SW for EZ-Rassor Arm

Group 18

Members: Austin Dunlop, Robert Forristall, Luca Gigliobianco, Christopher Jackson, Cooper Urich

Sponsor: Mike Conroy

Executive Summary 1
Description 1
Objectives 1
Approach 1
Narrative Description 2
Goals/Objectives 3
Statement of Motivation 3
Function 7
Legal, Ethical, and Privacy Issues 8
Legal 8
Ethical 8
Privacy 8
Requirements 9
EZ-RASSOR Requirements 9
Programming Requirements 10
Hardware Requirements 10
Operating System 11
Specifications 11
EZ-RASSOR Robotic Arm 11
Software Specifications 12
Budgeting 12
List of Ideas 12
Broader Impacts 15
Software 18
Python 18
Ubuntu 18.04 18
Blender 18
Gazebo 19
PySerial (Version 3.5) 19
OpenCV (Version 4.5.1) 19
MatPlotLib (Version 3.3.4) 20
Intel RealSense SDK 2.0 20

Open3D (Version 0.12.0) 20
LasPy (Version 1.7.0) 20
NumPy (Version 1.20.1) 21
PyTorchCV (Version 1.9) 22
ImageAI (Version 2.0.3) 22
Conda Python Environment (Version 4.6.1) 23
Linux Operating System 23
Virtual Machine 23
Dual Boot 23
Arduino Software IDE 24
Sketches 24
Sketchbook 26
Uploading 26
Libraries 26
Serial Monitor 27

ROS Simulation Research 34

Blender Research 34

Gazebo Research 32

Arm Design research 33

State Machines 34
Types 34
Representations 36
Classification of Finite-State Machines 37
Moore Machines vs Mealy Machines 38

ROS/Other Robot Middleware 39
ROS 39
ROS 2 40
Player 41
YARP 41
Orocos 42

Edge Detection Methods 43
Canny Edge Detection 43
Gaussian Blur 43
Intensity Gradients 44
Non-maximum Suppression 45

Double Threshold & Hysteresis 46
Completion 46
Roberts Cross Edge Detection 47
Convolution Kernels 47
Gradient Magnitude 47
Image Application 48
Laplacian Edge Detection 48
Laplaician Method Pros and Cons 49
Sobel Edge Detection 49
Sobel Method Pros and Cons 50
Canny, Roberts Cross, and Laplacian Method Comparisons 50

Object Detection 51
Video Object Detection 52
Gabor Filtering 52
Motion Processing 53
Contour Extraction 53
Descriptors 54
Box Bounding 54
Location of Object 55
Optical Flow Method 55
Pros and Cons of the Optical Flow Method 59
Background Subtraction 60
Pros and Cons of Background Subtraction 65

Point Cloud Visualization 66
Triangle Mesh Method 67
Non-Uniform rational B-spline Method 67
Which Method We Chose and Why 68
Point Cloud Visual Implementation 68

Useful ROS Tools for Vision and Motion Planning 69
Rviz 69
MoveIt 69
ROS Industrial 70

Neural Networks 70
Simple Network Explanation 72
Convolutional Neural Networks 75
Pros and Cons of Neural Networks 79
Project Use Case 80

Arm Path Planning using Stereo Vision 81

Arm Path Planning using Point Cloud 86
- Important Concepts 86
- Probabilistic Roadmaps for Robot Path Planning 87
- Dynamic Roadmaps for Robot Path Planning 89

Current EZ-RASSOR 97

Design Summary 99
- ROS 102
- Simulation Model Phase 1 102
- Simulation Model Phase 2 112
- Simulation Camera Integration 118
- Robot Vision Summary 122
- Communication 124
- Autonomous Controller 127

Running Tests 130
- Initial Challenges and Difficulties 130
- Simulation 130
- Physical Robot Implementation 131

Hardware 131
- RealSense Camera 131
- Arduino Mega 2560 REV3 132
- Nvidia Jetson Nano Developer Kit 133
- Integrating the Jetson Nano Developer Kit with the Arduino Mega 2560 135

Implementation 136

Facilities and Equipment 142

Project Milestones 143

Challenges 145

What We Would Do Differently 149

What The Group Has Learned 153

References 158

Executive Summary

Description

The goal of this project is to implement a robotic arm for the EZ-Rassor suite and possibly for the real-world Mini-Rassor rover. The EZ-Rassor suite is an open source software package for educational purposes. The suite includes a simulation that can create a simulated model of the Mini-Rassor with manual or autonomous controls as well as a simulated environment that includes places like earth and the moon. This rover can mine regolith which is used to create pavers that can then be placed by the arm this team will be implementing to create landing pads and roads for other rovers/spacecraft to use since regolith getting airborne and into machinery is a big problem. First the arm will be implemented with manual controls before adding autonomous controls by utilizing the camera mounted to the arm and a visual neural network. While the arm will be designed firstly for the simulated environment, if time permits the arm will be integrated into a physical version of the Mini-Rassor for real-world application.

Objectives

The project's current objectives are as follows; First the robotic arm will pick up a paver after it has been created using a mix of regolith and 3D printable plastic by a 3D printer. Secondly, the paver will be placed onto the center of the rover along with the arm that will sit on top of it during transport so that the rover keeps a solid center of mass and doesn't tip. Third, the rover will pull up to the location that the paver is to be placed and the arm will begin to move the paver to the ready position. Finally, the paver is placed so that it links properly with the other already placed pavers before returning the arm to its resting position on the rover for transport to get another new paver.

Approach

There are two approaches to this project with them being a simulated approach and real-world approach. First the arm model and controls will be integrated with a simulation to simulate the autonomous control of the arm as it places and picks up pavers. These autonomous controls will be powered by a visual network and stereo depth camera. Once the simulation approach is fine-tuned; we will begin development of the real-world approach which will involve the same steps as the simulation but include the integration of the Arduino and Nano boards on the rover. Both methods will include unit testing to ensure that each segment of the implementation is functioning as intended and fix any segments that are having issues.

Narrative Description

This project was pitched as one of two projects being sponsored by Mike Conroy from the Florida Space Institute, both of which are continuations of the EZ-RASSOR projects from previous senior design teams whose initial goal was to create a smaller-scale version of the RASSOR project being worked on at NASA, designed with open-source software that could then be used as an educational tool for schools around the world. Project goals for that team eventually expanded to instead create a rover that could be used at the Kennedy Space Center for demoing and prototyping features of the RASSOR rover being worked on at NASA. Beyond just having purposes for demoing and prototyping, it was intended that in the future the rover would be sent on lunar and mars missions to mine regolith (dirt on other planets and moons).

The first team was tasked with building a simulated environment and rover; building an autonomous loop that allows the rover to act independently without human intervention and develop manual and autonomous controls for mining functionality. This team was also able to develop a mobile app that acted as a controller for moving the rover and operating the mining shovel. The team that followed was tasked with implementing GPS-Denied autonomous navigation by using a more sophisticated path navigation software so the rover could navigate to planned destinations without human intervention. The reason GPS-Denied software was important is because there would not be GPS satellites orbiting other planets, so using a method that does not rely on GPS would be necessary.

The project we were given is to develop software for an arm extension that is intended to be used to place roads and landing pads on future lunar and mars missions. The arm is already fully designed by Jakob Bruckmoser, which is entirely built with 3d printed parts. The roads and landing pads are intended to be made while the rover is active by taking the regolith that the rover gets from digging with the mining tool on the front of the rover. The regolith would be mixed with a plastic material so that it could be 3D printed, allowing for more pavers to be printed as needed at any time. Our team, with the work done by the previous teams, will design software for the boards to control the arms movement as well as work with the arms camera to implement computer vision to allow for autonomous capabilities.

Goals/Objectives

First and foremost, our goal is to get the arm moving and grabbing/placing objects in a simulated environment. Once the simulation is working with the arm and we have done thorough testing, our efforts will be shifted to making sure our implementation works with the physical EZ-RASSOR rover. A key issue that we will have to be aware of, especially when shifting to the physical development and testing, is to maintain arm movements that stay towards the center of the arm rather than doing wide swings. This could affect the rover's stability while moving, especially in lower gravity.

As this is a project that stems from several teams before us and is intended to continue after us, as well as it being a project that is intended to have education use, it is important that we maintain clear and thorough documentation throughout development. If all goes to plan, the final goal would be for our implementation to be used within NASA's RASSOR design and potentially be sent on missions. A list of what we hope to accomplish is as follows:

- build integrated software for the Nvidia Nano and Arduino boards
- develop computer vision software for the arms camera
- implement artificial intelligence algorithm for autonomous use of the arm
- have a fully functioning arm that works in the ROS gazebo simulation environment that the previous team developed
- add to the already existing github repository to maintain the open-source aspect of the project
- create thorough documentation both for the github and for any scripts, environments or API's created
- test software on a physical rover achieving movement with the arm and test its ability to grab and place objects without causing the rover to tip

Statement of Motivation

Out of all of the projects that were presented to the class during the first few weeks, only a couple of them really excited me and made me feel motivated to work on. It just so happened that the first project presented was the one that stuck out the most, which really set my expectations for the rest of the presentations high. What really caught my eye about

this project was that it was presented and advised by the National Aeronautics and Space Administration (NASA). Personally, I grew up on the space coast my entire life. My whole family has worked out in Cape Canaveral, Florida and ever since I was in elementary school I had wanted to work for the space industry. When I saw the project pitch that Mr. Mike Conroy made, I saw it as an opportunity to get my foot in the door in an important field that is taking off now more than ever in my life.

The very nature of this project also made me motivated as well. Almost relying heavily on computer vision and robotics to make this a successful project, it really made me excited to get started on this project. Throughout my time at the University of Central Florida (UCF), I have wanted to become a more well-rounded software engineer/ developer. I have developed my algorithmic side, my web development side, and now I am excited that I can develop my robotic vision and machine learning side. It is also a project to get me more familiar with coding languages that I wouldn't learn due to the curriculum here at UCF. Relying heavily on python and C++, this project will introduce me to coding languages that are used heavily in the professional world of computer science and software engineering. Upon completion of this project, not only will I have contributed to a good project and helped further the advancement of space travel, but I will have been involved with vital experience to take with me in my future professional endeavors.

- Cooper Urich

Computers and how they work have been a passion of mine since as long as I can remember. Growing up, my dad was always making new builds or upgrading components in his pc, and later around the time I reached high school, I began to learn from him and started to build my own pc's, creating the best system I could afford at the time. It was around this time I first became interested in programming. Me and my dad also spent a lot of time playing games together, so it was my dream to one day create my own game. Through the help of a book I bought at a local bookstore, I started my path of programming by learning how to make simple C++ games in the terminal.

When I decided to major in computer science, the main field I always envisioned myself pursuing was one in artificial intelligence or robotics. Ever since I saw the amazing work coming out from the engineers at Boston Dynamics, my goal was to be able to build robots that were at that level of complexity. Since I unfortunately started my degree with no prior background in robotics and artificial intelligence, I have had to make up for that lack of knowledge by consuming as much information I could during my time in school. I have

done so by taking classes such as intro to AI and robot vision; joining clubs like the UCF AI club and working on my own projects with raspberry pi's.

What made me want to be a part of this project so much is not only because it fit my interests, but it would also give me a great opportunity to work on a robotics project that is much larger scale than I could ever work on myself, as well as it being a project that is very closely tied to NASA, a company that I hope to have the opportunity to work for one day. I also hope to gain a lot more experience helping develop the visual network for our system. I have been able to learn a lot from classes like robot vision, and I have begun to love the field of studying how computers can view the real world, so this will give me the opportunity to really test my understanding of computer vision so far, and hopefully help me learn even more. Overall, I think this project will be one that will provide a great challenge and will give me experience that will prove to be invaluable in the future.

- Christopher Jackson

This project is like a dream come true for me; ever since I was very young, I was enthralled by space and all the mystery that came with it. Every time I saw a rocket blast off I wished that one day I'd be on one of those rockets; to see all that is beyond our small piece of the universe. Sadly that dream never fully took off due to my poorer physical health as well as my overwhelming anxiety but that didn't stop me from still looking up and wondering what was beyond our world and hoping that one day I could help advance our understanding of it. Call me surprised or in utter shock from excitement when I saw the very first project presentation was from the Florida Space Institute with an opportunity to work on a robotic arm that could greatly help push the boundaries of what we can do in space. My dream was in my grasp, and it came with the caveat of being bundled with another great passion of mine.

Artificial intelligence is my chosen focus in the field of computer science. I enjoy the idea of building networks that can complete very challenging tasks as well as learn from their surroundings and become more capable. So when this project was presented I was already hooked from the space angle but, when the arm was also mentioned and needed an artificial intelligence network to operate it I was sold. Getting to finally work directly with artificial intelligence in robotics was an amazing opportunity to expand my knowledge, skill, and experience. Not to mention, this would be work with FSI and NASA who are beyond knowledgeable in this field and whose experience/expertise would be invaluable to my progress in this focus.

Beyond all I have said there is one other aspect that truly motivates me to want to see this project succeed. While space is a lifelong passion and AI is my chosen field of focus and enjoyment; the thought that I can help create something that can change the lives of so many for the better is probably the biggest motivation I have. To add my work to a project that can lead to a revolutionary new way of exploring the cosmos by creating an arm that can be used to build roads and landing pads on other planets/moons is a feeling that is beyond words. Not to mention, I want to give future students an arm that they can use to help bring their own dreams of creating something great to life. All in all, I am beyond excited and motivated to see this project succeed.

- Robert Forristall

I am personally interested in and motivated for this project as it is directly related to advancements in space exploration and has the potential for real impact and benefit to the field. Growing up, I visited the Kennedy Space Center countless times and even attended space camp. I remember watching the many launches of the NASA shuttle program in awe. To this day I still look up at the stars many a night in wonder of what could be out there waiting for us in the great beyond.

Programming this arm to build a landing pad is a means to an end, the colonization of the moon, which itself is a means to an end, a jumping off point for further exploration of the cosmos. To be a part of that effort is a great opportunity. To feel that one's work is contributing to a meaningful goal is essential, and there is no obstacle to finding the meaning in this work. Even inherently without greater meaning, it is an interesting project.

The skills and work required for this project is directly in line with my field of study and interest. Automation through machine learning, which is what the arm requires, is exactly what I would like to be doing in the future. Will my future career involve programming space robots? I do not know, but I'm sure it will involve machine learning in some form and the skills I will gain from working on this project will be invaluable.

- Austin Dunlop

The motivation I get from this project is knowing that the open source software I help develop today can be used by EZ-RASSOR units on the moon in the future. I personally have a strong sense of admiration for open source projects like this one that aim to help a group of users by utilizing the community and teams. I believe that learning resources

should be as accessible as possible to motivate learning and to set a foundation for others to use.

I am also very interested in robotic systems and enjoy watching software get translated into mechanical movements. Since I was young, I always found robotic systems such as the ones found in mass production assembly lines to be fascinating. The planning, testing, and technical work that goes into these systems is extremely impressive and shows how far technology has come.

Finally, this project will be a great opportunity for learning novel technologies. I am excited to learn how to program Arduino boards using sketches, how to process a visual network using a development computer, how to implement a controller for robotic movement, and many other learning opportunities that will come with the project

- Luca Gigliobianco

Function

The most obvious constraint we have to work with is that we are building on top of already completed work. This means any issues that may have come up during the previous development now becomes our issue, and any design choices made previously may determine the way we have to approach our solutions. One issue in particular that has already drawn concern is some possible compatibility issues, both in the version of ROS the previous team used which does not work with newer versions of Ubuntu, and the older versions of python that were used. We plan to mitigate these issues by working closely with the previous teams, and hopefully this either will be less of a concern than we assumed, or we can find methods around having to go about completely porting all previous work to newer versions of software.

Another potential drawback would be the costs to any parts we may have to buy, those being an Arduino and Nano boards, as well as a Stereoscopic camera for testing computer vision software. Our sponsor has already made us aware of the possibility to get a grant that could cover these costs should we need it; however, current cost projection doesn’t seem to be too high even if we are unable to obtain the grant.

One final constraint would be the importance of maintaining stability of the rover at all times. While it would be great to allow the arm to freely move at any speed or direction, the reality is that giving the arm too much free range of motion will cause it to tip over, and this is an even greater concern if it were to be sent on lunar or mars missions where

the gravity is lower. As computer science students, this forces us to think of factors that we aren't typically used to, so we will have to step slightly out of our comfort zone when dealing with this problem.

Legal, Ethical, and Privacy Issues

Legal

When developing the software, it is important that we credit any sources that may have given us information or ideas that come to fruition in the project. It is also critical that the team members do not take any code or designs from any source that isn't public.

Our team will be working with engineers from NASA that develop and design RASSOR units, so it is paramount that the design and software development don't infringe their work. Although the project is open source, taking any work from private sources may result in legal issues later in time.

Ethical

The software developed is intended to control robotic systems on the moon, therefore we will ensure that our work is as fault proof as possible. Errors at this level can jeopardize any missions performed by the EZ-RASSOR and cause expensive damages. It is also possible to put people at risk because of mistakes.

Additionally, our team will have to be concerned about possible expenses during testing. We will have to be as vigilant and careful as possible to ensure that we don't damage any parts during the testing phase. The funds for this project will come from the sponsor, so our group must be as frugal as possible when handling their resources.

Privacy

EZ-RASSOR software is completely open source and can be utilized by anyone. Any information collected by EZ-RASSOR is strictly for developmental reasons. Finally, any personal information pertaining to the team members will not be made public and omitted from any documentation.

Any other privacy concern might be due to the testing/development parts of this project. For example, metadata, images taken by the EZ-RASSOR, comments in the code, etc. Any concerns over intellectual domain would be a legal issue.

Requirements

EZ-RASSOR Requirements

- Build the software for a fully functional arm on the EZ Regolith Advanced Surface Systems Operations Robot (EZ-RASSOR)
- Must take in input from the stereo camera, and using that video navigate the arm to pick up the tiles and place them in the designated spot
- Robot must be able to keep balance through navigation and linking/ laying down the tiles and not tip over in the middle of operation
    - Potentially be able to stand itself back up if the EZ-RASSOR somehow manages to fall over
- The arm must be able to operate in multiple modes
    - User Operated Mode
        - Controller has already been built from previous groups.
        - Implement the Arm to be able to be controlled via the user's phone
        - Make software easy enough for people of all ages to use effortlessly
    - AI controlled Mode
        - AI will need to have full ability to use the arm and build the landing pads on its own
        - Uses well known AI software and countless trials in the software simulation environment in order to build the perfect AI controls
    - Swarm AI
        - Using the AI mode stated above, it must work in conjunction with numerous over EZ-RASSOR robots in order to build efficient as a "team" or "swarm"

- Work with previous and current EZ-RASSOR senior design teams, as well as the robot's creator to seamlessly attach the arm to the existing robot

Programming Requirements

- OpenCV Library
    - OpenCV is a library used for computer vision for programming languages C++ and Python
    - Use the built-in functions to help identify objects as the robotic arm is functioning
- Robot Operating System (ROS)
    - Use version ROS Melodic
    - Install *rosdep* tool to help install ROS dependencies
- Python
    - Use python version 2.7
    - Using pythons built in methods and computer vision libraries to help the EZ-RASSOR navigate
- C++
    - Use C++ for board integration for the robot arm to be functional within the already functional EZ-RASSOR

Hardware Requirements

- Arduino Board Integration
    - Must be able to integrate Arduino boards into the arm to function with the rest of the EZ-RASSOR
- Camera

    - stereo vision Intel real-sense D435i
    - Primary way of receiving video input in order to navigate EZ-RASSOR
- Nvidia Nano
    - Micro development kit to help the arm function

Operating System

- Linux Operating System
    - Ubuntu Desktop
        - In order for most of the software to work, we are required to use the Linux operating system due to the fact that it is open source. The previous EZ-RASSOR teams have used Linux before and it is in our best interests to use Linux to make sure there are no continuity errors.

Specifications

EZ-RASSOR Robotic Arm

- 5 joints for 5 different points of rotation
    - Area of motion covers around half of a meter (0.5 m)
- Be able to lift up to eight kilograms/ 17 pounds
- Arm must be 3D oriented in order to fit the rest of the robotic build/ material
- Combines robotics from the NASA mini-RASSOR
- Software and hardware must fit under budget. Due to this project behind funded by the Florida Space grant, all of our hardware and software must be able to be within that price

- Must be able to run from Nvidia Nano controller

Software Specifications

- All code written over the course of senior design will be all open source and posted to the Florida Space Institute (FSI) GitHub
- Software must be professional and follow best professional practices
- All programs must be under the same versions and updates as the rest of the EZ-RASSOR/ RE-RASSOR teams in order for all programs to run error free
- Operating System for each desktop must be in the Linux Operating System

Budgeting

So far, an accurate measure of the financing can't be calculated since we will need to solidify the numbers further in the planning phase. However, a preliminary estimate can be given. An Arduino Mega 2560 REV3 is valued at $40, a Nvidia Jetson Nano Developer Kit is $60, and an Intel RealSense Tracking Camera T265 is $200. This makes the cost for the known hardware used $300. Costs are expected to rise during development and planning.

This cost is associated with our group, but the total estimated cost for the robotic system is at around $8000.

List of Ideas

Cooper Urich:

- Add a function to the already existing application to account for the arm
    - Switch application into a different setting or even be able to navigate the EZ-RASSOR main body and the Arm at the same time.
- Create an instruction video on how to use the EZ-RASSOR robotic arm

- Create code that can function if another arm were to be places on the EZ-RASSOR for maximum efficiency and no need ro reprogram entire bot
- Have all code on a team GitHub page for easy transitions

Robert Forristall:

- Use ROS along with python's rospy library to create controllers for the functions of the arm
- Integrate the controllers created in rospy using an Arduino or Nvidia Nano board with C++ on the rover
- Utilize the arm's stereo vision camera with a CNN that can take the image from the camera and use the information to ensure the regolith pavers are placed in the correct spot
- Test this solution using the already created simulated environment before moving on to real world testing

Christopher Jackson:

- Create an API that would allow future teams or contributors to easily work with the arm software
- Build on top of the already made mobile device controller to allow for manual control of the arm
- Collect data on the current status and stability of the rover that could limit the range of motion of the arm both in autonomous mode and manual mode
- Add a live camera feed to the mobile controller through the arms stereoscopic camera
- Use C++ for board integration and python for computer vision with OpenCV and making scripts
- Use the computer vision algorithms not only for autonomous use of the arm but also for detecting potential collisions of other objects, particularly when rover is

being used for swarm purposes

Austin Dunlop

- Incorporate autonomous control with manual control of the arm to account for time delay between mission control and rover.
- In manual control, allow precise control of all six joints as well as fluid controls incorporating all six degrees of motion to move to the desired position.
- Design system to still be able to function when robot vision is impaired by regolith.
- Make all software forward compatible.

Luca Gigliobianco

- Maximize the amount of control the user has in every joint and turn to allow the most control
- Minimize delay between image capture and execution.
- Try to do some simulations in groups to see the results and discuss.
- Meet frequently with other software teams to exchange ideas.
- Have everyone dual boot Linux to get best results during testing.
- If a group member has made enough progress in their part, help some of the other members with theirs.
- Log/document test cases to analyze them later.

Broader Impacts

“One small step for a man, one giant leap for mankind”. That’s what Neil Armstrong said as he became the first man to ever step foot on the moon, only 11 other men share the distinction with him of having walked on the moon [1]. Since the Apollo program ended in 1972 nobody has been back, in person that is, and that may soon be set to change. NASA has plans to return to the moon in 2024 [2] and it won’t stop there. There are many whose ambitions for a trip to the moon involve more than taking pictures and collecting rocks (a simplification). The colonization of the moon is the next step. Colonization is a term previously relegated to the exploration and settlement of European explorers in the new world, but now with our world all but explored, it's time to look not beyond the horizon as they did, but above it. To create a sustainable settlement on the moon is a monumental challenge and requires many more advancements in technology to accomplish.

