows us to run computationally expensive software (i.e. SLAM algorithms, neural networks) in real-time, eliminating the need for offloading computations to another PC via a server. This demonstrates the potential for more robust and powerful small-scale robotics technology using new edge-computing platforms, such as the Jetson TX2. The potential of this technology reaches beyond that of our project scope, and even more computationally heavy

tasks may perhaps be performed on the Jetson TX2.

3.2 Performance of Final Demonstration

Currently, our demonstration consists of a few major modules that have yet to be integrated together. The robot is able to classify bottles from other objects with an accuracy between 80-90%. This accuracy diminishes slightly as the bottles get further away. The SLAM navigation generates a robust map, however conflicting odometry from the 2D mapping and keypoint mapping algorithms results in a erratically shifting internal odometry, and the robot has a difficult time determining it's own location. As for bottle retrieval, a simple PID-like approach we implemented allowed the robot to consistently pick bottles, however this approach will be overwritten once the SLAM algorithm is tuned. QR code recognition proved to be a challenge when the QR code was more than a foot away from the camera, and thus the drop off location will be determined in the mapping phase of SLAM. This has not yet been implemented.

Despite the lack of integration between the SLAM navigation, retrieval/drop-off, and bottle recognition modules, the results of the modules individually still demonstrate how powerful the Jetson TX2 can be, as all the SLAM mapping and object recognition computing is done on-board. Over the next few months we will fine-tune and integrate these modules together to develop a more coherent demonstration for the project fair in March.

4 Recommendations

The next stage of work to come along in this project is the improvement of individual modules, and the integration of all the separate modules into a systematic demonstration, as stated in section 3. We will be working on achieving this until the ENPH 479 project fair, and then we can provide an updated status on the complexity of the demo, and re-evaluate the next steps. Currently, the next steps include the following:

1. SLAM navigation needs to be improved by fine-tuning the algorithms that track internal odometry in order to allow the robot to better track itself in the map that it generates
2. SLAM implementation will then need to be applied to navigating towards a bottle and to the drop-off location
3. Remove the QR code detection and PID-like tracking approach, as they will be obsolete once we implement SLAM to accomplish the aforementioned tasks

4. Integrate the SLAM navigation module with the bottle recognition module, solidifying a single, coherent demonstration

Once we finish with this particular demonstration, future work on the robot will likely focus on improvement of the navigation reliability. This is an area where there is almost always room for optimization, and depending on how far we manage to develop the SLAM navigation, will likely be able to be improved. Future work may also include the development of entirely new demonstrations able to focus on other significant aspects of robotics, such as machine learning and hardware acceleration.

5 Deliverables

1. A fully functional autonomous robot powered by the NVIDIA Jetson TX2, capable of performing the aforementioned "trashbot" demonstration in the offices of MistyWest
2. Open-source documentation of our work (CAD files, notes, repositories, etc.)
3. The final report, to convey the information we gathered and communicate what was learned throughout the project

References

[1] A. Putz, "Robotic Steve 3: Body & Soul." [Online]. Available: https://www.mistywest.com/posts/robotic-steve-3

[2] D. Tyagi, "Introduction to fast (features from accelerated segment test)," May 2019. [Online]. Available: https://medium.com/@deepanshut041/introduction-to-fast-features-from-accelerated-segment-test-4ed33dde6d65

[3] R. M. M. Mur-Artal, J. D. Montiel, and Tardos, "Orb-slam: A versatile and accurate monocular slam system," *IEEE Transactions on Robotics*, vol. 31, no. 5, p. 1147–1163, 2015. [Online]. Available: http://webdiis.unizar.es/~raulmur/MurMontielTardosTRO15.pdf

[4] David, Brian, Austin, Aykut, Kumar, Paolo, A. Castro, and Yiorgos, "Ros integration - quadrotor 2d mapping navigation." [Online]. Available: https://www.wilselby.com/research/ros-integration/2d-mapping-navigation/

[5] "Yolo vs tracker.mp4." [Online]. Available: https://drive.google.com/open?id=1fIdDyuM_Fs39IiV1N7CT4_GmE8n6AdBS

[6] MistyWest, "MistyWest Website." [Online]. Available: www.mistywest.com

[7] Cloudwards, "What is Edge Computing: The Network Edge Explained." [Online]. Available: https://www.cloudwards.net/what-is-edge-computing

[8] NVIDIA, "Embedded Systems Developer Kits & Modules from NVIDIA Jetson." [Online]. Available: https://www.nvidia.com/en-us/autonomous-machines/embedded-systems

[9] ——, "An Education AI Robot Based on NVIDIA Jetson Nano." [Online]. Available: https://github.com/NVIDIA-AI-IOT/jetbot

[10] ROS.org, "ROS: Powering the World's Robots." [Online]. Available: https://www.ros.org

[11] "Simultaneous localization and mapping," Jan 2020. [Online]. Available: https://en.wikipedia.org/wiki/Simultaneous_localization_and_mapping

[12] "Orb (oriented fast and rotated brief)." [Online]. Available: https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_feature2d/py_orb/py_orb.html

[13] D. Tyagi, "Introduction to brief(binary robust independent elementary features)," May 2019. [Online]. Available: https://medium.com/@deepanshut041/introduction-to-brief-binary-robust-independent-elementary-features-436f4a31a0e6

[14] "Eight point algorithm." [Online]. Available: https://en.wikipedia.org/wiki/Eight-point_algorithm

[15] “Hector slam.” [Online]. Available: http://wiki.ros.org/hector_slam

[16] “Ros nav.” [Online]. Available: http://wiki.ros.org/navigation

[17] E. Marder-Eppstein, E. Berger, T. Foote, B. Gerkey, and K. Konolige, “The office marathon: Robust navigation in an indoor office environment,” 2010.

[18] “Depth camera d435i.” [Online]. Available: https://www.intelrealsense.com/depth-camera-d435i/

[19] “Kalman filter.” [Online]. Available: https://en.wikipedia.org/wiki/Kalman_filter

[20] “Appliedai initiative - orb slam 2 ros”.” [Online]. Available: https://github.com/appliedAI-Initiative/orb_slam_2_ros

[21] “Ros nav setup.” [Online]. Available: http://wiki.ros.org/navigation/Tutorials/RobotSetup