Working in hostile environments like on the moon requires a certain amount of ingenuity to overcome obstacles like abrasive lunar dust, extreme temperatures, $1/6^{th}$ earth's gravity, and no atmosphere leading to increased inertial effects [3]. All new tools and equipment specifically designed to overcome these obstacles are necessary for use in this environment. Vehicles like the EZ-RASSOR are specifically designed with these challenges in mind, both hardware and software aspects. The EZ-RASSOR is one of many pieces in the puzzle of settling on the moon.

The broader impacts of the EZ-RASSOR arm project could be enormous. The general project idea to program an arm that will construct a landing pad on the moon is itself a small and narrow task yet, a stepping stone to greater achievement. The fully functionally programmed autonomous rover will have the capability to build enormous landing pads in efficient time without human intervention in environments unsuitable for standard earth equipment.

So why is this project necessary in the grand scheme? When launching and landing a spacecraft on an adverse plane like the lunar surface, a significant amount of regolith is ejected. This can be damaging to the craft thus necessitating the need for a solid surface to land and launch from. Building landing pads will allow larger and cheaper spacecraft carrying more valuable materials to the moon. This will greatly contribute to the greater goal of the colonization of the moon and other extraterrestrial bodies like mars.

Fig 1. Concept of what a potential moon colony could look like. [4]

Colonization of Extraterrestrial bodies is significant for many reasons. Some people worry of an extinction level event, like the asteroid that wiped out the dinosaurs, that would endanger life on earth. Having sustainable earth colonies on other worlds would ensure the survival of humanity beyond our own world. If the earth dies humanity, everything we are and everything we've done, is gone. Of course, our first priority should be preserving/ saving the earth but it's always nice to have a backup plan, besides in a billion years the earth's oceans will have evaporated due to the sun's expansion anyways [5]. So, we're already on an inevitable ticking clock.

The moon is an ideal starting point for colonization because of its vicinity to earth, only days away vs months or years [3]. The moons hostile environment provides an excellent testing ground for technology that will help in future colonization efforts like on mars [3]. Mars is a much more hospitable place but also much farther away. Due to the moon's decreased gravity, It could be an excellent staging ground for future space voyages.

Research and development from this project can not only benefit the direct project goals but also serve as useful material to build on in future projects of autonomous robot control and robot vision. In this way, the project influence can be even greater. We all stand on the shoulders of giants. Sure, we acknowledge the individual greatness of inventors like da Vinci, Tesla, and Turing but even they didn't do it by themselves, they built upon the little contributions of countless people who came before them.

In 1903, the New York Times predicted it would be at least a million years before man would fly [6]. Just 40 days later, the Wright brothers proved them wrong with the first manned flight. In 1920 they ridiculed the idea of rocket spaceflight. As Apollo 11 soared towards the moon, they issued a correction [6]. There will always be those telling us that our grand ambitions are impossible, and in a way they are right. It is impossible to leap to

the top of a mountain in a single stride, but mountains aren’t climbed in a giant leap, just one small step at a time. The EZ-RASSOR ARM project is a small step in the grand mission to explore the great mystery of our universe and continue to go where no man has gone bef

Research

Software

Python

There are many programming languages that can be used for computer vision, C++ and Java being some, but python makes computer vision so much simpler. The majority of the group is already familiar with computer and robotic vision in python from previous coursework taken at UCF. The previous EZ-RASSOR teams have used python for vision in previous projects, so we decided that there was no other reasonable option other than sticking to what is easiest to implement and hook into the already working robot and its interface.

We are currently using Python 2.7 for our project since ROS Melodic is more compatible with libraries and programs in this version of python. After all simulation work is done, our group will attempt to make plans to update the project to use Python 3. If upgrading the entire project doesn't interrupt our schedule or make the project non-functional, then our group plans to do this over the Summer with the help of the other EZ-RASSOR teams.

Ubuntu 18.04

The OS we are all using for this project is Ubuntu 18.04 LTS. We have all installed it either alongside our native OS or by itself. The reason we aren't using the latest version of Ubuntu is that ROS Melodic is mostly compatible with this version. Newer versions might cause troubles down the line.

If our team can find a way to operate our project with a newer version of ROS, then we can consider upgrading our version of Ubuntu or using other supported Linux versions.

Blender

Blender is an open source 3D computer graphics software toolset. It supports modeling, animation, motion tracking, simulation, rigging, video editing, and 2D animations. This is the software we will use to make the arm model, and to edit or redesign other EZ-RASSOR components.

Gazebo

Gazebo is an open source 3D robotics simulator. It has very good dynamics simulation, using physics engines such as Bullet, ODE, and DART. This platform also offers other features like built-in sensors for video, audio, speed, etc., advanced graphic rendering using OGRE, and a large selection of plugins to name a few. All of these features make Gazebo ideal to use for the testing and development of the EZ-RASSOR arm.

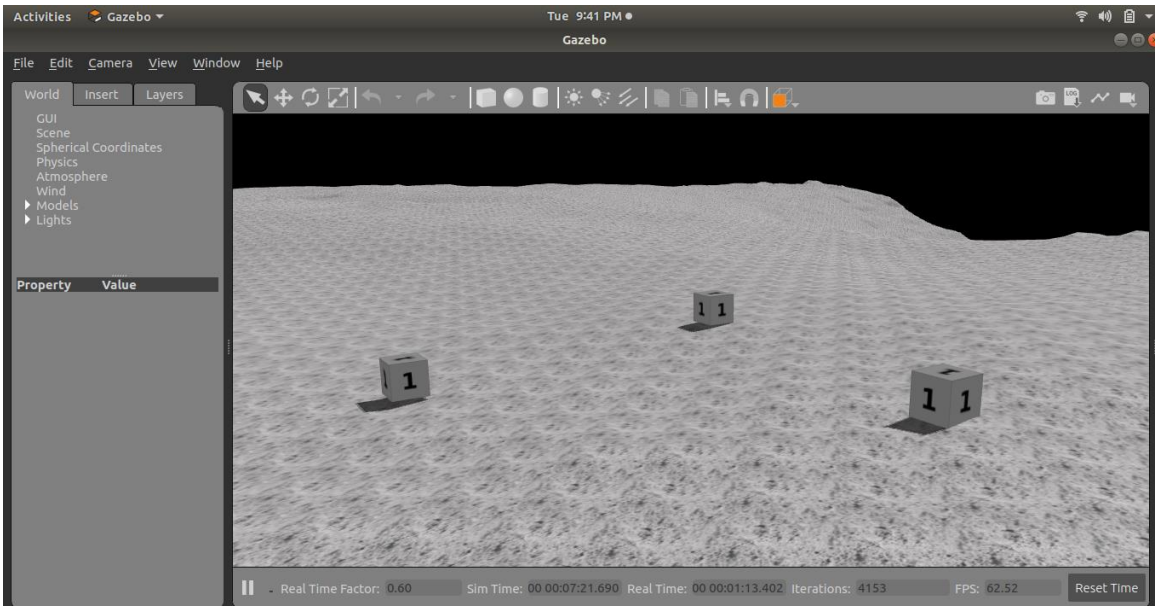


Fig 2. Image of Gazebo lunar environment.

PySerial (Version 3.5)

PySerial is a Python library that encapsulates access for serial ports, providing backends for Python. This library will be very useful for communication between the Arduino board and the Jetson Nano.

OpenCV (Version 4.5.1)

The OpenCV (meaning Open Computer Vision) library is a set of methods and functions made for processing and computing data from images or videos. This library is supported by various programming languages such as Python, Java, and C++. It is supported by the different operating systems such as Linux, Windows, OS X, Android, and iOS [7].

We will be using this software as a way to detect where the pieces of the landing pads will be. We will then move to the marked location with the EZ-RASSOR robotic arm, pick up

the landing pad piece, and then navigate to the correct position by using the stereo camera mounted on the EZ-RASSOR and the OpenCV methods.

MatPlotLib (Version 3.3.4)

Matplotlib is a library for creating static, animated, and interactive visualizations in Python. We will use this library to help visualize the terrain caught by the RealSense Camera.

Intel RealSense SDK 2.0

Intel RealSense SDK 2.0 is a cross-platform library for Intel RealSense depth cameras.

This dependency allows for depth and color streaming and provides intrinsic and extrinsic calibration information. The library also offers synthetic streams (pointcloud, depth aligned to color and vise-versa), and a built-in support for record and playback of streaming sessions.

There are many useful tools that this SDK comes with such as the Intel RealSense Viewer. This is an application that allows the user to view the depth stream, visualize point clouds, record and playback streams, configure camera settings, modify advanced controls, enable depth visualization, and many other options.

The Depth Quality Tool is another application that lets the user test the depth quality of the camera. This is done by measuring standard deviation from plane fit, the subpixel accuracy, distance accuracy, and fill rate.

Additional features that the SDK offers is debugging tools, code samples, and integration with many platforms like Python, C#, Node.js, ROS, ROS2, OpenCV, Unity, and many others. [8]

Open3D (Version 0.12.0)

Open3D is a library that is used for 3D image and data processing. We will use this library to construct our 3D rendering for our virtual environment. In combination with many built in Open3D methods we can construct different Point Cloud renderings of our environment for the EZ-RASSOR robotic arm.

LasPy (Version 1.7.0)

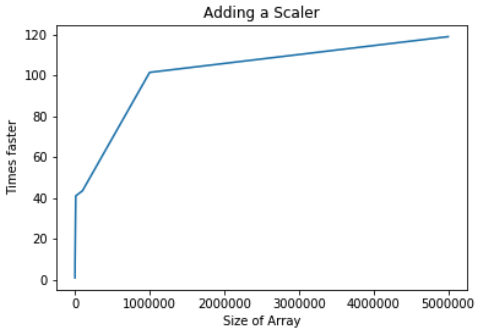
The LasPy Library is a built in library in python dedicated to reading, modifying, and creating LAS LiDAR Files [9]. We will use the LasPy library and some of its built in

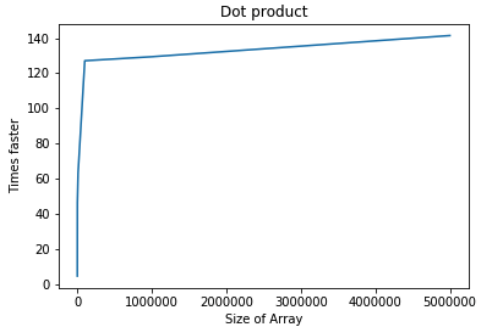
functions for the creation of our Point Cloud Visuals. We will be using its most recent version 1.7.0.

NumPy (Version 1.20.1)

The NumPy library is a python based library, with some methods and functions written in C/C++. This library mostly deals with array manipulation by using topics from linear algebra, matrices, and fourier transformations [10].

The use of NumPy arrays makes array manipulation faster, as it stores the whole array as a unit in one particular spot in memory [10]. Although this initially seems like it would not make the biggest difference in runtime and performance of image and video processing, shaving off milliseconds. Shown below are charts made by Shiva Verma [11] that show the difference in efficiency between different operations using NumPy arrays and lists in python.





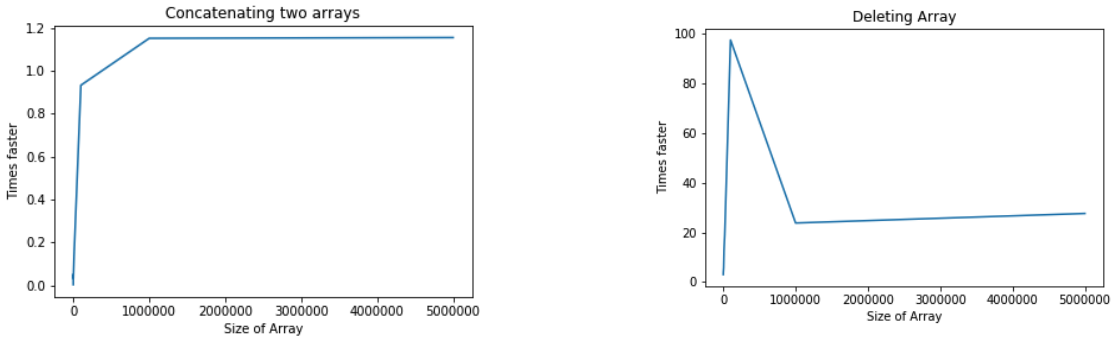


Fig 3. Efficiency of numpy depicted in graphs of time vs array size for different functions

As you can tell from the charts, there is an astronomical difference between the two uses, and when there is constant live video being processed with hundreds and thousands of operations a second, the difference in efficiency could be seconds, minutes, or even hours to do simple tasks. The combination of Python, OpenCV, and NumPy will make the EZ-RASSOR robot much more efficient.

PyTorchCV (Version 1.9)

Another python library that this group has been looking into has been the PyTorchCV library. According to online documentation, PyTorchCV has powerful computer vision methods that would make this project efficiency flourish. This library focuses mainly on segmentation, object detection, and image classification [12].Combining multiple computer vision python libraries could cause some compatibility issues going forward but inversely, having two different options could lead to better efficiency. Using methods from PyTorchCV that would not be as efficient from OpenCV and vice versa would give the group different paths for performance success.

ImageAI (Version 2.0.3)

This python library is very popular and used heavily with object detection. Object detection is important for this project, as the EZ-RASSOR must be able to identify where the tiles are and where to place it. Although the tiles will be 3-D printed and we could essentially hardcode where the robotic arm needs to go in order to pick up the tiles, however we believe it is best to have the arm be able to detect the tiles from anywhere. Thankfully since there

is so much documentation on this library, object detection implementation will be fairly straight-forward and simple.

Conda Python Environment (Version 4.6.1)

Conda is a python environment and package manager. It is used to control and manage different sets of libraries and imports from environment to environment. The upside to using Conda is that you can use different versions of the same library in different Conda Environments, for example one environment can have ImageAI version 2.0.3, while another environment in the same folder can be using an earlier version like ImageAI 2.0.1. It is a great thing to have installed as you will never have compatibility errors due to incompatible libraries.

Linux Operating System

From the very beginning of this project, we knew that we needed to have the Linux operating system on our machines due to the fact that the Robot Operating System (ROS) doesn't work with Operating Systems like Windows or MacOS.

To fix this issue there are two different courses of action that you could take. The first and most common is downloading a virtual machine like VM Virtual Box or VMWare. Then there is the better, yet more complex solution, dual booting.

Virtual Machine

As stated above, virtual machines work as a simple application window that runs a terminal with the operating system of your choice. Some Key files that make up a typical virtual machine include a log file, NVRAM setting file, virtual disk file, and a configuration file [13]. There are some advantages of using it, the most important of which being that it is very simple to boot up multiple operating systems at once, as it doesn't take over your entire systems OS. The one major downside to using a virtual machine is that it isn't as powerful as using a regular operating system, as it is only giving enough processing power to run a normal application [13]. This is why this group, and previous EZ-RASSOR groups alike, have decided to opt for the Dual Boot method of obtaining the Linux operating system.

Dual Boot

Like the virtual machine, Dual Booting is a sure way to get access to any specific operating system that you want and run it as it's intended, not just in an application window like Virtual Machines. Dual booting has a few crucial steps in order to ensure success, and if

done improperly can do irreversible damage to your computer (as one of our groupmates learned the hard way). The first step is to make sure you have a copy of your native operating systems installer downloaded onto a USB drive and a copy of Linux on another separate USB drive. You then want to download a software called Etcher that makes the USB that is holding Linux bootable. The final and most important step is to partition the hard drive of your computer to allow for both operating systems to be able to take over the computer when instructed [14]. After all of these steps are completed you must restart your computer and choose which Operating System to run. And after switching the Linux OS, you will be able to run and code in ROS.

Arduino Software IDE

A program for Arduino hardware can be written in any programming language with compilers that make binary machine code for the target processor. For standard purposes, most coding on Arduino boards is done with the Arduino Software Integrated Development Environment (IDE). The IDE is a cross-platform application that is written in Java, based off of the IDE for Wiring (another platform used to program single-board microcontrollers). It has a code editor with features like text cutting, copying, and pasting, searching and replacing text, syntax highlighting, automatic indenting, brace pairing, and a compile tool. Once code is compiled, the user can upload it to their board. It also contains a console, a message box, and a toolbar with plenty of operations and buttons.

The Arduino IDE supports C and C++ with special rules for structuring the code. The IDE uses a software library from the Wiring project that provides many common input and output methods. Any code made by the user needs only two functions: one to start the sketch and another that is the main program loop. They are then compiled and linked with a program stub into an executable program with the GNU toolchain. The IDE theen uses the program called avrdude to convert the executable code into a text file in hexadecimal that is then booted into the Arduino board by a loader program.

Sketches

Programs written using the IDE are called sketches. These sketches are written in the text editor and are saved with the file extension .ino. The message area gives feedback when saving and exporting, and also displays errors. The console displays text output by the IDE, including error messages and other information. The bottom right hand corner of the window displays the configured board and serial port. The toolbar buttons allow you to verify and upload programs, create, open, and save sketches, and open the serial monitor.

Additionally, the IDE has many more options for sketches in the toolbar. The options in the ‘Sketches’ tab are:

- Verify/Compile: Checks sketch for errors when compiling it. Reports memory usage for code and variables in the console.
- Upload: Compiles and loads the binary file onto the connected board through the specified port.
- Upload Using Programmer: This option will overwrite the bootloader on the board. However; it allows you to use the full capacity of the flash memory for the sketch. To revert this, use the options under the ‘Tools’ tab called ‘Burn Bootloader’ to restore it and be able to upload to the USB port again.
- Export Compiled Binary: Saves a .hex file that can be kept as an archive or sent to the board.
- Show Sketch Folder: Opens the current sketch folder.
- Include Library: Adds a library to your sketch by inserting #include statements at the start of your code. This menu option also gives access to the library manager where you can import new libraries from .zip files.
- Add File: Adds an additional file to the sketch. The file is saved to the data subfolder of the sketch, meant to be used for things like documentation. The sketches in the data folder aren’t compiled by the sketch program.

The toolbar options for the ‘Tools’ tab are:

- Auto Format: This formats your code in a structured manner, for example indenting nested statements or lining up braces.
- Archive Sketch: Archives a copy of the current sketch in .zip format. The archive is saved in the same directory as the sketch.
- Fix Encoding & Reload: Fixes any possible differences between the editor char map encoding and other operating systems char maps.
- Serial Monitor: Opens the serial monitor window and starts the exchange of data with any connected board on a selected port. This can reset the board.
- Board: Select the board currently being used.

- Port: This menu contains all the serial devices on your machine.
- Programmer: Meant to select a hardware programmer when configuring a board without a USB connection. Mostly used when burning a bootloader onto a new microcontroller.
- Burn Bootloader: Allows you to burn a bootloader onto the microcontroller on an Arduino board.

Sketchbook

The Arduino IDE uses a sketchbook for organization: a place to store your sketches. The sketches in your sketchbook can be opened using the 'File' tab and then the 'Sketchbook' menu, or from the 'Open' button on the toolbar. The sketchbook directory can be changed using the menu.

Uploading

Before uploading any sketches, you must select the correct items from the 'Tools' tab and then the 'Board and Tools' menu followed by the 'Port' menu. Selecting the proper port depends on the OS used and possibly the board, investigate properly before selecting. Once the correct serial port and board have been chosen, press the upload button in the toolbar or select the 'Upload' tab from the sketch menu. Arduino boards will reset automatically and begin the upload. Some older boards that don't have auto-reset and will need to have their reset button pressed before the upload. On most boards, the RX and TX LEDs blink as the sketch is uploaded. The IDE will display a message when the upload is complete, or show an error.

Whenever a sketch is uploaded, a program loaded onto the microcontroller called the Arduino bootloader handles the request. It allows you to upload code without using any additional hardware. The bootloader is active for a few seconds when the board resets; then it starts whichever sketch was most recently uploaded to the microcontroller. The bootloader will blink the pin 13 LED when it starts.

Libraries

Libraries allow access to more functionality for use in sketches. To add a library to a sketch using the toolbar, select the 'Sketch' button and then choose the 'Import Library' menu. This will insert one or more #include statements at the top of the sketch and compile the library with your sketch. Libraries take up extra space in the program, so any unutilized libraries should be deleted by simply removing the #include statements from the sketch.

Libraries can either be included with Arduino software, downloaded from the ‘Library Manager’ or importing one from a .zip file.

Serial Monitor

This displays data sent from the Arduino board over USB or serial connector. To send data to the board, enter text and click on the ‘send’ button or press enter. Choose the baud rate from the drop-down menu that matches the rate passed to ‘Serial.begin’ in your sketch.

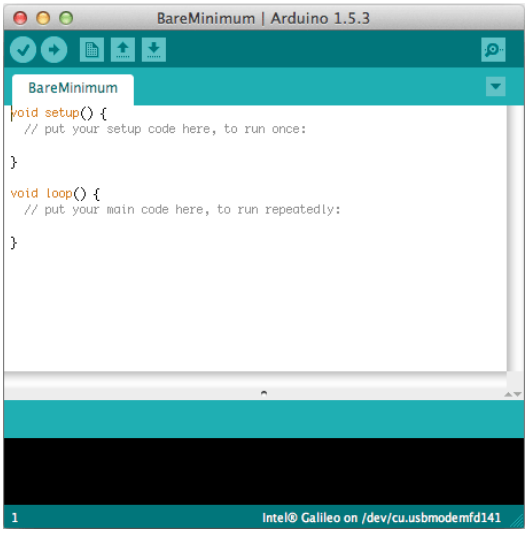


Fig 4. An image of a simple sketch in the Arduino IDE.

ROS Simulation Research

ROS, or the robot operating system (not your typical operating system), is the software library of the EZ-RASSOR robot. This section goes over ROS architecture as it specifically relates to the EZ-RASSOR simulation. The ROS catkin is a workspace that contains build system macros and all the infrastructure for ROS [15]. The use of the catkin workspace in assembling ROS allows greater portability, cross compliance, and package distribution [15]. This is an upgrade from rosbuild, the original workspace. The image below shows all the packages in the workspace of the current EZ-RASSOR project.

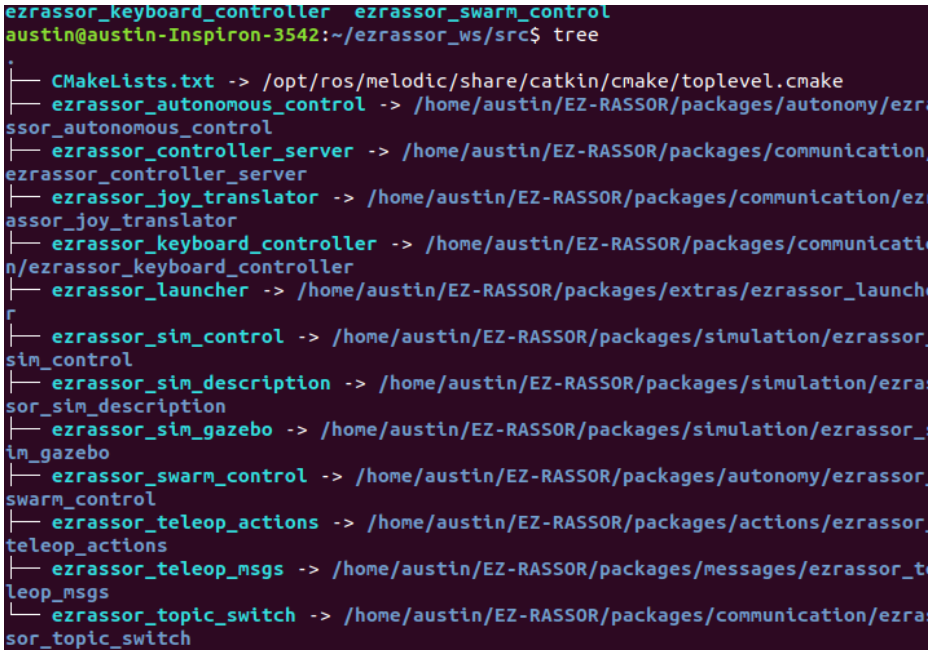


Fig 5. Packages in EZ-RASSOR workspace

Several packages in the current EZ-RASSOR are used in launching a simulation, depending on the parameters passed to it. The most basic simulation only uses the three sim files to load the robot in an empty environment: erassor_sim_description, ezrassor_sim_gazebo, and ezrassor_sim_control. The Robot description package contains all the files which describe the look and function of the arm within the simulation. This is mainly through the unified robot description format (URDF). This package contains specifications for the robot's models, sensors, etc.

According to the ROS wiki on ROS.org [16], URDF includes:

- Kinematic and dynamic description of the robot
- Visual representation of the robot
- Collision model of the robot

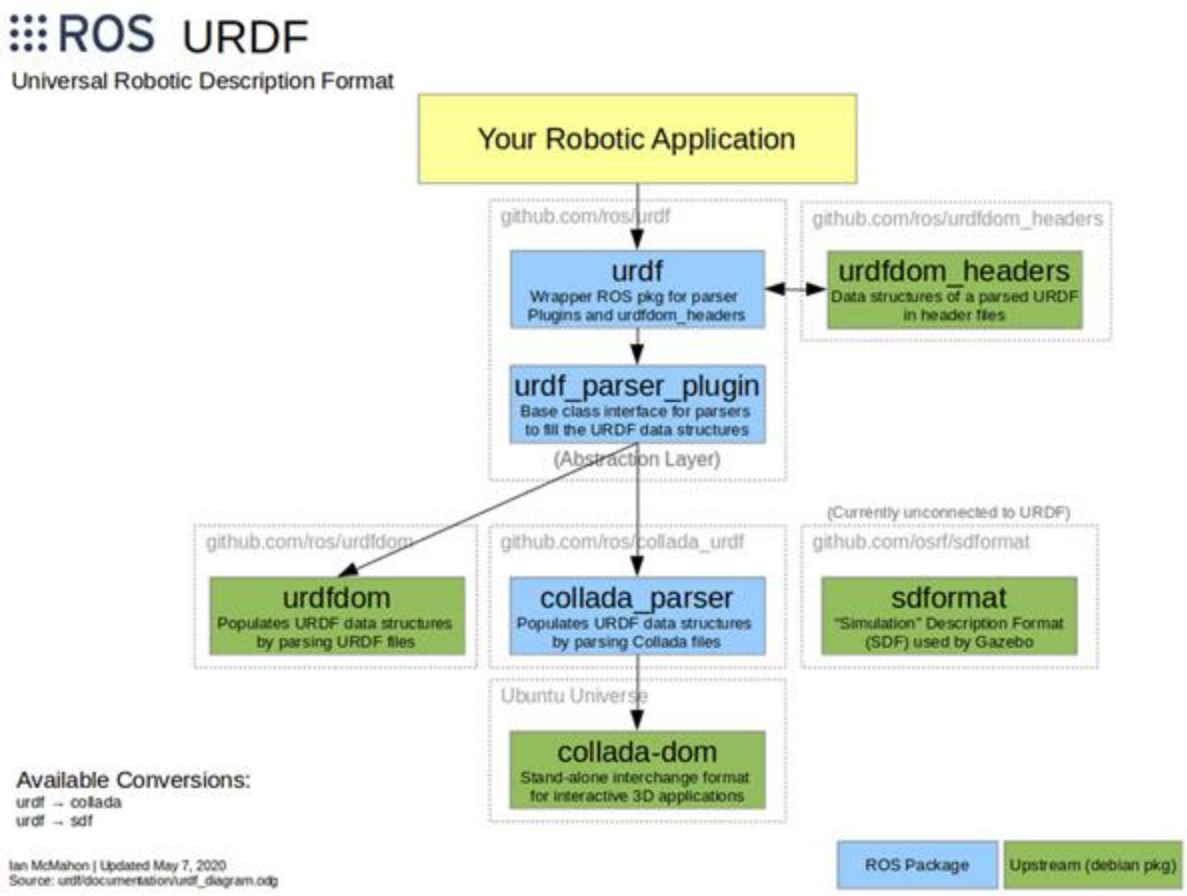


Fig 6. URDF visual [15]

Xacro is a macro language designed for increased readability and maintenance in the robot description files. This is used instead of the URDF format or more precisely, on top of URDF. It shortens and simplifies the URDF file [17].

For assembling a robot in URDF, the robot's moving parts are split into links and those links are connected with joints.

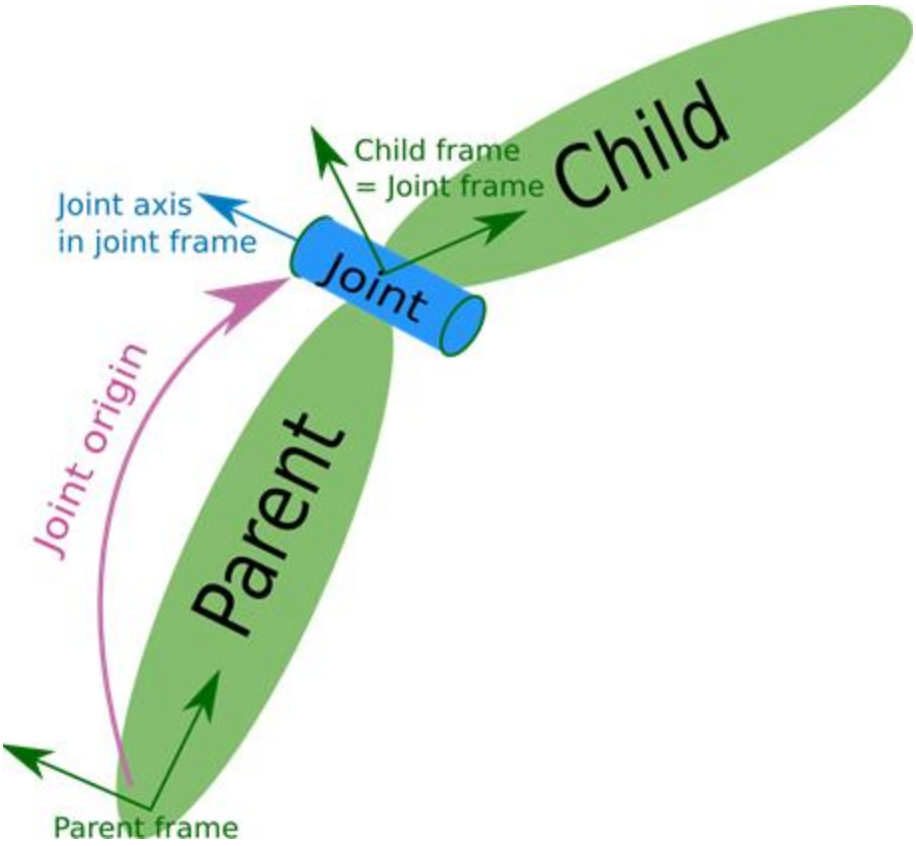


Fig 7. URDF/XML/joint visual [18]

There are six types of joints and are specific to the kind of movement the robot link will have in relation to its parent link. Revolute is a hinge with limited range. Continuous is a hinge with no limit. Prismatic slides along a given axis with limited range. Fixed is a non-moving joint. Floating allows for complete freedom of motion. Planar allows perpendicular motion to a given axis [18].

A more complicated robot, like fig.7, has many links with joints connecting them. The obvious limitation of URDF is that it can only describe robots in a tree structure and only rigid elements [19]. But for robots that fit into these limitations, it provides a simple and more general way to describe said robot.

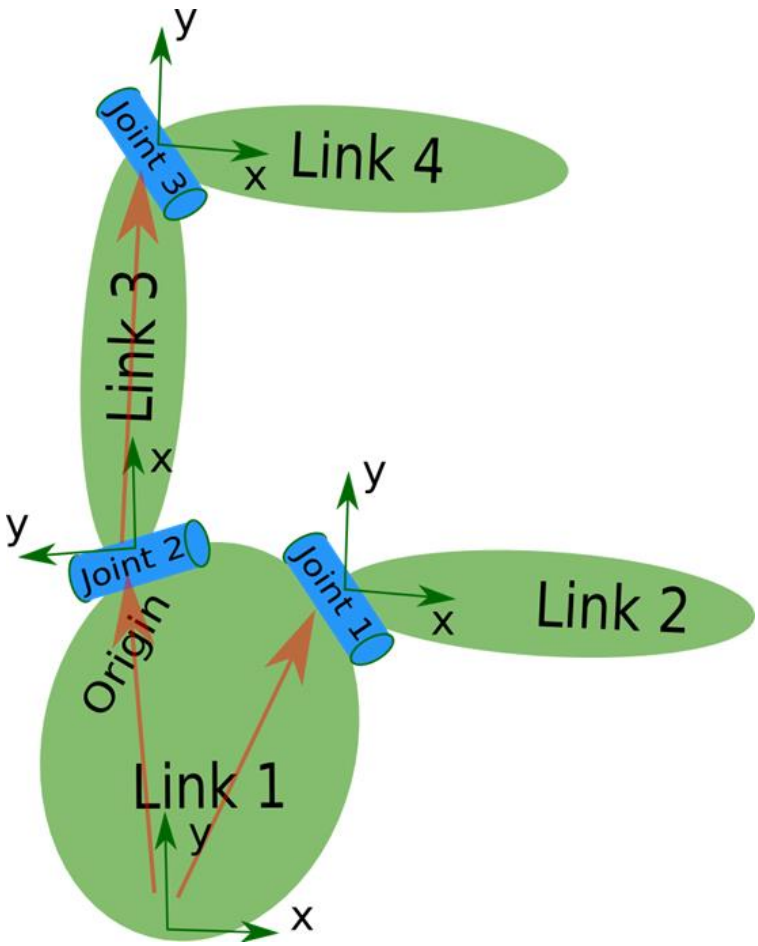


Fig 8. URDF robot visual [19]

Blender Research

Blender is a 3D modeling software with near limitless potential. Though blender contains many features including animation, the blender research was limited to core concepts and later expanded when new models of lesser resolution were needed. Initial research included movement, orienting, exporting, and arranging objects in blender [20].

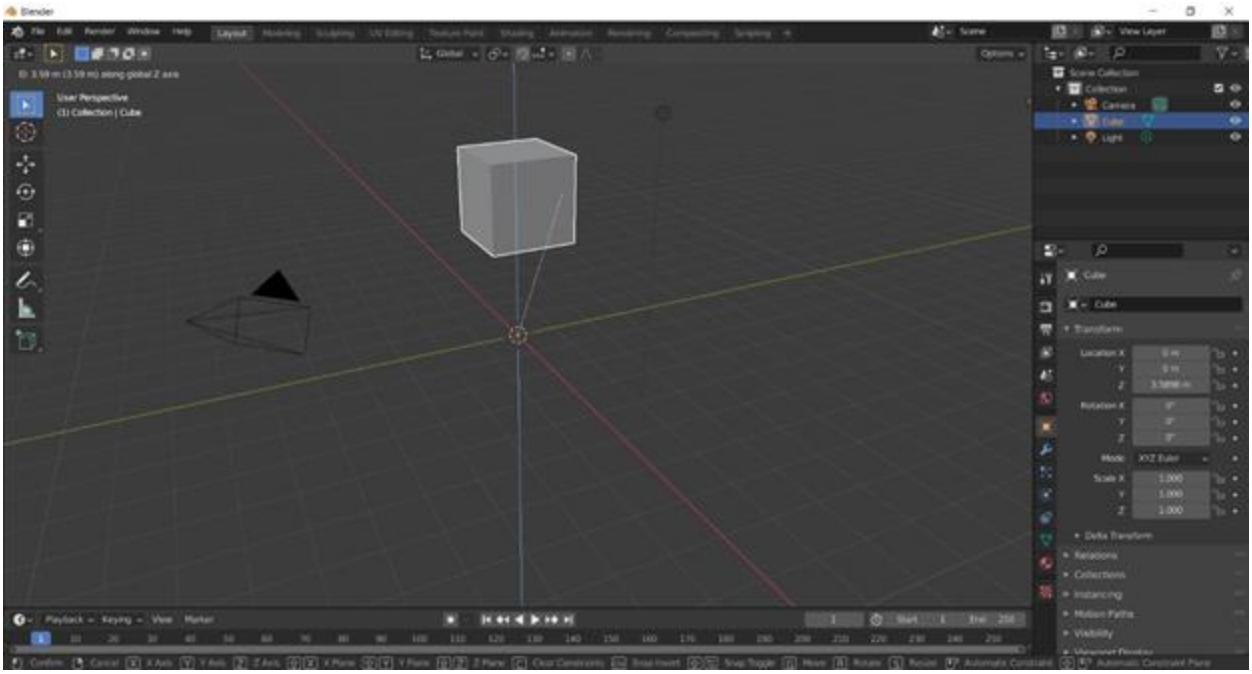


Fig 9. Blender workspace with a cube shape rendered

Blender uses many hotkeys to make working with objects (like the cube in fig.8) more efficient, ‘g’ to grab and move objects, ‘r’ to rotate, etc.

Further research into blender concerned creating shapes to approximate the arm model in a lower resolution version. This involved researching the basic modeling aspects of Blender with shapes. There are three main modes of blender: object, edit, and sculpt. Once a shape is created in object mode, it can be modified in edit and sculpt. ‘Box Trim’ in sculpt mode was the major editing tool used to approximate the Cylinder shapes to arm links.

Gazebo Research

Gazebo research was minimal. Simply using the existing gazebo environment setup by past teams while implementing new models for testing purposes. The groundwork for simulating the robot is done in ROS while gazebo launches this model and places it in the simulated environment. This environment may be a very basic flat plane or the more complicated render of the moon with all its physics involved, like with the command: “roslaunch ezrassor_launcher configurable_simulation.launch \control_methods:=keyboard \world:=moon” (below).

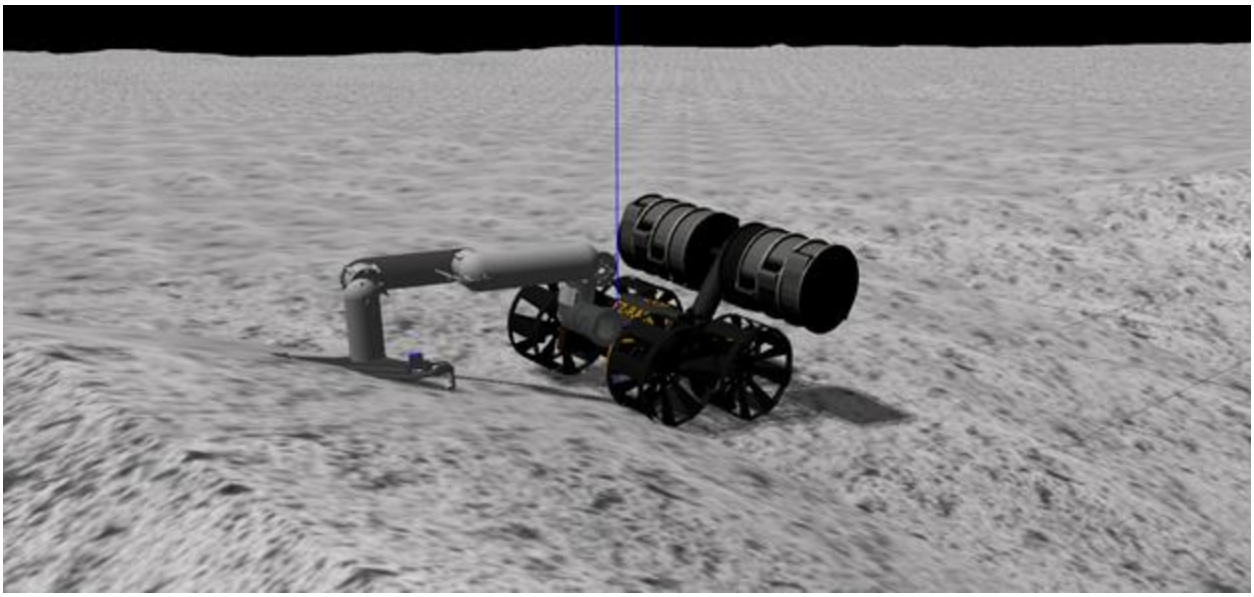


Fig 10. Early EZ-RASSOR model simulated on the lunar environment in Gazebo

A full list of launch commands was provided by previous teams who also heavily recommended we run the simulation on a Linux system and not through a virtual machine.

Arm Design research

Information about the arm started with project sponsor Mike Conroy who introduced us to the concept. Additional materials have subsequently been provided by arm designer Jacob Bruckmoser. The arm was designed in Autodesk Fusion 360 and designed to be added to the EZ-RASSOR rover as an additional appendage for increased functionality. This is the model of the rover with the arm addition as the final simulation model should look as modeled in Autodesk (below).

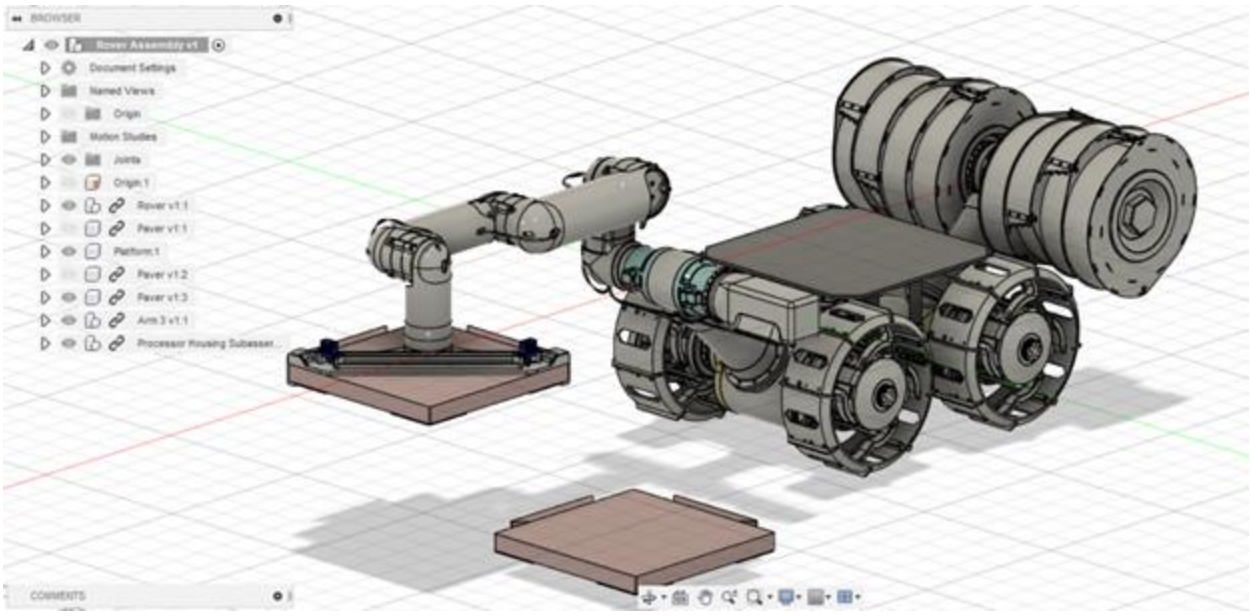


Fig 11. Rover with arm design as seen in Autodesk Fusion 360

The Autodesk file, along with a video demo, also provided by Jacob, provide the best initial demonstration of the Arm's look and function.

State Machines

A state machine, otherwise known as a finite-state machine, is a mathematical model of defining behavior or computations. It is abstract and can only be in one of its states at any moment. The state machine can change states when given a certain input, this is called a transition. A state machine is defined by its initial state, the inputs that cause transitions, transition functions, and the other possible states. Finite-state machines can be either deterministic or non-deterministic, where any non-deterministic machine can be made into a deterministic one.

Types

Deterministic finite-state machines take in regular languages (a language that can be represented by regular expressions or finite-state machines). They can have only one transition function (δ) for every input in the input alphabet (Σ) from each state.

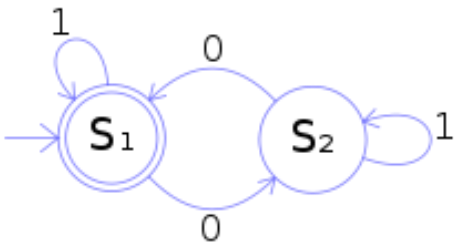


Fig 12. A state diagram for a deterministic automaton.

Non-deterministic state machines are very similar to deterministic ones, with the exception that there can be more than one transition per state per input. They also don't require that every state have a transition for every input symbol and can transition on the empty string (or null) denoted by ε.

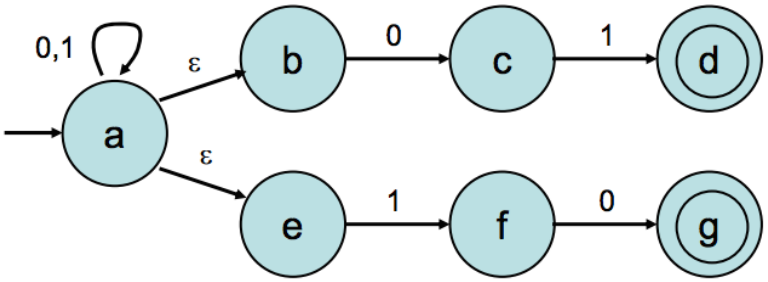


Fig 13. A state diagram for a non-deterministic automaton.

The mathematical definition of a deterministic state machine is a quintuple $(\Sigma, Q, q_0, \delta, F)$:

- Σ is the input alphabet, a finite and non-empty set of symbols.
- Q is the states, finite and non-empty.
- q_0 is the initial state, where $q_0 \in Q$
- δ is the transition functions, where $\delta: Q \times \Sigma \rightarrow Q$
- F is the set of the final states, where $F \subseteq Q$

The definition of a non-deterministic state machine is also quintuple $(\Sigma, Q, q_0, \delta, F)$:

- Σ is the input alphabet, a finite and non-empty set of symbols.

- Q is the states, finite and non-empty.
- q_0 is the initial state, where $q_0 \in Q$
- δ is the transition functions, where δ: $Q \times \Sigma \rightarrow P(Q)$
- F is the set of the final states, where $F \subseteq Q$

Representations

State diagram: a type of diagram used to describe the behavior of systems. State diagrams must have the system that they describe have a finite number of states. Each state is represented by a circle, the transitions are the arrows, and the conditions for the transitions are the text over the arrows.

State-transition table: a table that shows what possible states a state machine can take based on the current state and the input. They can be either one-dimensional or two-dimensional. One-dimensional tables list all of the possible inputs with their current state, and the next state.

A	B	Current State	Next State	Output
0	0	S1	S2	1
0	0	S2	S1	0
0	1	S1	S2	0
0	1	S2	S1	1
1	0	S1	S2	1
1	0	S2	S1	1
1	1	S1	S2	1
1	1	S2	S1	0

Table 1. An example of a 1D state-transition table.

Two-dimensional tables can be represented in either of two ways, in the first one of the axes represents the current state while the other represents input. This maps to the next state of the machine.

Input / Current state	I_1	I_2	…	I_n
S_1	S_i/O_x	S_j/O_y	…	S_k/O_z
S_2	$S_{i'}/O_{x'}$	$S_{j'}/O_{y'}$	…	$S_{k'}/O_{z'}$
…	…	…	…	…
S_m	$S_{i''}/O_{x''}$	$S_{j''}/O_{z''}$	…	$S_{k''}/O_{z''}$

Table 2. An example of a 2D state-transition table that maps next states.

The other representation for a two-dimensional table is where one of the axes represents the current state, and the other axis represents the next state. This table maps the input of the state machine.

Next state / Current state	S_1	S_2	…	S_m
S_1	I_i/O_x	—	…	—
S_2	—	—	…	I_j/O_y
…	…	…	…	…
S_m	—	I_k/O_z	…	—

Table 3. An example of a 2D state-transition table that maps inputs.

Classification of Finite-State Machines

Finite-state machines can be placed into four distinct categories: transducers, acceptors, classifiers, and generators.

Transducers are machines that read a string of input and produce output based on input. The two most defining types of transducer machines are Moore machines and Mealy machines (see the section below).

Acceptors produce a binary output that indicates if the input is accepted, with all the states either accepting or rejecting. When all the input has been processed the machine looks at the condition of the final state, and accepts if and only if the state also accepts, otherwise it rejects. Only sequences of characters are allowed as input, not any actions.

Classifiers are very similar to acceptors, except the final state has more than 2 terminating conditions.

Generators are a type of acceptor and transducer that have a single input alphabet. They make an output sequence of acceptor or transducer outputs.

Moore Machines vs Mealy Machines

A Moore machine is a type of deterministic finite-state machine whose output is determined solely by the current state.

The mathematical definition of a Moore state machine is a sextuple (Σ, Q, q_0, δ, λ, F):

- Σ is the input alphabet, a finite and non-empty set of symbols.
- Q is the states, finite and non-empty.
- q_0 is the initial state, where $q_0 \in Q$
- δ is the transition functions, where δ: $Q \times \Sigma \rightarrow Q$
- λ is the output function, where $Q \times \Sigma \rightarrow F$
- F is the set of the final states, where $F \subseteq Q$

A Mealy machine is a type of deterministic finite-state machine whose output is determined by the current state and the current input symbol.

The mathematical definition of a Mealy state machine is a sextuple (Σ, Q, q_0, δ, F, λ):

- Σ is the input alphabet, a finite and non-empty set of symbols.
- Q is the states, finite and non-empty.
- q_0 is the initial state, where $q_0 \in Q$
- δ is the transition functions, where δ: $Q \times \Sigma \rightarrow Q$
- λ is the output function, where $Q \times \Sigma \rightarrow F$
- F is the set of the final states, where $F \subseteq Q$

The biggest and clear difference between Moore and Mealy machines is that Mealy machine's output relies on the state of the machine and the input symbol, while the output of Moore machine only depends on the current state. All Moore machines can be converted into equivalent Mealy machines, but not all Mealy machines can be translated into Moore machines.

Some other general differences are that Mealy machines usually have less states, have asynchronous output, require more hardware to implement, and are faster to respond to input. Moore machines require more states to accomplish the same task, have synchronous output, need less physical components to operate, and respond to input one clock cycle later.

ROS/Other Robot Middleware

There are many frameworks/middleware for allowing communication and control between the program controllers and physical hardware. For each middleware there are advantages and disadvantages that are important to consider when selecting a system for the robot. While this team's choice for middleware is already decided due to the requirement that our code is compatible with the existing codebase, it is worth while to examine and compare others with the one chosen by this team.

ROS

ROS (Robot Operating System) is one of the most accessible and well-known middleware for operating robots as well as the chosen middleware for this project. Being built with flexibility and modularity in mind, ROS has an extensive collection of libraries, functions, and tools to give almost any robot the communication and control it needs [21]. These systems also promote collaboration by making it easy to build off each other's work and integrate easily.

ROS is built around the use of its runtime "graph" which is a peer-to-peer network of processes that are coupled through the ROS communication infrastructure. This allows ROS to include implementations of several communication styles such as synchronous RPC-style communication over services, asynchronous streaming of data through the use of topics, and storing data on a Parameter Server that can be accessed by nodes during runtime [21]. While ROS is not a real-time framework and thus can't handle real-time code that places a time constraint on the actions; ROS was developed with the ability to integrate with other frameworks that do handle real-time code such as the Orocos Real-time Toolkit which is discussed further down.

Some of the functional parts of the runtime graph explained above are packages, messages, services, nodes, and topics. Packages are the main unit of organization for ROS and contain many things usable by ROS such as nodes, libraries, datasets, and configuration files. Nodes are the main unit for the computational level of the runtime graph being scripts that denote processes and computations. Each node generally controls one aspect of the rover such as a sensor, motor, localization calculations, and path planning which allows for the nodes to easily be modular and integrated into different robotics easily. Each node communicates with each other node by sending data structures with corresponding type fields called messages. These type fields are usually a primitive data type or an array of a primitive data type and carry information like the velocity needed for the next joint movement. To properly send messages between nodes while still having them relatively decoupled, ROS uses services in the form of publishers and subscribers. Publishers and subscribers send their messages between each other through the use of topics that ensure a publisher that sends messages about a specific topic are only read by the subscriber subscribed to that topic.

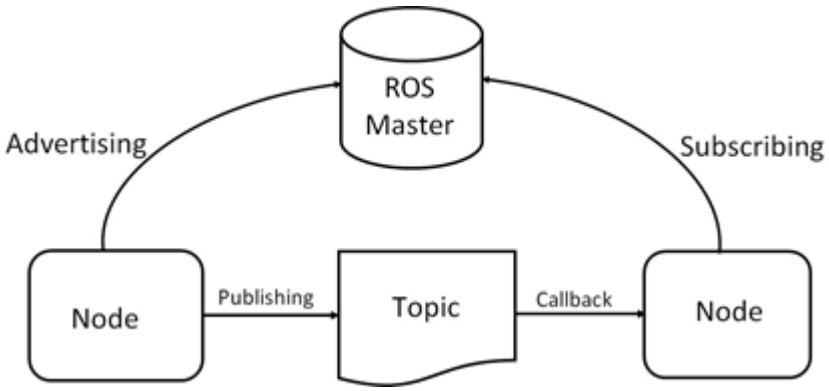


Fig 14. Visualization of the publisher/subscriber system. [21]

ROS also comes with dedicated client libraries for both C++ and Python. These libraries, roscpp and rospy, allow for ros scripts to be written easily using well-known and developed languages. [21]

ROS 2

A newer version of ROS, ROS 2 is very similar to ROS (or ROS 1) but does come with notable changes that make it distinct from its predecessor. First worth noting is a general upgrade to availability and language standards. ROS 2 has compatibility to multiple OS including Ubuntu Xenial, OS X, and even Windows 10 while ROS 1 only had compatibility with Ubuntu 18.04. This OS compatibility makes ROS 2 far more accessible to users since

it has flexibility in what platform it is used on and includes windows which is arguably the most widely used OS by the average computer user. ROS 2 upgrades its used languages including C++11 and python 3 which both have notable upgrades worthy of upgrading for. ROS 1 is limited to C++3 and python 2 which are capable languages but lack a fair bit of functionality when compared to the more modern versions.

Leaving the general changes, ROS 2 has a number of system side changes as well including changes to the build system, messages/services system, and client libraries. The build system is both a bit more flexible but also more rigid, allowing for more options in packages and environment setup but limits non-isolated builds that have multiple packages built using one Cmake file's context. Messages/services mostly have fixes to ensure that generated code uses different namespaces than the .msg and .srv files to avoid collision. The client libraries also got some quality of life upgrades such as subclassing components to allow loading of nodes in more deterministic ways and currently being developed is a method for remapping nodes and topics at startup as well as during runtime. [22]

Player

Player a robot middleware that is quite widely used such as in the Pioneer 2 space probe launched by NASA in 1958. This middleware supports numerous robotic hardware and boasts an active community of contributors that supply new drivers. Focusing on being minimal and adaptable, Player has a more flexible structure in comparison to other middleware which allows for programmers to structure their robot control programs more freely such as having action multithreading or simple action loops. The last notable feature for Player is its ability to handle almost any number of clients along with easily sharing data between them so that a robot can receive sensory information from another robot.

Player defines a set of standard interfaces for the user to utilize that are each tailored to a specific type of hardware that they interact with. One such library is the position2d library that interfaces with ground based mobile robots. The library lets the robot receive commands to move using velocity or position, these commands are converted to usable data by the rover and sent to its driver. Data is also sent back from the rover informing the user of its current state. [23]

YARP

YARP is a robot framework that is built around the idea of being used with other frameworks and robotic operating systems. Lightweight and focused, YARP allows for building robot control systems from an extensive family of connection types that include tcp, udp, multicast, local, MPI, etc. These connections can be swapped in and out as needed

by the program with little issue making it much easier to integrate different hardware as well as modify programs to work with new robots. YARP also includes channel prioritization which allows for connections to be assigned a priority to improve performance and reduce latency. The key downside to the YARP framework is that it lacks much of the functionality that other middleware have such as ROS which includes a proper operating system structure while YARP must be integrated with an existing operating system on the robot.[24]

Orocos

Orocos (Open Robot Control Software) is a widely used and capable middleware that has some powerful tools to offer in regard to programming robots. The Orocos Real-Time Toolkit (RTT) is a component framework that allows the user to write real-time components in C++ that handle the real-time communication and execution of software components.

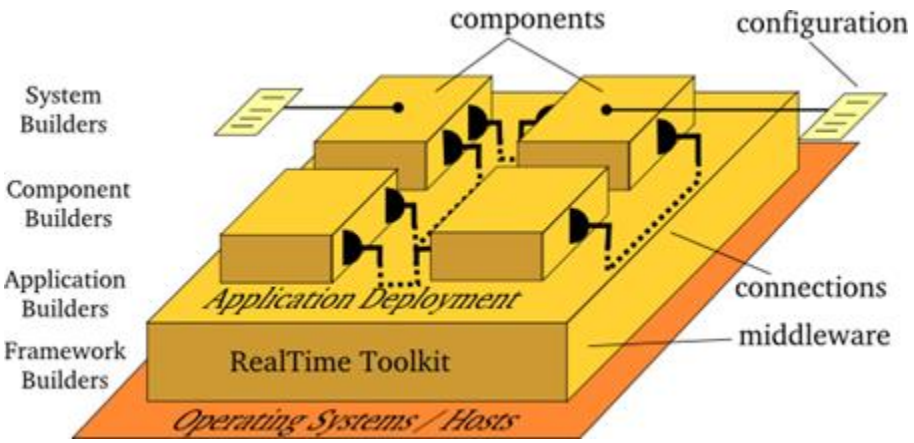


Fig 15. Visual depiction of the Orocos Real-Time Toolkit acting as the middleware[25]

The Orocos Component Library (OCL) is a library for Orocos that contains all the necessary components to start an application and interact with it during runtime. Some of the components included in the OCL are TaskBrowser (an interactive console designed to allow communication with any components), Reporting (a component that reads most data sent between components and periodically writes them to a file for viewing), and Deployment (a component capable of loading and altering components at the same time).

Outside of the two previously mentioned libraries, which are the most important, Orocos also has a few extra libraries focused on more advanced machine/robotic control.
Kinematics and Dynamics Library (KDL) is a framework focused on the computation

and modeling of kinematic chains allowing for 3D frame and vector transformations, kinematics and dynamics of kinematic chains, and kinematics of kinematic trees. Instantaneous Task Specification using Constraints (iTaSC) is a framework for generating motions for a robot through specified constraints between the robot and environment allowing for greater control over a robot's motion in different environments. Bayesian Filtering Library (BFL) is another framework focused on recursive information processing/estimation for algorithms based on Bayes' rule which is widely used in machine learning algorithms. [25]

Edge Detection Methods

This section will cover multiple edge detection methods and their upsides and downsides. The methods listed below are some of the most widely used methods across all computer science. After researching all of the methods we will then come to a conclusion on which method or even multiple methods we end up using during the implementation stage of the EZ-RASSOR Robotic Arm project.

Canny Edge Detection

There are many different algorithms and built in methods from the libraries listed above that successfully detect edges in photos and videos. There is a popular edge detection method already built into the OpenCV library called Canny Edge Detection. Canny Edge Detection works by smoothing out images in order to remove any noise from the frame, then using scans the frame pixel-by-pixel in order to find differences in RGB or Grayscale values [26]. There are criteria for edge for a given image or frame. The criteria includes

1. Detection of edge with low error rate, which means that the detection should accurately catch as many edges shown in the image as possible
2. The edge point detected from the operator should accurately localize on the center of the edge.
3. A given edge in the image should only be marked once, and where possible, image noise should not create false edges [26].

Gaussian Blur

Since some individuals in our group have taken computer/ robotic vision classes, we are familiar with the Canny edge detection algorithm. The algorithm is fairly straightforward

and only requires a few built in methods. Initially, the frame undergoes a process called a Gaussian Blur. The Gaussian Blur uses a standard deviation to generate a particular matrix and uses convolution and its built in algorithms to smooth the image [27] The functions for one dimensional blurs and two dimensional blurs are below [28].

One Dimensional Gaussian Blur

$$G(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{x^2}{2\sigma^2}}$$

Two Dimensional Gaussian Blur

$$G(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \quad (1)$$

Intensity Gradients

The next step of the process includes getting multiple gradients from the image. There are built in function for this in OpenCV, however if we were to implement these ourselves, the images gradient function is as follows:

$$\nabla f = \begin{bmatrix} g_x \\ g_y \end{bmatrix} = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix}, \quad (2)$$

$$\frac{\partial f}{\partial y} = \begin{bmatrix} -1 \\ +1 \end{bmatrix} * \mathbf{A} \quad (3)$$

1. $g_x$and $g_y$are the gradients for each axis $\frac{\partial f}{\partial x}$and $\frac{\partial f}{\partial x}$are the derivatives in respect to the individual axis'
2. The partial derivatives are defined by the convolution equation. **A** is defined by the 3x1 matrix

$$[-1 \;\; 0 \;\; +1]$$

$$\theta = \tan^{-1}\left[\frac{g_y}{g_x}\right] \qquad (4)$$

$$\sqrt{g_y^2 + g_x^2} \qquad (5)$$

3. The gradient direction/ angle can be calculated by taking the arctangent of the two directional gradients

4. Final magnitude is given by the hypotenuse of the two gradients

Non-maximum Suppression

After obtaining the gradient magnitude of the image, a non-maximum suppression algorithm is applied to the image to remove any of the additional noise. This algorithm takes a list of proposal boxes (indicated by **B**), overlap threshold (indicated by **N**), and confidence scores for each proposal box (indicated by **S**). The algorithm then identifies the highest confidence score and selects the corresponding proposal box to work from. Then you create a list that stores the proposals (indicated by **D**) and add the found proposal. You then want to compare any found box **B** and compare it to the rest of the proposals in list **D**. You then want to put the found proposal through a process called intersection over union (IOU) [29]. This takes the intersection of the list of processes and divides it by the union of the whole list. After that is done, any box **B** in list **D** if and only if it is above the overlap threshold **N.** Pseudocode of the noise suppression algorithm presented by *towarddatascience.com* will be shown below.

Algorithm 1 Non-Max Suppression

```
procedure NMS(B,c)
    B_nms ← ∅                                   Initialize empty set
    for b_i ∈ B do                              Iterate over all the boxes
        discard ← False                         Take boolean variable and set it as false. This variable indicates whether b(i) should be kept or discarded
        for b_j ∈ B do                          Start another loop to compare with b(i)
            if same(b_i, b_j) > λ_nms then      If both boxes having same IOU
                if score(c, b_j) > score(c, b_i) then
                    discard ← True              Compare the scores. If score of b(i) is less than that of b(j), b(i) should be discarded, so set the flag to True.
        if not discard then                     Once b(i) is compared with all other boxes and still the discarded flag is False, then b(i) should be considered. So add it to the final list.
            B_nms ← B_nms ∪ b_i
    return B_nms                                Do the same procedure for remaining boxes and return the final list
```

Fig 16. Pseudocode for Non-Maximum Suppression Algorithm [29]

Double Threshold & Hysteresis

After the frame goes through non-maximum suppression, a double threshold is then applied to identify potential edges [26]. The double threshold tracks different types of pixels in order to identify these edges. The three different types of pixels identified are: strong pixels, weak pixels, and non-relevant pixels [30]. Strong pixels are classified as such because they have such high intensity that there is no question that the pixel belongs to an edge. Non-relevant pixels are the exact opposite of strong pixels, as they have such a low threshold that there is a guarantee that the pixel does not belong to an edge. Weak pixels are somewhat in the middle, their threshold is not high enough to be strong, but not low enough to be non-relevant. Weak pixels are flagged and put through a process called Hysteresis [30]. Hysteresis takes weak pixels and turns them into strong pixels if that pixel is neighbors with at least one strong pixel, if not then it is classified as non-relevant.

Completion

After all of these steps are completed, the image of the frame that you have put through the Canny Edge detection should look like an outline of all of the different edges of the frame. Repeating this process for every frame of a live video would show us live edge detection, as it accounts for the edges moving between frames. Using this algorithm and process, we can help the autonomous robots detect where to place down the landing pads when in operation.

Roberts Cross Edge Detection

Convolution Kernels

Another method for edge detection is the Roberts Cross algorithm. This process is similar to the Canny Edge detection method, as it takes two, 2x2 convolution kernels and applies them to the image [31]. This edge detection method is comparable to double thresholding, as it spans the image and highlights areas of high spatial frequency. The two convolution kernels, named $G_x$and $G_y$are rotations of each other.

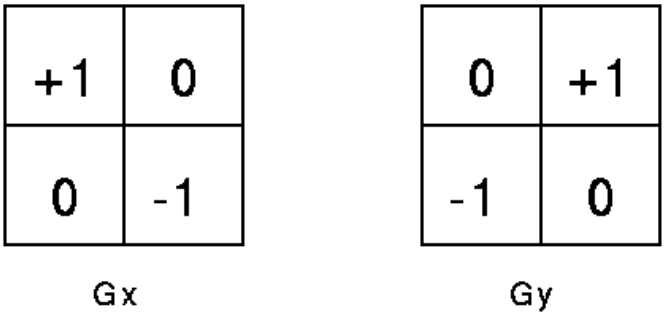


Fig 17. Convolutional Kernels in Roberts Edge Detection. [31]

As you can tell from the images above,$G_y$is simply a 90° clockwise rotation of G_x.

Gradient Magnitude

Finding the correct gradient magnitude is crucial in order for Roberts Cross detection to work correctly. Finding the gradient magnitude for Roberts Cross method is basically identical to the gradient magnitude of the Canny method. This step requires you to find the hypotenuse of the two kernels.[31]

$$|G| = \sqrt{Gx^2 + Gy^2} \tag{6}$$

Although this equation calculates the gradient magnitude efficiently, there is a much faster way to receive the magnitude. By just adding together the magnitudes of $G_x$and $G_y$you there is no need to have an extra calculation, which if there are thousands of calculations being done shaving off an extra operation will save time and increase efficiency. [31]

$$|G| = |Gx| + |Gy| \tag{7}$$

Image Application

An example of this on an actual matrix from a given image would be a 2x2 matrix will values, P_1, P_2, P_3, and P_4. After identifying the matrix, you would find the approximated magnitude by the formula below [31].

$$|G| = |P_1 - P_4| + |P_2 - P_3| \tag{8}$$

Using more than one method of edge detection may lead to success and an increase of efficiency.

Laplacian Edge Detection

The laplacian edge detection method is similar to the Canny edge detection methods and the Roberts cross edge detection methods, it uses an image that has been blurred by a gaussian filter, and then applies a specific kernel to the image. Unlike other different types of edge detection methods, the Laplacian method uses one kernel for the entire filtering process instead of multiple.

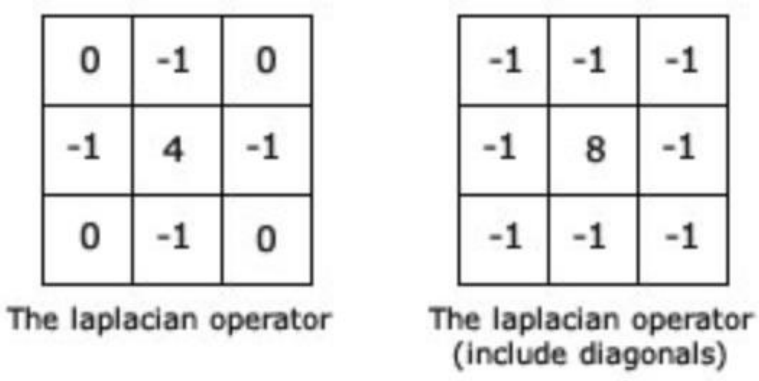


Fig 18. Convolutional Kernel for Laplacian Edge Detection [32]

As shown in the picture, there are two different kernels that you can choose from, one that does the basic edge detection, and the second that includes the diagonals in the kernel. If you choose either of the kernels, it will be the same output. If you wanted to make the

edges more defined and a better approximation, you could just use a bigger laplacian kernel size [32].

Laplaician Method Pros and Cons

The major upside to the Laplaician method is its speed. Since there is only one kernel to compare to, it is much more efficient than Canny and Roberts cross because those two have so many steps. Although Laplacian can be less accurate, it is a huge factor that our team needs to weigh in on, sacrificing accuracy for efficiency is a decision that we will have to come to when we decide which method we go with. Another huge contributor to the fast runtime is the process of zero crossing that the Laplaician method does. Zero cross operators detect not only the edge, but also the orientation of the edge as well. The only downside, like previously stated, is the efficiency of the zero cross operations. If there is a lot of noise in the image or frame that the laplacian method is working on, it does not do as efficient of a job compared to the ore in depth methods [33]. Since the Canny method uses steps like double thresholding and hysteresis, it gets a much better picture to detect an edge/ potential edge in a frame.

Sobel Edge Detection

Like the previous edge detection methods, the sobel edge detection method uses kernels to make an outline of the image in order to detect the edges. The Sobel kernels are similar to the Roberts Cross kernels, as there are two kernels that are rotated versions of each other. Unlike the Roberts Cross method, the Sobel kernels are three-by-three in dimension, allowing for a more precise outline compared to the rest of the methods.

-1	0	+1
-2	0	+2
-1	0	+1

Gx

+1	+2	+1
0	0	0
-1	-2	-1

Gy

Fig 19. Convolution Kernel for Sobel Edge Detection [34]

The reason for these kernels is to take the difference from the right and left of the pixel (for x gradient) or the difference from above and below the pixel (for y gradient). For example if you wanted to get the x gradient of an image. For every pixel you would cycle through the surrounding three-by-three matrix and take the sum of each pixel value multiplied by the specific directional gradient kernel. *Medium.com* has a great example of how this convolution works.

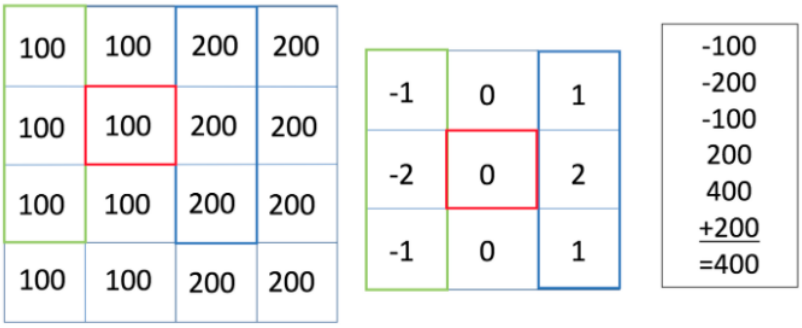


Fig 20. Kernel Convolution Explanation [34]

Shown on the left is the greyscale value of the pixels of an image, the center matrix is the x gradient kernel, and shown on the right is how the final pixel value is calculated. You can see that the pixels above and below the red box are ignored because the gradient kernel has zeros in those positions. The final pixel value will be the absolute value, in case of any negative pixel values (-200 because value of 200).

Sobel Method Pros and Cons

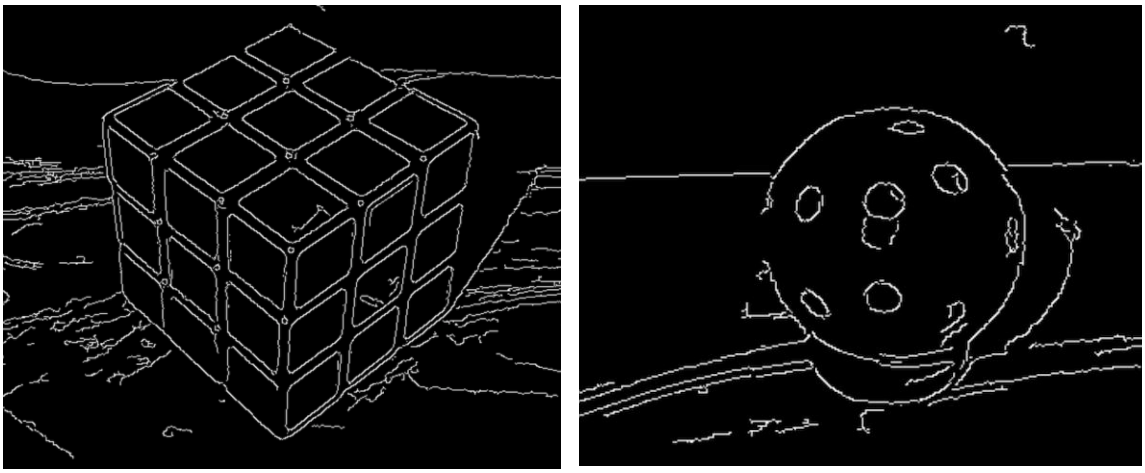
The sobel method is used mostly for simplicity. Compared to some other methods, it takes far less time to come up with an image that outlines all potential edges. Another huge upside to the use of Sobel operators is that they calculate and show direction of edges which could be important for different challenges that we come across. The disadvantage to the Sobel however is that it does not function as planned when the image or frame as a lot of noise present [35]. This is very similar with some of the other edge detection methods, as they sacrifice efficiency for speed.

Canny, Roberts Cross, and Laplacian Method Comparisons

Obviously every edge detection method has its pros and cons, or else there wouldn't be any variation in methods. The canny method of edge detection does out-perform many other edge detection methods, showing why it is the most widely used edge detection method in the software engineering world. Roberts Cross does a decent job at edge

detection, but when compared to the efficiency of the Canny method, it is not nearly as accurate with small details [36]. And like previously stated in the previous section on the Laplacian method pros and cons, the speed is much faster than the Canny edge detection, the efficiency is what makes the performance subpar.

The reason that the Canny method would work best for this project is because it is in real time, using a camera that is not in a controlled environment. Since it will be used outside there could be an abundance of noise in some of the frames. In order to have maximum efficiency, we need to use the method that deals with noise the best. The only downside of the Canny method is that real time computation could take a long time. The built in openCV cv.Canny() in combination with other steps like cv.Gaussian() and cv.threshold(), however can be used and will keep up with real time video with only a slight delay. Shown below are images of a couple of objects being outlined by the built in methods while being captured through the live video feed of a built in laptop camera.



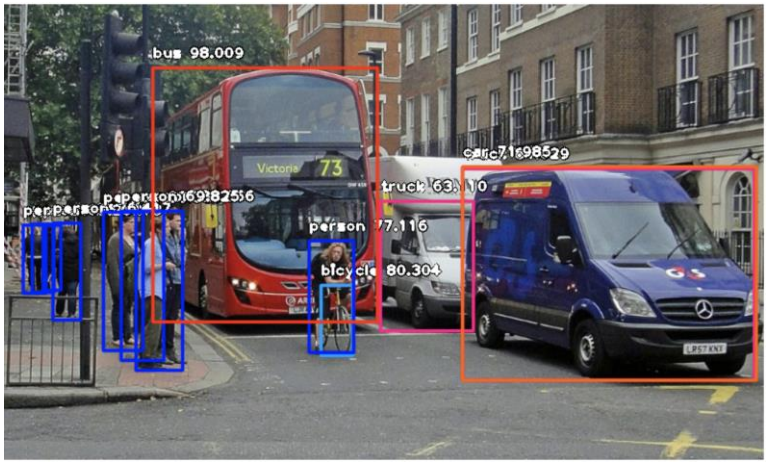
Rubik's Cube Golf Ball

Fig 21. Images taken from a group member's laptop.

Object Detection

After the edges are detected from each frame, it will be easier to detect certain objects within the frame. Like stated before, there is plenty of implementation that will help us with the implementation stage of this project. There are so many built in methods that a

fully working object detection software doesn't take more than around 10 lines of code from start to finish [37]. Object detection works by having pre-loaded images from a folder to compare to the objects found in the image/ frame. Each object that is detected is added to a list, and then we loop through the list and compare each object to the imported images and assign it to the object that it best matches. Shown below is an image that has gone through object detection and the code used.




```
from imageai.Detection import ObjectDetection
import os

execution_path = os.getcwd()

detector = ObjectDetection()
detector.setModelTypeAsRetinaNet()
detector.setModelPath( os.path.join(execution_path , "resnet50_coco_best_v2.1.0.h5"))
detector.loadModel()
detections = detector.detectObjectsFromImage(input_image=os.path.join(execution_path , "image.jpg"),
output_image_path=os.path.join(execution_path , "imagenew.jpg"))

for eachObject in detections:
    print(eachObject["name"] , " : " , eachObject["percentage_probability"] )
```

Fig 22. Pseudocode and Sample Output for Object Detection Method [37]

Video Object Detection

Obviously for object detection to work for this project, it needs to be able to detect objects in real time. Similar to how the Canny Edge detection method works with pictures and videos, we simply treat each frame as its own picture and detect the objects from there.

Gabor Filtering

This type of image filter is a linear filter used for edge detection, texture analysis, and feature extraction [38]. The main use of a Gabor filter is to get a sense of the direction of

edges in each object. If an object has multiple edges going in one direction, it will find all of the edges in the frame going in that direction. The equation, as complex as it is, is shown below.

Complex

$$g(x, y; \lambda, \theta, \psi, \sigma, \gamma) = \exp\left(-\frac{x'^2 + \gamma^2 y'^2}{2\sigma^2}\right) \exp\left(i\left(2\pi\frac{x'}{\lambda} + \psi\right)\right)$$

Real

$$g(x, y; \lambda, \theta, \psi, \sigma, \gamma) = \exp\left(-\frac{x'^2 + \gamma^2 y'^2}{2\sigma^2}\right) \cos\left(2\pi\frac{x'}{\lambda} + \psi\right)$$

Imaginary

$$g(x, y; \lambda, \theta, \psi, \sigma, \gamma) = \exp\left(-\frac{x'^2 + \gamma^2 y'^2}{2\sigma^2}\right) \sin\left(2\pi\frac{x'}{\lambda} + \psi\right)$$

where

$$x' = x\cos\theta + y\sin\theta$$

and

$$y' = -x\sin\theta + y\cos\theta$$

Fig 23. Gabor Filter Equation [38]

We don't know whether or not this equation will be of good use in this project, but we do know how important it is. If we run into issues with the Intel Realsense Camera not being able to detect the image efficiently enough, then we will more than likely end up resorting to using this function. There is a built in function in the OpenCV library that returns the Gabor Kernel. In combination of the two functions *getGaborKernel()* and *filter2D*. This makes it very simple to implement this type of filtering if needed. Due to the efficiency of both Python and the OpenCV libraries, it takes no more than 5 lines of code to pass a live image through this object/ edge detection method.

Motion Processing

Detecting motion between images and through video feed can be done through a variety of methods, but there are two common solutions that we are considering. These are optical flow (or 2D/image motion equation) and background subtraction [39]. Like the different approaches for edge detection, there is no "one size fits all" solution to motion processing, nor is there a method that has no cons. Below we will go into detail about both approaches, their use cases and which solution could be used in this project.

Contour Extraction

Contours are used in computer and robotic vision to detect and identify shapes of different objects and images. It is very similar to how humans can look at a picture and identify what

something is based off of what it looks like [40]. Since methods don't know what an object looks like without being trained first, the object has to be put into a neural network and train the computer in order to correctly identify it in real time. We will discuss neural networks in a future section.

Descriptors

Descriptors are used mostly in combination with Contours and used to highlight and identify different objects or key points in an image. Keypoints are defined as locations of importance in an image, like borders, blobs, or locations that are repeatable [40]. The use of descriptors and Contours could help our group get the accuracy that we need.

Box Bounding

In object detection, the most important step is drawing a bounding box to identify the object that is detected. There are multiple different algorithms or methods used to draw these bounding boxes. The most commonly used method gets the *x* and *y* coordinates of the detected objects upper left corner, as well as the *x* and *y* coordinates of the lower right corner[41]. With these two points, the algorithm can determine where the sides of the boxes need to line up with using basic geometric equations. There are multiple built in libraries used for these calculations, but since we are already using the tensorflow and PyTorch libraries, we will more than likely end up using those in testing.

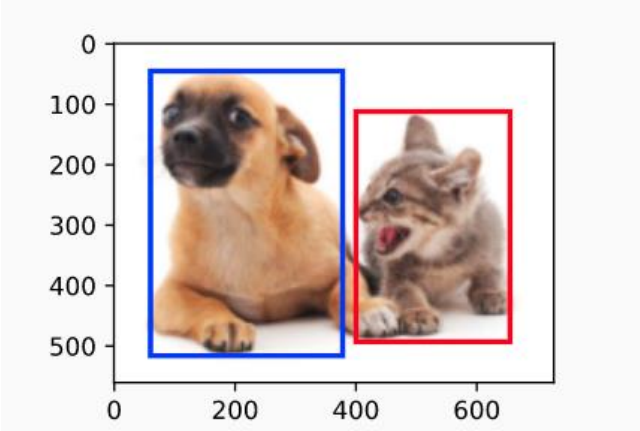


Fig 24. Example of Box Bounding [38]

As you can see, there is a box that roughly outlines both the dog and cat separately. This kind of box bounding will be essential for our project, as our group needs to be able to inform our EZ-RASSOR robotic arm where the individual 3D printed tiles are, and also

where the tile needs to be placed. Although we won't know whether or not the box bounding will be efficient enough to identify the holes in the landing pad in order to place them, the group knows that the boxes will be able to correctly identify the landing pad pieces to pick up.

Location of Object

While attempting to detect different objects in a frame from the live feed, we need to know where the object is in the frame in order to have the EZ-RASSOR robotic arm to navigate to that area and pick up or readjust the tile. We need some sort of way to know the coordinates of the object first. Our plan is to use the coordinates of the bounding box from the previous research section. Using openCV, drawing a box involves knowing the coordinates on the screen, so the robotic arm will know exactly where to navigate to. We will have to test whether or not it is best to navigate in the middle of the bounded box or on the edge, which just requires simple math. After our testing phase of this project we will come to a verdict on where to navigate the arm to using ROS.

Optical Flow Method

Optical flow is an approach that has only started to see practical use within the past seven years that defines motion from a physics and mathematics perspective [42]. Instead of defining motion as the difference or displacement of pixels or specific features in an image between frames, optical flow defines motion as a change in brightness intensity between frames. Below is an image we can use to develop our equation.

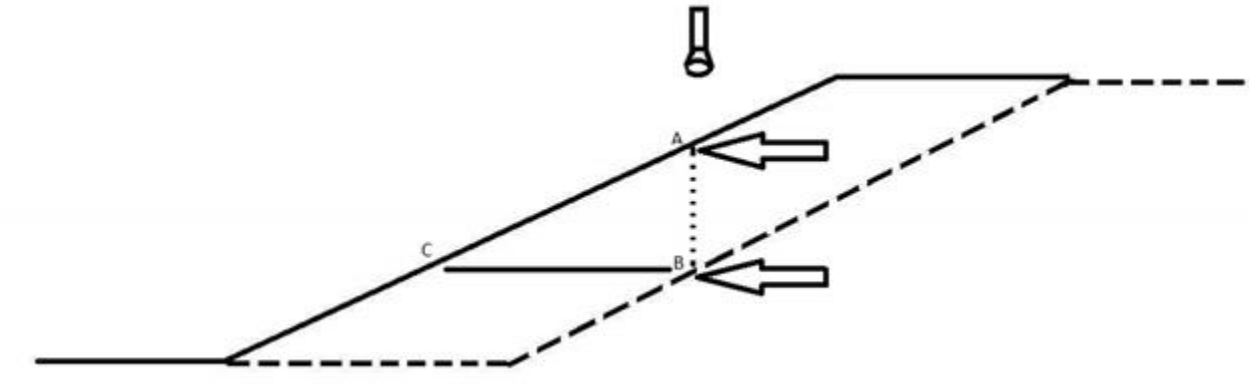


Fig 25. Diagram showing a single pixel camera or sensor detecting brightness drop.[42]

In this image, we have a single pixel camera or sensor that is detecting a change in brightness in the pixel after a certain amount of time, which is given by the points A and B. The distance between these points is called the brightness drop, which can be labeled as

I_t. The amount of brightness drop is dependent on the speed, shown in the figure as the lowermost dashed line which can be measured as the length of the line CB. Notice that if the dashed line for our speed was steeper, the distance between A and B would increase, meaning we would have a greater brightness drop, and a shallower slope would decrease the distance of A and B, meaning a smaller brightness drop, therefore, the amount of brightness drop is dependent on speed. We can label the speed as u. So now that we have I_t for the distance between A and B, and u as the length of the line CB, we finally can get the slope of the triangle that is formed in the figure and label that as I_x, combining each of our labels to create the equation: $I_t = u \cdot I_x$. Since I_t is a drop, we should make this a negative quantity, so our equation becomes: $-I_t = u \cdot I_x$. So far, we only have motion in one direction, so we can add another equation that is the same, with the only difference being that our I_x becomes I_y, and our label for speed becomes v. Then both equations can be combined to make: $-I_{t_x} - I_{t_y} = I_x \cdot u + I_y \cdot v$. Both labels for our drops can be combined into one label, so we get a final equation:

$$- I_t = u \cdot I_x + v \cdot I_y \tag{9}$$

From this equation, we can derive all the information we need to begin making vectors that will represent motion in the scene. We can set up two equations at different points:

$$\begin{aligned} - I_t^{P1} &= u \cdot I_x^{P1} + v \cdot I_y^{P1} \\ - I_t^{P2} &= u \cdot I_x^{P2} + v \cdot I_y^{P2} \end{aligned} \tag{10,11}$$

and solve for u and v when the image gradients are different, which means that we have two separate edge directions. This is necessary, as we cannot solve for u and v if both edge directions are the same. We can use one of these equations to derive what we need to solve for the speed in both directions. Notice that the right-hand side of our equation is the same as if we did a dot product separating the speed components from the slope components, so we will have: $-I_t = [u\ v\ ] \cdot \left[I_x\ I_y\ \right]$. The two vectors on the right-hand side can be simplified into a vector $\vec{v}$ which has components (u, v) and a vector $\vec{\nabla} I$ which has components (I_x, I_y). We now have: $-I_t = \vec{v} \cdot \vec{\nabla} I$, which then can be expanded out to $-I_t = \|\vec{v}\| \times \|\vec{\nabla} I\| \times cos\ (\theta)$. The final thing we must do is separate our known and unknown quantities. The brightness drop and the image gradient slope are knowns, so we can divide both sides by the image gradient to then have an equation that solves for velocity: $-\frac{I_t}{\|\vec{\nabla} I\|} = \|\vec{v}\| \times cos\ (\theta)$. We can draw out each of our vectors against an edge so we can visualize what $\|\vec{v}\| \times cos\ (\theta)$ represents.

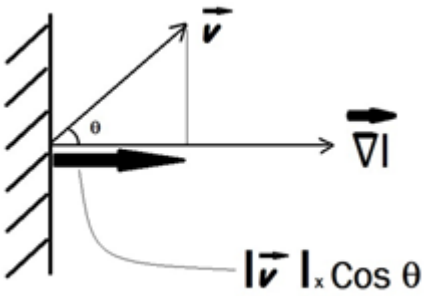


Fig 26. Diagram for each of the vectors in relation to the surface.[42]

Shown by the thick arrow, $\|\vec{v}\| \times cos\ (\theta)$ represents the perpendicular drop from $\vec{v}$ onto the gradient. It may seem like we are done now, but unfortunately, there is one issue. Using one equation, we only get one edge, which means we have a limited view of motion. This is known as the aperture problem.

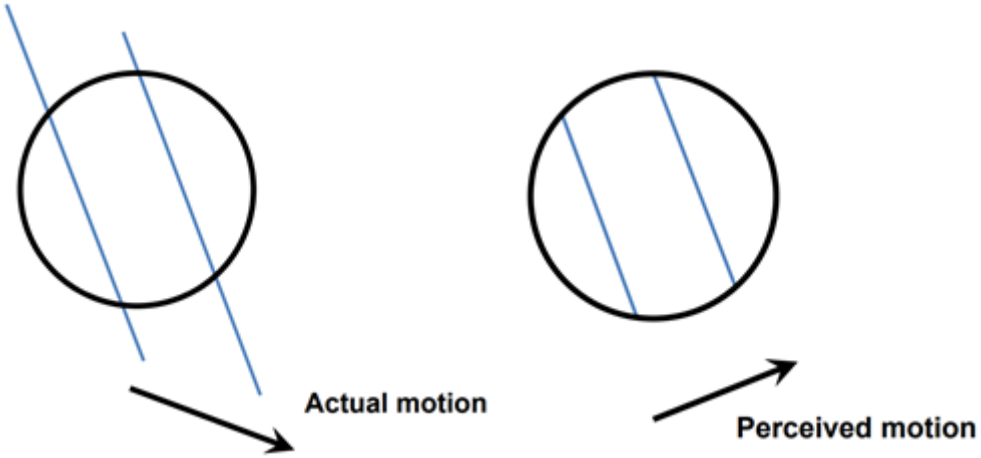


Fig 27. Visualization of the aperture problem.[43]

As you can see above, only having one edge creates a scenario where we can have motion in two directions, but due to our limited view, we only see motion in one direction. As we mentioned earlier, we can add another equation which solves our aperture problem, but only having two equations is unreliable, because both gradients and derivative calculations would need to be perfect for this to work, so in practice, three or more equations are used. With two equations, solving for the perpendicular drop is simple, because we can make a matrix out of the gradient components which will be a 2x2 matrix, so it can be inverted,

but adding more equations makes it a Nx2, and since it is no longer square, inverting is not as simple. First, we set up our equation to include the newly added gradients.

$$-\begin{pmatrix} I_t^{P1} \\ I_t^{P2} \\ \vdots \\ I_t^{PN} \end{pmatrix} = \begin{pmatrix} I_x^{P1} & I_y^{P1} \\ I_x^{P2} & I_y^{P2} \\ \vdots & \vdots \\ I_x^{PN} & I_y^{PN} \end{pmatrix} \times \begin{pmatrix} u \\ v \end{pmatrix} \tag{12}$$

To make things simpler, let's say that $Y = (I_t^{P1}\ I_t^{P2}\ \vdots\ I_t^{PN}\ )$, and $A = \begin{pmatrix} I_x^{P1}\ I_y^{P1}\ I_x^{P2}\ I_y^{P2}\ \vdots\vdots\ I_x^{PN}\ I_y^{PN} \end{pmatrix}$. In order to invert our Nx2 matrix, (which is necessary in order to solve for the velocity vector), we need to left multiply both sides by A transpose, which means the columns of A become the rows, and the rows of A become the columns. We now have: $-A^TY = A^TA \times (u\ v\ )$. Now that our matrix is square we are ready to invert, so our equation becomes: $-(A^TA)^{-1}A^TY = (A^TA)^{-1}A^TA \times (u\ v\ )$, which can be simplified to: $(u\ v\ ) = -(A^TA)^{-1}A^TY$. With this, we now have what we need to solve for the perpendicular drop given a system of equations. The vectors that we are solving for can be visualized by a map that represents the speed and direction that objects are moving in the scene.

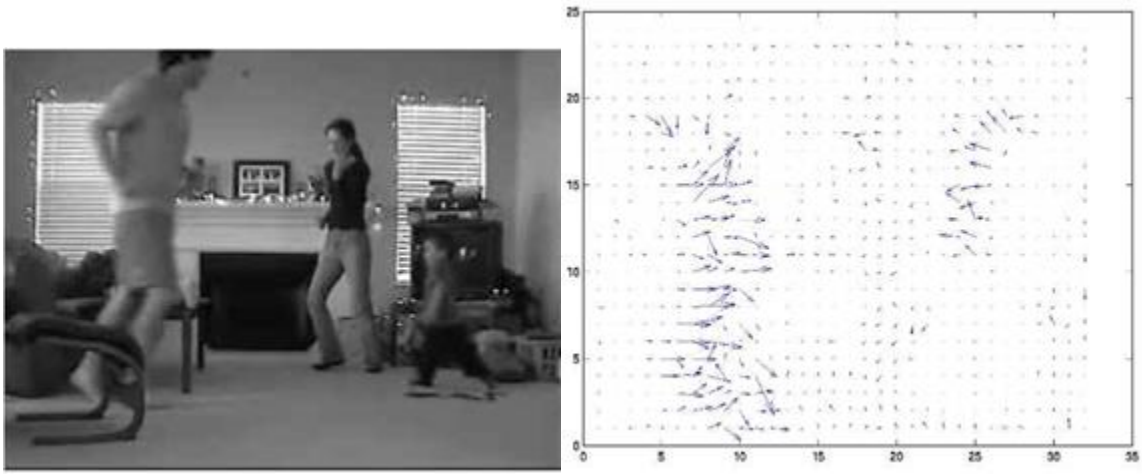


Fig 28. Results of optical flow showing the vectors of moving objects in the scene.[42]

Above we can see an image with people moving in different directions, and the map to the right shows the motion vectors we would get from our calculations.

Pros and Cons of the Optical Flow Method

What is great about the optical flow method is that since you are finding the velocity vectors for moving objects in the scene, you are not only getting data on the direction of movement, but also the speed at which objects are moving. This method also works well when objects are moving relatively slowly, and since we will need to make sure we are not making quick and exaggerated movements with the arm, we can expect that this will be the case.

There are downsides to the optical flow method though that could pertain to this project. The first problem with this method is that it does not work well when combining many points and there is a set that moves differently from another. For instance, when we have motion that is fluid, such as a flag blowing in the wind, or chaotic motion, such as flies moving around a room, then we have many vectors moving at different velocities. Neither of these are an issue for us, but another instance that falls in this category is when you have rotation, a scenario that could come up depending on how the pieces the arm is holding need to be placed.

Another scenario where this method falls apart that could be a major issue of us is when lighting is inconsistent or low. Having low or inconsistent lighting conditions causes the ramp for our lightness intensity to be shallower, meaning there will seem to be a drop in our brightness value creating a perceived sense of motion that is not actually there. When the rover is being used for educational demonstrations, this will not really be much of an issue, but should this design ever be used for lunar missions, this could be a concern should the rover be performing actions during lunar nights. Currently our sponsor does not believe the rover should be expected to be active during these times; however, we could possibly fix this issue without limiting any capabilities. If we placed a light on the front of the rover or on the arm, we could allow for the vision software to still work with minimal errors once it gets dark. This would require more hardware though, so we will need to discuss this further with our sponsor and designers of the rover or arm to see if the benefits of this method would outweigh any costs of adding a light, but this may cause us to go with another method.

The last few issues with this method are when surfaces of objects in the scene get wet, they appear dark (similar to the brightness issue); objects in the scene are moving too fast, causing motion blur; there is a flat/uniform intensity profile and there are no edges, so there is motion that is occurring that can’t be detected, such as a smooth ball being spun, and when combined points only have one edge direction, so there are not enough independent

equations to find a solution. These issues are worth understanding, but most likely will not be ones that we need to be concerned about. Overall, optical flow is a relatively sophisticated method that will be worth testing and potentially using for our motion detection, but we will need to carefully consider some of the problems that arise from the method, and we will also need to do some benchmark tests against other approaches, as I anticipate this method being more computationally intensive than other methods.

Background Subtraction

As the name suggests, background subtraction is a method that takes static areas of an image and classifies this area as a background model, which is removed (or subtracted) from the scene. A foreground mask is then calculated and placed over objects that are moving or have specific features such as edges [44]. This is done in three main steps [45]. First, we estimate the background model at time t. Then we subtract the background model from the image. Finally, we apply a threshold, Th, to the absolute difference to get the foreground mask [45].

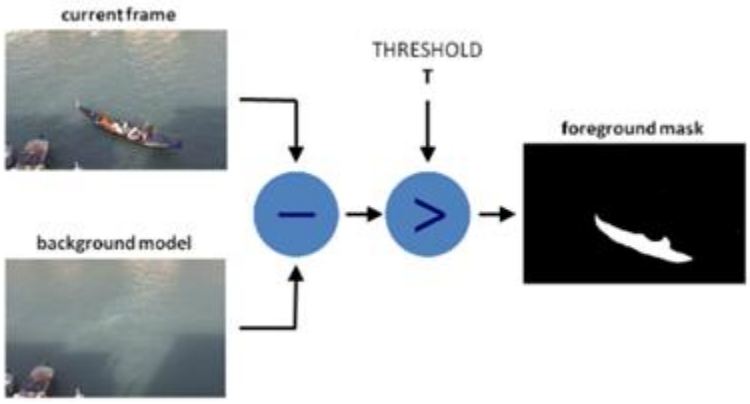


Fig 29. Visualization of the steps for background subtraction. [44]

We can accomplish this by using two functions. One for the whole image: $I(x, y, t)$, and one for the background: $B(x, y, t)$. The background can be estimated by using the previous frame, so our background function becomes: $B(x, y, t) = I(x, y, t - 1)$. Now we can take the absolute difference: $|I(x, y, t) - I(x, y, t - 1)|$, and check if the result is greater than the threshold: $|I(x, y, t) - I(x, y, t - 1)| > Th$. This approach is known as frame differencing. While the simplicity of this approach is great, results are often poor due to its dependence on many variables such as speed of objects, frame rate and the global threshold. Below are images of results using frame differencing at different thresholds.

Fig 30. Example input image. [45]

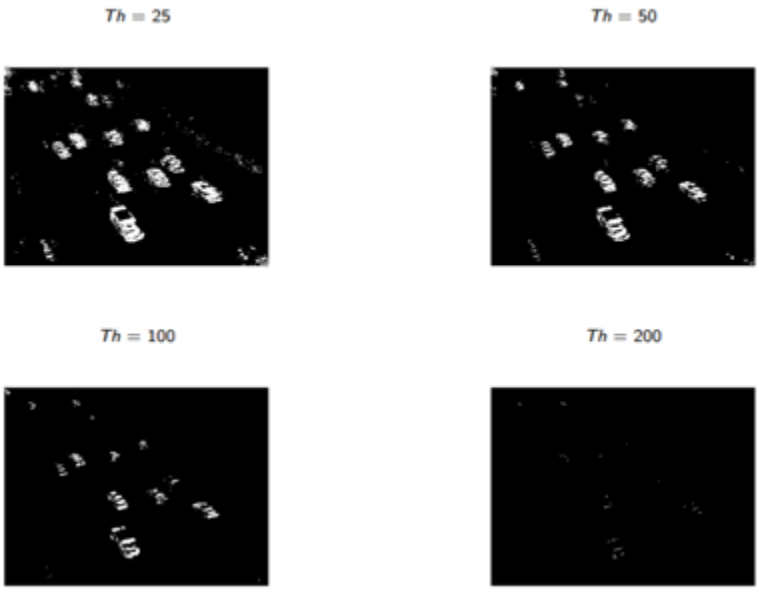


Fig 31. Results from standard approach. [45]

We can slightly improve these results by using either the mean or median filter approach. The mean filter approach treats the mean of the previous n frames as the background. This now makes our background function: $B(x,y,t) = \frac{1}{n}\sum_{i=0}^{n-1} \quad I(x,y,t-i)$, and our full equation is: $\left|I(x,y,t) - \frac{1}{n}\sum_{i=0}^{n-1} \quad I(x,y,t-i)\right| > Th$. Here are results with $n = 10, n = 20$ and $n = 50$ respectively.

Fig 32. Results from mean approach. [45]

Similarly, we use the median filter approach, which takes the median of the previous n frames to create the background model. Now, $B(x, y, t) = median\{x, y, t - i\}$, and $|I(x, y, t) - median\{x, y, t - i\}| > Th$ where $i \in \{0, \dots, n - 1\}$. The results using the same values for n as we previously used will look like this:

Fig 33. Results from median approach. [45]

The results from the mean and median filters do appear to be better than the frame differencing, but the mean and median filters have virtually the same results. All three of these methods will fall apart given these scenarios: repetitive motion from clutter in the scene, change in lighting, objects moving too slowly, shadows, overlapping visual fields and objects being introduced and removed from the scene. These issues are due to the fact that our threshold is not a function of time and is therefore never updated. Luckily, we have another approach that deals with these issues.

What we can do instead of taking the pixel values as one distribution, we can take these values as a mixture of Gaussian distributions [46]. Having a mixture will allow us to deal with several surfaces in one pixel, and the change in lighting conditions. Taking the values of a particular pixel over time which we call the "pixel process", which are scalars for gray values and vectors for color images, we can use that fact that at any moment in time we know a pixel's history, equated as: $\{X_1, \dots, X_t\} = \{I(x_0, y_0, i) : 1 \leq i \leq t\}$. Using (R, G)

scatter plots, can see the pixel process values in two image sequences two minutes apart, values from specularities on the surface of water, and values from a flickering monitor screen.

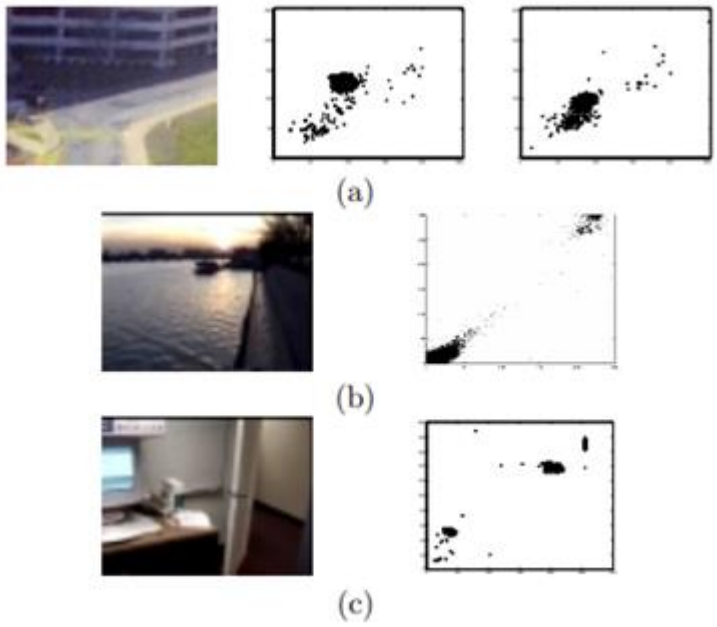


Fig 34. Scatter plots of different surfaces.[46]

These illustrate why there is a need for adaptive automatic thresholds, and a multi-modal representation. For our update procedure, we can use the pixel history modeled as a mixture of K-Gaussian distributions. The probability of observing the current pixel is given as:

$$P(X_t) = \sum_{i=1}^{K} \omega_{i,t} * \eta(X_t, \mu_{i,t}, \Sigma_{i,t}) \tag{13}$$

K represents the number of distributions, $\omega_{i,t}$ is an estimation of the weight of the i^{th} Gaussian at time t, $\mu_{i,t}$ is the mean value of the i^{th} Gaussian at time t, $\Sigma_{i=1}^{K}$ is the covariance matrix of the i^{th} Gaussian at time t, and η is the Gaussian probability density function. The η function can be understood as:

$$\eta(X_t, \mu, \Sigma) = \frac{1}{(2\pi)^{\frac{n}{2}} |\Sigma|^{\frac{1}{2}}} e^{-\frac{1}{2}(X_t - \mu_t)^T \Sigma^{-1} (X_t - \mu_t)} \tag{14}$$

To update our Guassians, we use an on-line K-means approximation, as it would be too costly to implement an exact algorithm on a window of data since each pixel has a mixture model. We check each pixel, X_t, against the existing Gaussian distributions with a standard deviation of 2.5 till we find a match. When a match is found, the parameters are updated as follows :

$$\mu_t = (1-\rho)\mu_{t-1} + \rho X_t$$
$$\sigma_t^2 = (1-\rho)\sigma_{t-1}^2 + \rho(X_t - \mu_t)^T(X_t - \mu_t) \quad (15, 16)$$

where $\rho = \alpha\eta(X_t|\mu_k, \sigma_k)$, and α is the learning rate. The prior weights, $\omega_{k,t}$, are updated as $\omega_{k,t} = (1-\alpha)\omega_{k,t-1} + \alpha(M_{k,t})$. $M_{k,t}$ is 1 when there is a match, and 0 where a match cannot be found. In the case where no match is found, the least probable distribution is replaced with the current values mean value. Now we have enabled ourselves to order our distributions that have the most supporting evidence, and least variance, which is the value ω/σ. Doing so will cause static persistent objects to accumulate with high supporting evidence and low variance, whereas a new object that occludes the background object will not match with one of the distributions, either creating a new one or increasing the variance of existing ones. The B distributions can then be chosen for the background model as: $B = argmin_b(\sum_{k=1}^{b} \quad w_k > T)$, with T being the minimum portion of data should be accounted for by the background. A smaller T means that we most likely have a unimodal distribution, so we can save processing power by only using the most probable distribution, and when T is a higher number, we most likely have a multi-modal distribution, allowing for two or more separate colors to be accepted into the background.

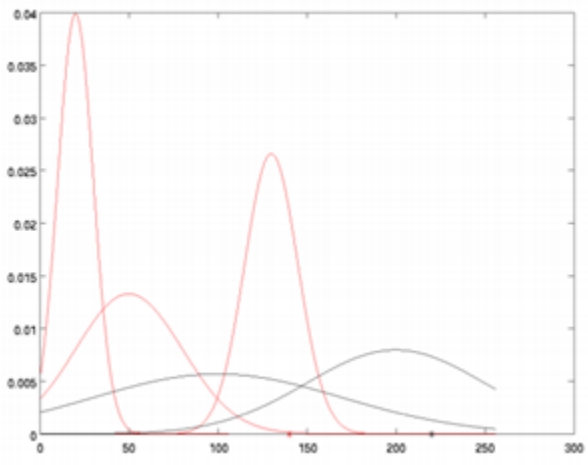


Fig 35. Plot showing the different distributions. [46]

The plot above has red distributions that will become the background model, and black distributions that become the foreground.

Pros and Cons of Background Subtraction

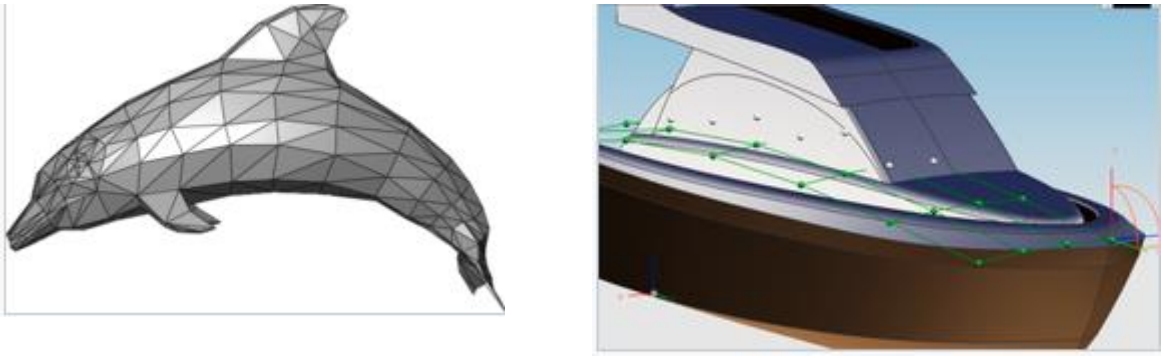
The benefits of background subtraction are mainly found in its simplicity and efficiency. Of course, this does depend on which approach we are using for our background subtraction, but both approaches mentioned above have advantages over the optical flow method. The frame differencing approach is significantly simpler and more efficient than the optical flow method, but poor results will most likely cause us not to go with this approach. Using either the mean or median approaches make our computations only slightly more complicated, while still maintaining a relatively efficient model compared to optical flow, and the results are much better than the frame differencing approach. It should be noted that these two approaches do have high memory requirements, but this can be improved for the mean filter approach by keeping track of a running average for the background, so the equation for the background can become: $B(x, y, t) = (1 - \alpha)B(x, y, t - 1) + \alpha I(x, y, t)$ with α as the learning rate.

The first three approaches, while simple, work poorly on scenes with many changes such as lighting or new objects coming in and out of the scene. Using the Gaussian distribution approach fixes this issue, but it is more complicated than the first three approaches, so some benchmarking may have to be done to see the performance compared to the optical flow method; however, there are many steps included to reduce performance problems, and the fact that there is no matrix inversion is a major reason why I believe it will most likely perform better than optical flow. In the end, we may not even need to use the Gaussian approach, as most of the problems that it fixes are ones that we should not expect for our project. Although we mentioned the issue with lighting during the optical flow section, this is mostly involving low lighting scenarios, not frequent changing lighting conditions. The hope is that this means we can go with the simpler approach that saves computational complexity, and we will just have to focus on optimizing for space complexity.

The final issue with this method, which is true for all approaches, is that the camera is required to remain static at all times. This means that we need to keep the rover still as the algorithm is working, and while I doubt it will be necessary for the rover to be doing too much movement while doing work with the arm, there could be some slight instability caused by the arm moving no matter how much we try to limit this. This may just mean we need to periodically re-calibrate, but it's also possible that this could cause too many errors to allow us to use this method for our implementation.

Point Cloud Visualization

A point visual is a three dimensional rendering of an image using different 3 dimensional cameras in order to map out an environment. Our group believes that implementing this into our virtual environment will help tremendously in our beginning stages of testing. There are a couple of models that are used for point cloud renderings. These different models all do the same thing, but there are differences in speed and accuracy of the models for each one: Triangle mesh constructs a 3D image using nothing but different sizes of triangles, Polygon mesh constructs a 3D rendering using multiple different types of polygons for a more detailed picture, and Non-Uniform rational B-spline (NURBS) which uses curved planes for a more solid rendering [47].



Triangle/ Polygon Mesh

NURBS

Fig 36. Showing the difference between Triangle Mesh vs. NURBS Visualization [48, 49]

As you can tell from the images above, there are slightly different ways to achieve a 3D rendering of an object and different algorithms and data structures that are needed for each one.

Triangle Mesh Method

The triangle mesh method uses a combination of vertices, edges, and maps to create each image. An easy way to combine all of the triangles needed for an image is to create a triangle strip [48]. With a triangle strip, each triangle must share an edge with a neighboring triangle meaning there are no triangles off by themselves. If the image is more circular, then a triangular fan is used instead [48]. This connects a vertex from each neighboring triangle at a center point and fans them out. Combining these two methods will result in a very accurate 3D rendering.

There are multiple python libraries that can implement triangle meshes, the most used and mist accurate is Open3d. In combination with the NumPy library, you load in an image and create a data path to export it into. Then you have a choice of which meshing method you would like to use, in this case it is the Triangle mesh. Then after the algorithm has run its course, you may need to downsize or upsize your model to fit the specific dimensions [50]. This can all be done with just a few lines of built in functions in python using Open3D and NumPy.

Non-Uniform rational B-spline Method

The Non-Uniform rational B-spline or NURBS method uses all different sorts of shapes and curves to render a 3D image. There are many different curve, surface, and Volume factories that this algorithm to choose from when the algorithm is running [51]. There is a hierarchy used when creating an object that is rendered. The whole B-spline controls the Spline object, and the object controls its surface, curves, and volume.

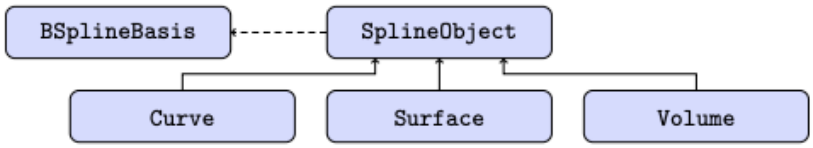


Fig 37. B-spline Algorithm Hierarchy [51]

Which Method We Chose and Why

After further research, we are deciding to model our point cloud visualizations with the triangle mesh method. It is far easier to implement and can be achieved with minimal code due to the built in functions using the Open3D and NumPy libraries. Going forward into our next semesters implementation stage, we will begin to map out our environment using these methods to make it easier to create the simulation for the robotic arm.

Point Cloud Visual Implementation

Now that we have done our research on point cloud visualization, it is now time to implement it. Implementing a regular Point Cloud Visual in python is fairly straight-forward, it just requires importing some libraries and setting up a conda environment. The libraries you have to install are NumPy, LasPy, Open3DConda environment (see library definitions), which can be installed with a simple Pip call into your environment. We also need to download Conda for the environment.

The next step is to load in the environment and point cloud datasets so that it knows what to generate. As Flourent Poux states [52], there are multiple different websites and dependencies that can load in the environment, but since the environment will be in real time, we will let the Intel RealSense Camera capture the environment. This will be our Point Cloud data set.

We then choose a sampling method that divides 3D space into equally sized boxes and squares to segment the environment into cells [52]. Then you find the representation of each cell. There can be zero points or there can be many points in each voxel, but you need to run an algorithm that determines which point will represent that specific voxel by looping through each point and determining which one has the smallest euclidean distance from the barycenter of the voxel.

After you find a representative for each voxel, it is time to visualize your results by using the MatPlotLib library (see library definition). This simply plots the representatives of each voxel in 3D space, showing the terrain of the environment in more detail. Then after all of this we will finally see our point cloud visualization of the environment so we can use it for further testing.

Useful ROS Tools for Vision and Motion Planning

Rviz

Rviz is a robotic visualization tool built for ROS that creates a 3D visualization of sensor data, the robot model and other 3D data that is combined into one view. This makes it much easier to get an idea of what the robot is seeing and how it is detecting objects in its environment. Since rviz is a ROS specific tool, it can be integrated with other ROS tools or plugins such as Gazebo and MoveIt. This means that rviz can be used in both simulation testing and real-world testing, so you will not have to wait until testing the physical cameras/sensors to get data on how well the robot is viewing its environment.

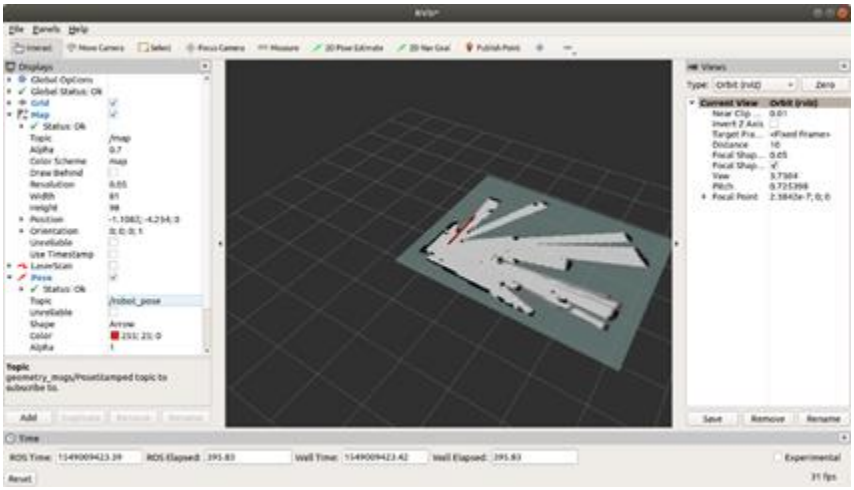


Fig 38. Image of what the Rviz will display in the GUI. [53]

MoveIt

MoveIt is a motion planning library for robots using ROS [54]. With MoveIt, you can either use premade algorithms or implement your own algorithm for motion planning. This gives us the flexibility to use whatever approach will best fit our design. MoveIt can also be integrated with rviz to visualize the data and the plans can be executed through the ROS control system. MoveIt can also be combined with Gazebo for the simulation phase.

Creating motion plans, setting joint or pose goals, moving the robot, adding/detaching objects and adding objects into the environment is made easier through two interfaces that come with MoveIt. These are a MoveGroupInterface class for C++ [55], and a Python based Move Group Interface [56], both of which essentially accomplish the same goal of making the previously mentioned tasks easier for the user to accomplish. Our team may

find more use out of the python interface since we will be working with openCV, but we have the option to use C++ if we find it to be more useful.

Finally, MoveIt also provides features for benchmarking and testing as well. Using the benchmarking package, you can benchmark your motion planning algorithms to ensure you are using the most efficient algorithm for your robot [57]. MoveIt also provides integration and unit testing capabilities through its python-based integration testing scripts, and unit test files which uses the Google Test framework [58]. Google Test provides a TEST function that takes the test case along with the name of the test, and you can also check if the if your code is working as intended by using the ASSERT function which stops if the test fails and the EXPECT function which continues regardless but will still show failure of the test. All the features I have mentioned work with Melodic, so there are no compatibility issues with the version of ROS we will be using.

ROS Industrial

ROS Industrial is an open-source project that uses ROS to provide interfaces for working with grippers, manipulators, sensors, and device networks, as well as including libraries for path and motion planning, sensor calibration and testing [59]. Like the name implies, ROS industrial is designed to be used for the industrial sector, and it is also commonly used for robotics research. Since it is open source, ROS Industrial provides industry grade path planning and grasp planning for a reduced cost. Adding to its own features, ROS Industrial also uses features from MoveIt, so many of the tools that you would find useful from MoveIt can be used with ROS Industrial as well. ROS Industrial is divided into a general package and vendor packages which contain drivers for various vendor platforms such as Fanuc, Motoman and ABB [60]. The tools that ROS Industrial provides may be worth looking into, but since it is mainly used for industrial solutions, it may provide more than we really need. In our final implementation, especially when moving to implementing our software on the physical boards, MoveIt may be all that we need, which can reduce the number of packages that will have to be stored and running on the board.

Neural Networks

Neural networks are a machine learning method that try to achieve artificial intelligence through a similar manner to how the human brain works. The network is made up of a multitude of nodes, sometimes ranging in the thousands that are interconnected and have weights associated with them. Through a series of complex math, data gets passed through each of the connections of nodes, eventually leading to an output layer. This feeding of data can simply be a “feed-forward” method where data is only passed in one direction,

but some models have a forward and backward propagation phases that is usually used to minimize error by calculating a loss and using that to update weights in the previous layers, then repeating the process until loss is minimized as much as possible.

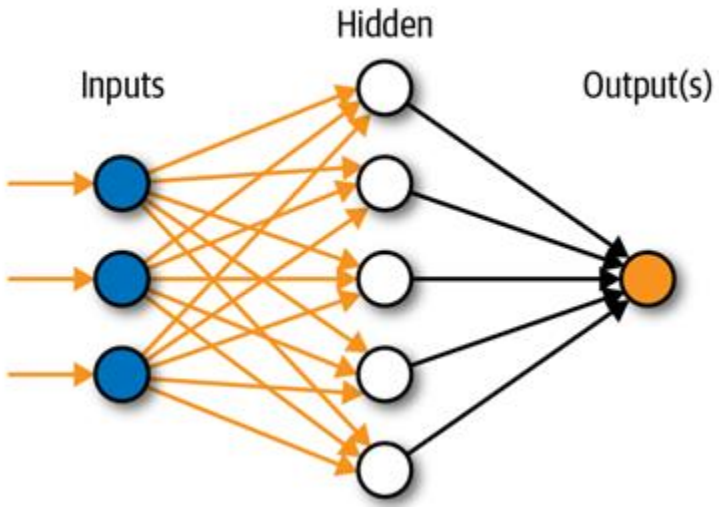


Fig 39. Image of a simple neural network. [61]

Although neural networks have mainly gained popularity in recent years, the concept of a neural network has been around for around 70 years, although the specific ways of achieving these neural networks and how useful they have been has changed significantly over time. The first neural network was proposed in 1944 by Warren McCullough and Walter Pitts who were researchers at the University of Chicago [62]. The proposed neural nets resembled less of what today's neural nets look like and they were not trainable. The purpose of their research was more of a way to show how the human brain could be thought of similarly to a computing device. The first trainable neural network came out of Cornell University in 1957 by Frank Rosenblatt, referred to as the Perceptron. The Perceptron was a much closer representation of what modern day neural networks look like, with the main difference being that the Perceptron only had one layer in-between the input and output layers. Despite the fact that this design was quite ahead of its time, the Perceptron was met with a lot of criticism, and ultimately it didn’t really take off. This mostly was due to the fact that it had only been a few years prior that people were still using analog computers, so the concept of using a programming language was not widely accepted yet, but the biggest reason was that computing power at the time was far too limited to use the Perceptron without taking far too long once you scaled up the problem you were solving for. With advances in GPU’s (Graphics Processing Unit), computers are finally able to use neural networks in much more reasonable amounts of time, which is why we have seen

them become so popular. Today's neural networks not only have hundreds or thousands of interconnected nodes, but there can also be many hidden layers in-between the input and output layers, rather than having the single one like before, improving the capabilities of the training. The inclusion of many more hidden layers has formed another sub-field of machine learning called deep learning, referring to depth that the neural network goes in its hidden layers.

Simple Network Explanation

To fully understand how a neural network works, however, it helps to see what is happening at each step. As mentioned previously, the basic concept is that each neuron takes in multiple inputs with a weight that is randomly determined and applied to each, and a bias is included to account for noise [63]. A weighted sum is computed as $z = w^T x + b$, where w is the weight of the input, x is the input, and b is the bias. Before the result is passed to the next layer or the output layer, the result z is usually put through an activation function to find non-linear patterns.

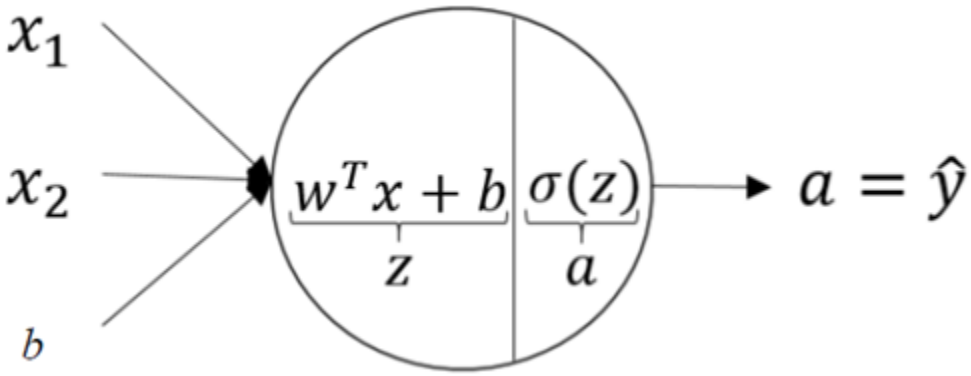


Fig 40. Visualization of a neuron in a network. [63]

There are a variety of activation functions, each with their own pros and cons. The one used in the image above is a sigmoid function, which is the most commonly used activation function. The sigmoid function is great for classifying, as the large changes in x result in small outputs for y and vice-versa [64]. The problem with the sigmoid function is that changes are very small at the tail ends of the function, and the derivative values in these regions converge to zero, which can result in slow learning, also known as the vanishing gradient problem. Another function is the tanh function, which is similar in shape to the sigmoid function, but its derivative is much steeper so it can have more efficient learning compared to the sigmoid function. It still has the issue of the tail ends converging to zero, so it also suffers from the vanishing gradient problem. Two functions that do solve this issue are the ReLU and Leaky ReLU functions. ReLU finds the max between zero and the

input z, so in other words, any negative input will result in an output of zero. This not only solves our issue with the vanishing gradient, but it also significantly increases performance, as there is less computational load compared to the two previous functions. The issue is ReLU, is that because we will not take negative values, there is no learning happening in regions below zero, which is what Leaky ReLU accounts for. Leaky ReLU is just like the ReLU function, except instead of finding the max between zero and z, it finds the max between $0.01z$ and z. This means that you can still have learning in the negative regions, while still maintaining speed since it is still close to zero. There exist more activation functions than the ones discussed here, each with there own positive and negatives, and ultimately the activation you use depends mainly what you are training for.

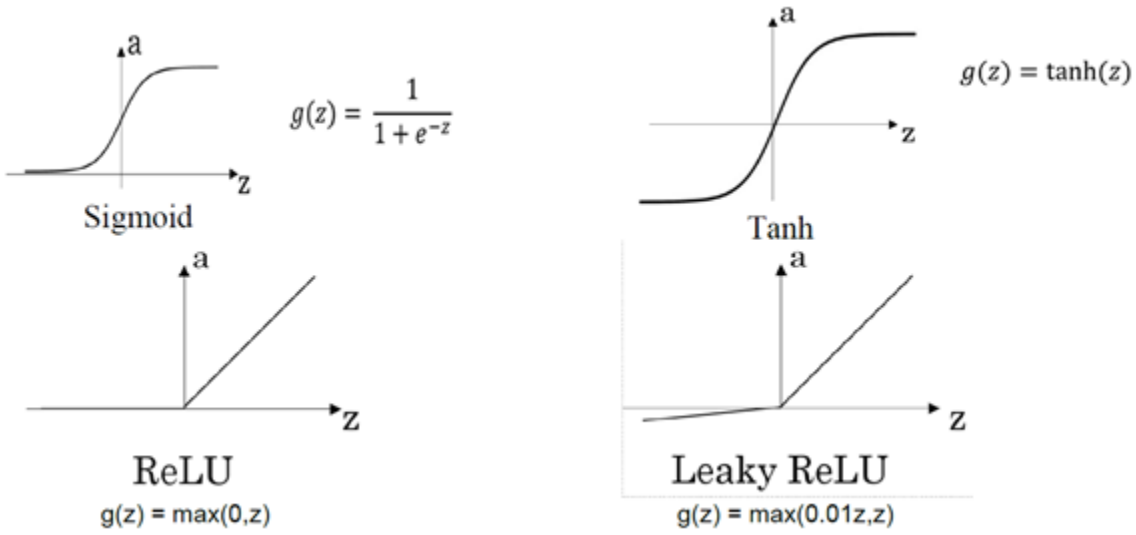


Fig 41. Plotted functions for the Sigmoid, Tanh, ReLU and Leaky ReLU functions [63]

What we have discussed so far is what is called the forward pass. Some models will only use a forward pass, but some use a method called backwards propagation that uses a loss function to calculate how close that pass came to a target output, using that to go backwards through the network, updating the weights along the way. Much like the activation functions, a loss function is chosen based on what type of problem we have. There are two groups you usually can place a loss function under, classification functions and regression functions. Classification functions are good for finding labels, like whether a part of an image is a face or not. Loss functions for these types of problems are cross entropy (log loss) which is good for binary classifications, and multi-class cross entropy which is good for more than two classes that we want to classify. Regression functions are good for finding quantities rather than labels. An example of a regression function often used is mean square error. Regardless of the loss function used, a process known as gradient descent takes place where the models' parameters are optimized until a local minimum to

the loss function is reached. These gradients are what is backpropagated through the network, with the new weights determined as: $W^{k+1} = (W^k - learning\_rate * (gradient))$.

Let us use this network with inputs, weights and an output as shown in this image:

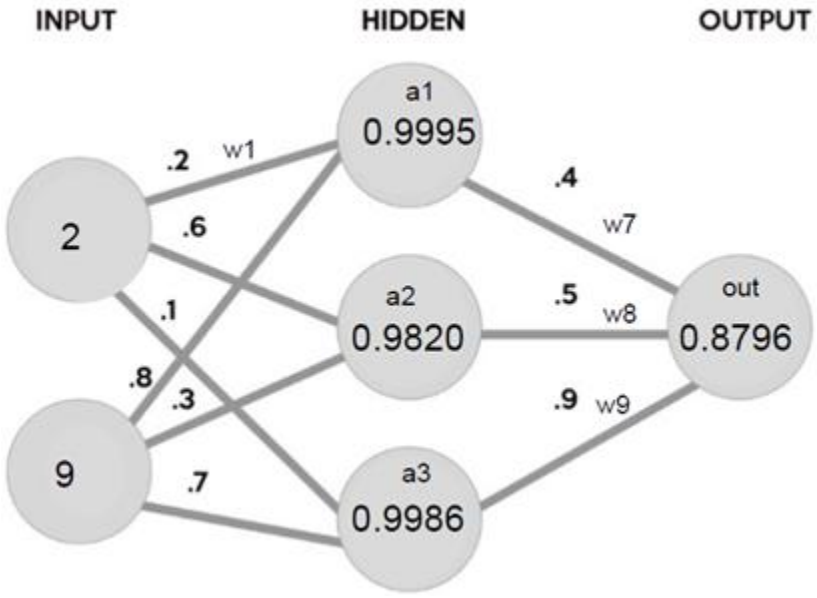


Fig 42. Example network before backpropagation. [63]

For the gradients, we will need to find the derivatives for each of the weights using a chain rule. Using the topmost connection of neurons, we first find $\frac{\partial E}{\partial w7} = \frac{\partial E}{\partial out} * \frac{\partial out}{\partial z} * \frac{\partial z}{\partial 7}$, with E representing the output after the loss function. In this case, we will use mean square error: $\sum \frac{1}{2}(target - output)^2$. We can find each of the partials separately to make our calculations easier. Finding our first partial, $\frac{\partial E}{out} = \frac{1}{2} * 2(target - output)^{2-1}(-1) + 0 = -(target - output)$, and if our target is 1 and output is 0.8796, then our result is $-(1 - 0.8796) = -0.1204$. Next, we for $\partial out/\partial z$, we must take the derivative of our activation function, in this case we will use the sigmoid. The derivative of the sigmoid function works out to $\sigma(1 - \sigma)$, so $\frac{\partial out}{\partial z} = 0.8796(1 - 0.8796) = 0.1509$. Finally, we must find $\frac{\partial z}{\partial w7} = \frac{\partial(a1*w7+a2*w8+a3*w9+b2*1)}{\partial w7} = a1 + 0 + 0 = a1 = 0.9995$. Combing our calculations together, we have a result of $\frac{\partial E}{\partial w7} = -0.1204 * 0.1059 * 0.9995 = -0.0127$. This makes up the gradient for our formula to find the new weight, so with a learning rate of 0.1, our new weight becomes $w7 = w7 - learning\_rate * (-0.0127) = 0.4 - 0.1 *$

$(-0.0127) = 0.40127$. For $w1$, we take the same steps, with the chain rule equation coming out to $\frac{\partial E}{\partial w1} = \frac{\partial E}{\partial a1} * \frac{\partial a1}{\partial z} * \frac{\partial z}{\partial w1}$.

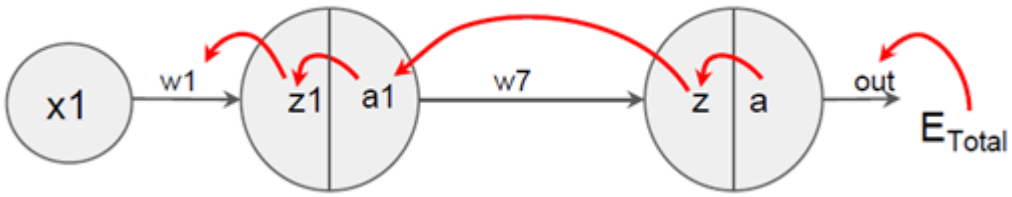


Fig 43. Visualization of the backwards pass for updating w1. [63]

This same backwards pass will occur on the other connections, updating all the weights in the network, and the forward passes and backward passes will continue to happen iteratively until the loss reaches its minimal value. One thing that has still not been answered, however, is how do you determine the learning rate. Unfortunately, finding the right learning rate can be tricky, as you just have to try different values until you find what works best for your model. Having too small or too large of values can cause major issues. Too small of a learning rate, the gradient descent will eventually reach the local minimum, but it will take a long time to do so. Too large of a learning rate can cause even more disastrous results, making the descent take big steps that can bounce back and forth between the convex of the function, completely missing the minimum. For this reason, a good rule of thumb is to start with a lower learning rate, and gradually increase it until you find a rate that reaches the minimum, but also maintains efficiency.

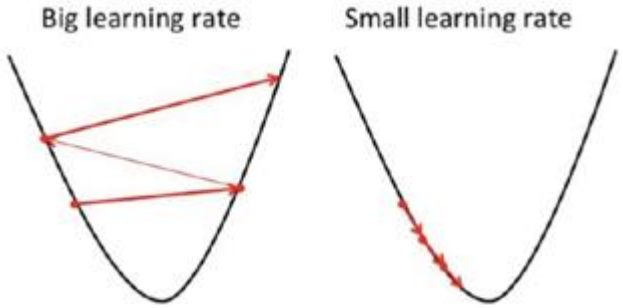


Fig 44. Result of having too small or large of a learning rate. [63]

Convolutional Neural Networks

While the additions to the neural networks we just mentioned do help solve more complex problems, they still are limited in how much they really can learn, so newer models have

been made that add more dimensions to each of the layers, allowing for more complexity in the learning the model is capable of. Convolutional Neural Networks (CNN's) are models that are great for learning features of an image. If we were to use a normal feed forward network, we could take the image and feed it into the input layer as a vector rather than a matrix, but this would only work on simple images, whereas the CNN can capture spatial and temporal dependencies in the image that the standard network could not [65]. A CNN is able to achieve this by using a series of layers that each serve their own purpose in manipulating the image until it can be flattened and put through a network that more closely resembles the one we saw before.

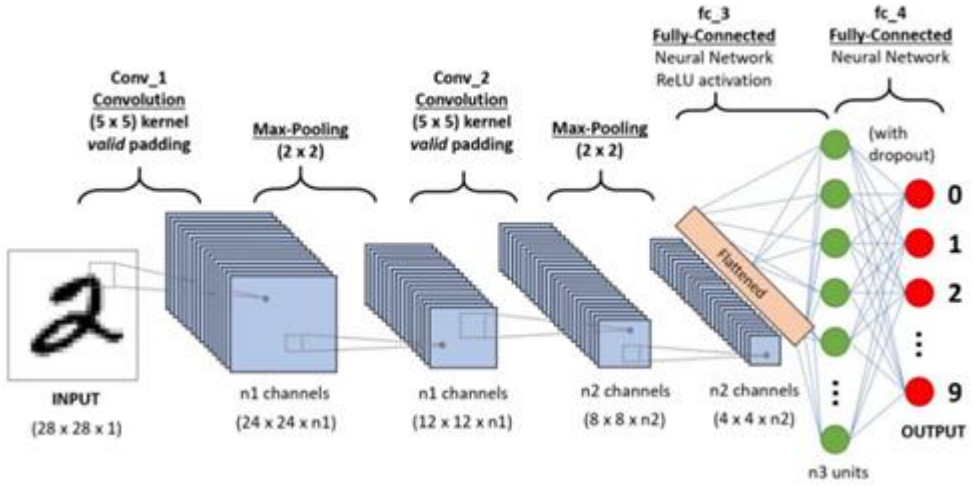


Fig 45. Example of a CNN that learns to classify handwritten numbers in an image [65]

First, we have the convolutional layer. The convolutional layer is great at detecting features in an image such as edges or color gradients. Each added convolutional layer helps in detecting higher level features. How this works is by taking what is called the kernel or filter, which is then applied over the matrices of each dimension in the image (the input volume). The filter is first placed over the top left area of the image matrix, and the image matrix and filter are then multiplied, with the result of each multiplied matrix and filter throughout the input volume then added together to produce the result for one pixel of the output volume [66]. The filter then moves to the right some spaces depending on what is called the stride, and the process repeats, producing the next value for the output volume, which continues until the full image matrix has been covered by the filter. In what way does this help us find features though? Assume we are looking for curves in an image. We could use a filter that represents a curve shape by having pixels in the matrix that have values greater than zero form the shape of a curve, so that all pixels surrounding those have values of zero. When the convolutions occur on the image, a portion that does not contain a curve will result in a low value, since most of the pixels in the image with values higher

than zero will be multiplied with the pixels that have values of zero in the filter matrix. Conversely, when there is a curve in the area the filter is currently over, there will be a larger number of pixels in the image matrix that will be multiplied with the curved shaped pixels in the filter matrix, producing a larger value as a result.

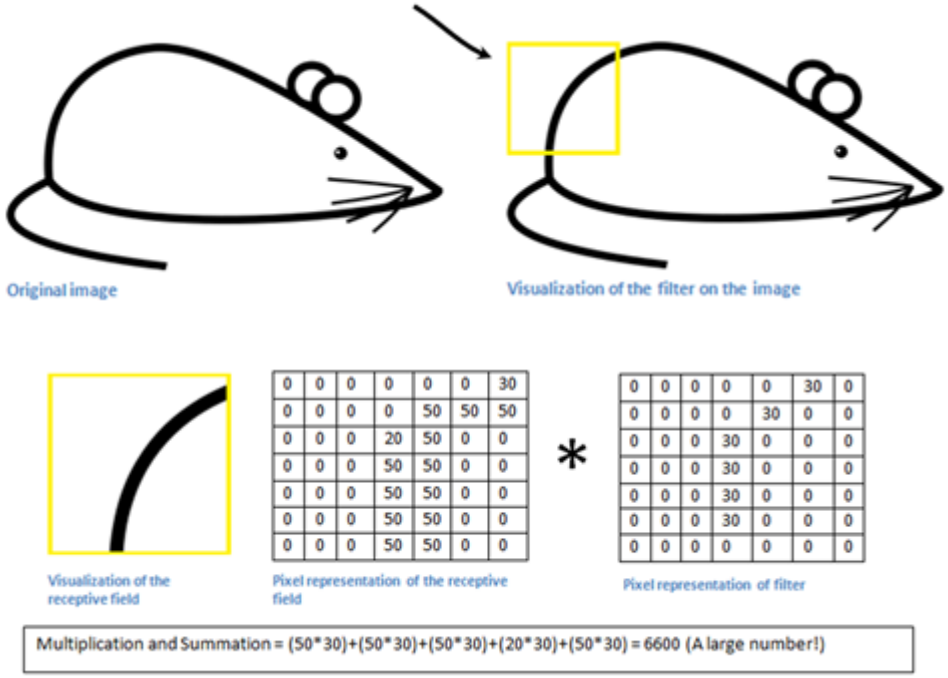


Fig 46. Convolution with a curve filter resulting in a large number in the curved area of the mouse image [66]

The final dimensions of the output volume can also be increased or decreased depending on a value called padding, which means when the padding is higher than zero, the filter will extend past the dimension of the input image. So, when padding is used, we get an output volume that is bigger than the input, since we are doing more convolutions, and when we do not use padding, the output volume will be less than the input volume [65]. An equation to help solve for the dimensions after a convolution is $\frac{W-F+2P}{S}+1$, with W being the width of the image, F being the width of the filter, P being the padding and S being the stride. The height is found the same way, replacing W with H. The third dimension is determined by the number of filters applied, so if the result for the height and width was 28, and the number of filters used was 6, then the dimensions for the output volume would be $28 \times 28 \times 6$. Next, we have the pooling layer, which helps lower the computational load by further reducing the dimensions of the image. It also helps to reduce the likelihood of overfitting, a problem where a model is very accurate for the dataset it is trained on, but inaccurate for any datasets it tests with later. No learning occurs during this

layer, however. The two main types of pooling are max and average pooling. In max pooling, a filter is placed over each area of the image just like the convolutional layer, but instead of doing a convolution, we simply take the max value in that area as the pixel in the next position of our output. Average pooling takes the average rather than the max value. The benefit of max pooling over average pooling is that max pooling also performs de-noising while also reducing dimensional size, so it tends to perform better than average pooling [65].

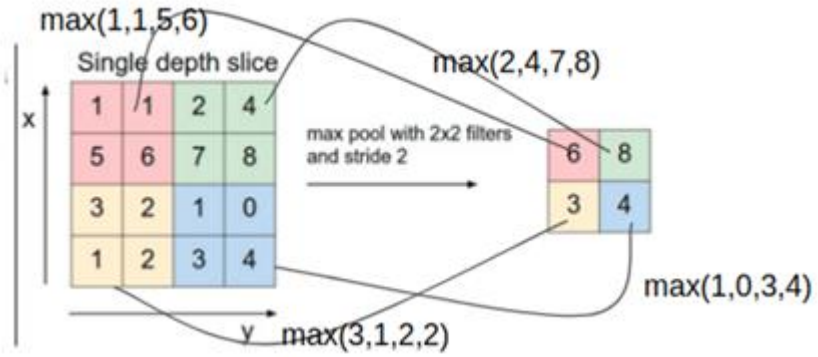


Fig 47. Example of the output after max pooling. [66]

You can also determine what the dimensions after pooling will be similarly to the method in the convolutional layers, using the formula $\frac{W-F}{S}+1$ for the width, and $\frac{H-F}{S}+1$ for the height. The depth will not change, however. The final steps in the CNN are the flattening layer, and the fully connected layers. The flattening layer simply takes the 3-dimensional output volume and "flattens" it down to a single column vector, with a size that can be determined by multiplying height, width, and depth together. This is then fed into the fully connected layer, which creates the interconnection of neurons in layers in the same way the feed forward network works. Each layer, like the networks mentioned before, contain an activation function accounting for non-linear patterns, with the ReLU function typically what is used (although others can be used). Backpropagation with a loss function is included to minimize error and optimize our final result, with the final step being the inclusion of a softmax function, whose job is to place our image in one of the multiple classes that the network is checking for. It does this by finding the probability of each of the classes, where the summation of each of the probabilities must add up to 1. The probabilities are determined by the formula $\sigma(z)_j = e^{z_j} / \sum_{k=0}^{K} \quad e^{z_k}$, where $\sigma(z)_j$ is the probability of the output vector z being class j, and K being the number of classes. The class with the highest probability is then the class chosen as the final output. This is encoded by making a bitstream of 1's and 0's, whose length is the number of classes

there is, and a single 1 is placed in the position representing the class, with the rest of the bits simply being 0. This means that if class 3 was chosen, then the bitstream would contain a 1 at the 4th position.

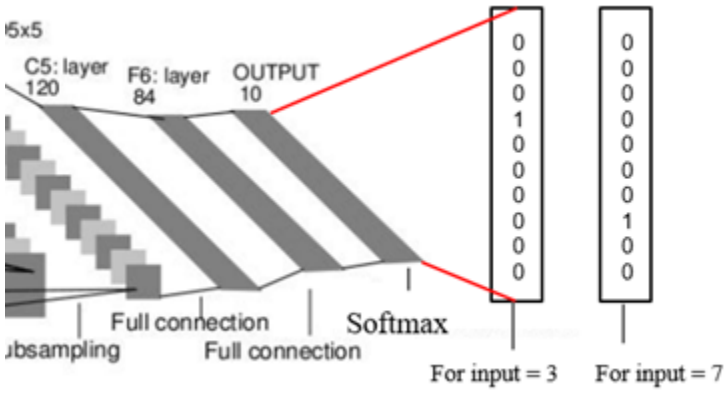


Fig 48. Example of the flattening and fully connected layers with the final bitstream after softmax and encoding. [66]

While this is the basic design that a CNN will follow, there are many different architectures that have been made that use the premise of the CNN with the inclusion of new layers and exclusion of other layers used in a basic CNN. Architectures like AlexNet, GoogLeNet and VGG have continued the trend of increase the depth of the networks through increased number of layers that improves accuracy but also comes at a cost to performance and in some cases, introduces a higher likelihood of vanishing gradients through the increased depth of the network. One way of solving this that has been developed is an architecture called ResNet, which includes residual blocks that allow for layers to be skipped, reducing the effects of vanishing gradients, and then the skipped layers are gradually restored later when the feature space is more properly learned [67].

Pros and Cons of Neural Networks

Neural networks are innovative forms of machine learning that have achieved incredible results when compared to older methods. Neural networks can account for vast amounts of features especially when using an architecture like a CNN. This makes them great for solving problems that require classifying a data into a group or searching for specific objects or details in an image. Neural networks can handle a level of complexity that many traditional models tend to have trouble with. They also are a big trend right now, so there is a lot of research being done finding new ways they can be used to solve

problems. Models that take stereo images, finding redundancies in the image pair to limit bitrate when encoding while maintaining quality when decoding, to models that 2D images and learn to reconstruct objects in the image into 3D all using neural networks have been developed and researched in just the past few years [68, 69]. There are also libraries such as keras and tensorflow that allow developers to build the complex networks without having to spend days or weeks building every aspect of the model. Instead, these libraries allow the developer to simply specify the layers they wish to include for their model, with the parameters they need in the case of keras, or, in the case of tensorflow, there is a happy medium where the developer can have more control over some of the finer details, while still having functions that keep them from having to build everything from scratch.

Despite all of these positives, neural networks of course aren't perfect. One of the biggest issues they face is they almost always are computationally intensive. Even the most efficient and clever models can take up to weeks to train bigger datasets, and the only hope of making them anywhere close to efficient is by using a powerful GPU or TPU (Tensor Processing Unit), a piece of hardware developed by Google that is specifically built for dealing with neural networks. This means, training on a neural network is not cheap, and you need to be willing to deal with the long training times if you have a complex problem or big dataset. Besides just the issue with computation, neural networks have what is called the "black box" problem. The black box problem refers to the fact that when you pass data into the network, it spits out an answer, but you have very little idea of how it arrived at that answer. Like many of the other issues with neural networks, this problem only expands as the complexity of the neural network increases. One problem this creates is in situations where a human may be affected by the decision the network makes. If it is not known how the network came to that conclusion, then the person may not be happy to find out that there is no explanation for the decision (like being banned from their favorite social media website). This also creates a scenario where understanding how the network arrived at its conclusion could be useful so the developer can tweak some of the parameters or modify the model, better optimizing it for a more accurate result, but more often than not, you have to resort to trial and error to find how to make the model work best. This leaves some people to be a bit skeptical whether neural networks are really anything more than hype, and if we are better off sticking with the older forms of machine learning already developed.

Project Use Case

Whether or not neural networks end up being the definitive form of machine learning, they definitely have their applications, and they are worth testing for use in our project. In

our case, a neural network could be useful for classifying the different pavers or landing pads the arm is going to be picking up. A neural network that is trained to take an image of one of the pavers and determine which type of piece it is could then feed that data to our autonomous motion network to determine how the arm needs to move so that the paver can be placed in the correct position. Since we have the summer free to work on the project, this should give us plenty of time to develop a network that fits our needs and to do the necessary training to get the best accuracy. We will need to gather what the potential paver shapes could look like so we can set each as its own individual class to be decided when the image goes through the network. We do not anticipate needing too complex of a network, since we should not need to be searching for too many features. Mostly what we will need to be searching for is the edges around the paver which determines which shaped piece it is, so a basic CNN model should suffice for our purposes. Training most likely will be done locally on one of our own systems, and Nvidia nano should definitely have the power needed to run an already trained model. Like the rest of our visual network, however, testing will determine what we ultimately go with. A neural network could give us a powerful tool that could accomplish a lot of what we're looking for, but it could also end up being complete overkill, causing us to go with a much simpler approach.

Arm Path Planning using Stereo Vision

When it comes to path planning for an arm using stereo vision there are numerous methods for accurately tracking and moving the arm to the desired location but almost every method has aspects that overlap with each other. These aspects are the key to stereo vision path planning for arms and are built upon for almost every solution.

Object detection is the first key step, following the methods of most path planning algorithms. Using a stereo camera, two images are taken and scanned for Regions of Interest (ROIs) that denote detected objects using object detection algorithms like sobel, canny, etc. These images then must be processed and aligned using some form of ROI alignment to ensure each pixel in one image is properly paired with the corresponding pixel in the other image.

Following Object detection there is creating a coordinate system that accurately denotes the location of the object and arm. Using a coordinate system, the arm controller can accurately create parameters to ensure the arm moves to the needed location. The creation of this coordinate system generally starts with the x and y axes being centered on the same plane as the stereo camera and having the z axis extend out in front of it. This method

allows for the coordinate system to easily be defined and all calculations made about distances are easy to denote. Next is the process of locating the object and denoting its center. The most common method for this is the use of triangulation with stereo cameras as shown in Figure 50 below. Figure 49. is a visualization of a stereo camera and how it sees, this includes denoting each image and how a position relates on both images. As seen, there is a difference in the two images on the location of the object; this is called the disparity or difference between the same point on two separate images from a stereo camera. The disparity is used in most position calculations. Also shown is b or the baseline of the camera which shows the distance between the centers of each camera.

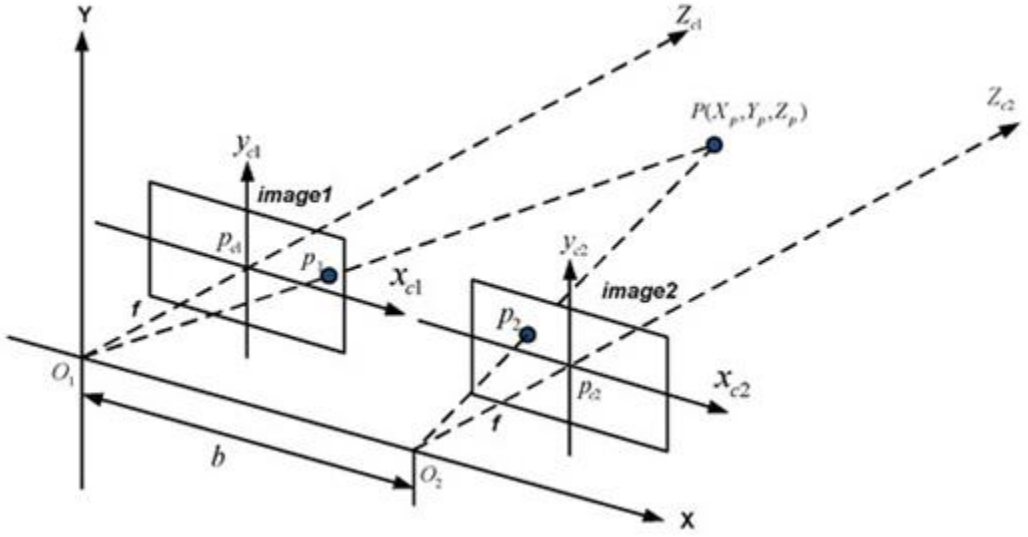


Fig 49. visualization of what stereo cameras see. [70]

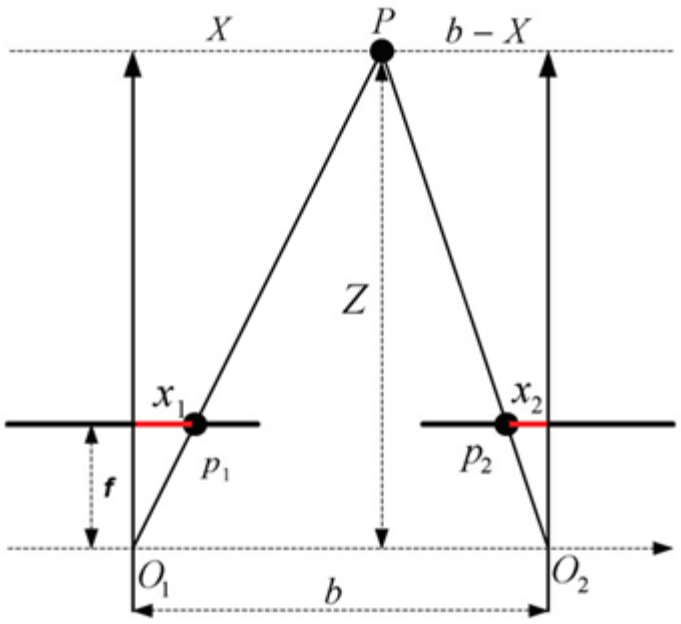


Fig 50. Triangulation method for stereo vision. [70]

As shown by the figures above, each coordinate for the center of the detected object can be found by using the variables denoted above with the functions below where f is the focal length of the camera, d is the parallax ($x_1 + x_2$), and b is the baseline or the distance between the optical centers of the two cameras:

$$Z = b * f/d \tag{17}$$

First the depth must be measured since this value is used in both the X and Y coordinate calculations. For this reason, it is important that the depth value be calculated with the least amount of error possible since any error will lead to compounding error when finding X and Y. It is also important to note that depth estimation has a very high error when it comes to distant objects. This is because the disparity, which is an important aspect in calculating depth, has very minor differences the further away an object is. For example an object that is only 5 meters away will have a very large disparity and will have large changes as the object is moved even another meter away; in contrast an object that is 100 meters away already has a very small disparity and thus when moved further the change in disparity is far less noticeable or even undetectable.

$$X = Z * x_1/f \tag{18}$$

$$Y = Z * y_1/f \tag{19}$$

Next X and Y can be found using the depth value Z. These coordinates denote the x and y positions of the objects center.

But before any of these values can be calculated, the camera must be calibrated to ensure as minimal error as possible. This calibration involves both intrinsic and extrinsic parameters. Starting with the intrinsic, the u and v axes of the image plane must be included with (u_0, v_0) being the coordinates of the principal point (image center); α and β are the scale factors for the axes; and finally γ is the skewness of the camera's images. The extrinsic parameters that also need calibrating are the rotation (R) and translation (t) of the right camera when compared with the left camera [70].

After calibration and coordinate locations have been found, Kinematic equations for the arm can be developed. These equations are built around the idea of using the information gathered from the coordinate system and relating that to the parameters of each joint in the arm. The figure below shows an example of how many parameters a 3 jointed arm has to handle for movements:

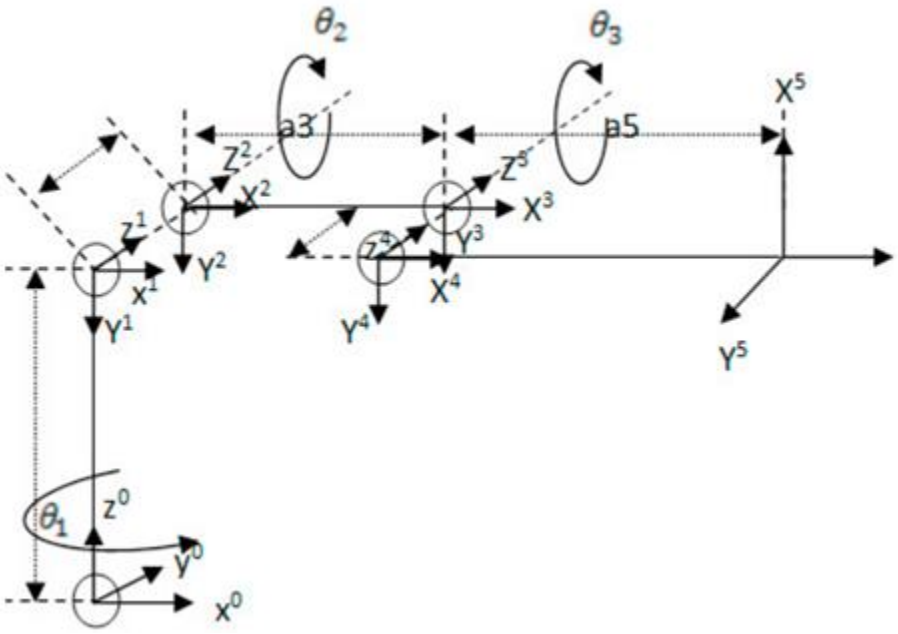


Fig 51. DH model of an arm with 3 degrees of freedom.[71]

A well-used method for creating these kinematic equations using the visual network parameters and parameters of each joint in the arm is to map the joint's position vector (Q) to the position vector of the end-effector of the arm (X) [71]. The end effector of the arm is generally the spot that some form of grabber or claw is located and is the part of the arm

that must reach the specific position of the object. This mapping is done by finding X with a function that takes Q as its parameter. Using this equation along with coordinate transforms, parameters denoting the position and orientation of each joint in the arm can be found.

First the position and orientation of the tool frame in relation to the base frame is found using the consecutive link transformations matrices relating fixed to adjacent links. Using the example arm above, the equation would look similar to:

$$T_{Base}^{Wrist} = T_{base}^{shoulder} * T_{shoulder}^{elbow} * T_{elbow}^{wrist} \tag{20}$$

Where the subscript is where is the starting location and the superscript is the ending location for each of the transformation matrices. Finally, this creates a 4X4 homogeneous transformation matrix which can be used with the tool configuration vector of 3X1 to obtain the coordinates of the tip in relation to the base coordinates [71]. Using the example arm above, this equation will have the form below where θ is the joint angle, d is the joint distance, α is the link twist angle, and a is the link length:

$$\begin{matrix} x_0^5 \\ y_0^5 \\ z_0^5 \end{matrix} = \left| \begin{matrix} a_5cos\theta_1cos\theta_2cos\theta_3 - a_5cos\theta_1sin\theta_2sin\theta_3 - d_4sin\theta_1 - a_3cos\theta_1cos\theta_2 - d_2sin\theta_1 \\ a_5cos\theta_1cos\theta_2cos\theta_3 - a_5cos\theta_1sin\theta_2sin\theta_3 + d_4sin\theta_1 - a_3cos\theta_1cos\theta_2 + d_2sin\theta_1 \\ a_5sin\theta_2cos\theta_3 + a_5cos\theta_2sin\theta_3 - a_3sin\theta_2 + d_1 \end{matrix} \right| \tag{21}$$

Now that the coordinates of the tip in relation to the body have been found, inverse kinematics can be used to find the specific joint angles for each joint to have the arm reach the intended location. The process involves finding the angles by using values related to locations near the joint as shown below using the above arm example:

$$\theta_3 = arctan\ 2(sin\theta_j, sin\theta_j) \tag{22}$$

$$\theta_2 = arctan\ arctan\ 2(rd - sc, rc + sd) \tag{23}$$

Where $r = a_5cos\theta_3 + a_3, s = a_5sin\theta_3, c = a_5\ cos\ cos\ (\theta_2 + \theta_3)\ + a_3cos\theta_2,$ (24, 25, 26)

$$d = a_5\ sin\ sin\ (\theta_2 + \theta_3)\ + a_3sin\theta_2 \tag{27}$$

$$\text{And } \theta_1 = arctan\ arctan\ 2(My - Nx, Mx + Ny) \tag{28}$$

$$\text{Where } N = d_4 + d_2, M = a_5\ cos\ cos\ (\theta_2 + \theta_3)\ + a_3cos\theta_3 \tag{29}$$

Now with all of the joint angles, the arm can move till its joints are located at these angles which will equate to the arm being in the correct location to manipulate the object. To keep track of the angles without the use of a stereo camera watching them; the arm's starting position angles could be programmed into the controller. Afterwards any movements made by the arm can be tracked, allowing for the controller to know when it has reached the desired angle.

To improve this method, some worthwhile measures can be taken such as ensuring the image is of high enough resolution to accurately measure and more importantly use algorithms that can estimate the disparity with sub pixel accuracy [71]. This would ensure that all position calculations for the object are as precise as possible which is one of the biggest causes of error in these calculations.

Arm Path Planning using Point Cloud

While path planning using stereo vision is a solid option in the design of our implementation, it is also worthwhile to consider the power and capability of an arm path planning approach that uses point cloud data in contrast to standard stereo imaging since it will be incorporated with the robotic arm. Point cloud has many advantages over stereo imaging with the largest being point cloud's ability to very accurately give spatial information about the environment that greatly outshines the spatial information that a 2D stereo image can give. Because of this many methods for such planning have been explored with many building on the previous method's success. This section will break down some of these methods that were found during research along with important concepts.

Important Concepts

- Configuration Space (C-space): a configuration is the complete specification of the position of every point in the system. The space of all configurations is the configuration space or C-space [72]. This can be generally found for things like an arm and its surroundings by first getting the information of the environment to get an understanding of the arms base position which is used in kinematics to find the specific position of the robotic manipulator.

- Jacobian Matrix: The Jacobian matrix is a matrix used in robotics that relates corresponding small displacements in different spaces and is represented by the following matrix:

$$\delta \mathbf{x} = \begin{bmatrix} \frac{\partial f_1}{\partial q_1} & \cdots & \frac{\partial f_1}{\partial q_n} \\ \vdots & \vdots & \vdots \\ \frac{\partial f_m}{\partial q_1} & \cdots & \frac{\partial f_m}{\partial q_n} \end{bmatrix} \delta \mathbf{q} \tag{30}$$

where f is a function that maps space defined by one variable, which in this case is q, to another variable which will be x for this example [73]. This matrix can also be differentiated with respect to time t to get the following relationship between the velocities of the mechanism in joint and cartesian space:

$$\dot{\mathbf{x}}_{(m\times 1)} = J(\mathbf{q})_{m\times n}\dot{\mathbf{q}}_{(n\times 1)} \tag{31}$$

- Minimum cut set algorithm: The minimum cut set algorithm is an algorithm used to find a cut $(A, \underline{A})$ for any graph $G = (V, E)$ in $O(|V| + |E|)$ time that has a size of no more than $(2 + \epsilon)$ times the minimum cut size [74].

Probabilistic Roadmaps for Robot Path Planning

For path planning to evolve to handle any environment, there must first be a method that is designed for a robot/robotic arm in a static environment. One such method that has become quite used for such static environments is the probabilistic roadmaps (PRM). These roadmaps allow for a robot with 3-16 degrees of freedom to accurately create a path without collisions for the robot to take. This is done through two main stages: the preprocessing and the query stages [75].

Starting off is the preprocessing; in this stage the planner begins by creating a roadmap in the C-space using a probabilistic method based on the environment. This roadmap is an undirected graph that is given the notation $R = (N, E)$ where N is the set of configurations for the robot/robot manipulator chosen over the free C-space and E corresponds to simple paths which are edges between two nodes that have a feasible path between their configurations [75]. The graph, which begins empty, starts adding randomly generated free configurations using the spatial data obtained to N. For each new node q, a number of nodes are selected from the current list of nodes in N and used to try and create connections between q and the selected node q'. Should a successful connection be made then the edge (q, q') is added to E. A simplification of this process is shown in figure ? using pseudocode.

- Δ be a symmetrical function $\mathcal{C}_{free} \times \mathcal{C}_{free} \rightarrow \{0, 1\}$, which returns whether the local planner can compute a path between the two configurations given as arguments;
- D be a function $\mathcal{C} \times \mathcal{C} \rightarrow R^+ \cup \{0\}$, called the *distance function*, defining a pseudo-metric in $\mathcal{C}$. (We only require that D be symmetrical and non-degenerate.)

The roadmap construction step algorithm can be outlined as follows:

1. $N \leftarrow \emptyset$
2. $E \leftarrow \emptyset$
3. **loop**
4. $q \leftarrow$ a randomly chosen free configuration.
5. $N_q \leftarrow$ a set of candidate neighbors of q chosen from N.
6. $N \leftarrow N \cup \{q\}$
7. **forall** $q' \in N_q$, in order of increasing $D(q, q')$ **do**
8. **if** $\neg$*same_connected_component*$(q, q') \wedge \Delta(q, q')$. **then**
9. $E \leftarrow E \cup \{(q, q')\}$
10. Update R's connected components.

Fig 52. Pseudocode for the roadmap construction detailed above for simplification.[75]

Once the roadmap is constructed, it can be expanded if needed to ensure the map possesses enough information for the algorithm to properly query. If the number of nodes in N is large enough than N is an acceptable uniform covering of the free C-space (C_{free}). However, there are environments or scenes where there is not enough nodes for N to be an acceptable covering of C_{free}. In these cases it is necessary to pad R with more nodes such that it facilitates the formation of larger components that more accurately represent the more narrow or difficult parts of C_{free} rather than a collection of smaller components. This is accomplished by defining a weight for each node $w(q)$, which is larger when q is in a difficult region, smaller when q is in an easier region, and has all weights add up to 1. Now the expansion of q can be done by creating new nodes in the free neighborhood of q till the desired M new nodes are added. These selected nodes q are chosen using the probability:

$$Pr\ Pr\ (q\ is\ selected)\ = w(q) \tag{32}$$

along with the weight of node q found using the results from the roadmap construction:

$$w(q) = \frac{1}{d_q+1} / \sum_{t=1}^{N} \frac{1}{d_t+1} \tag{33}$$

where $d_q, and\ d_t$ are the degree of the node.

Now to finalize the preprocessing, there is the component reduction phase which is only used when the roadmap expansion is not able to solve the issue of a non-viable covering of C_{free}. In this situation for any two components, starting with the largest, a pair of nodes are selected with p as the first value and q as the second value. Then the two component's nodes are compared to see if a connection can be produced which would allow for the two nodes to be merged together and a new node pair to be selected [75]. In the case where a connection is not able to be made within a time bound, to ensure the calculations don't take too long, the selected pair is ignored without merging and a new pair of nodes is selected and tested for a connection. Once the algorithm fails to make any more connections for the selected components the components are retained as they are since they are not able to be connected.

With the roadmap created and processed to ensure it is the best roadmap for the environment that the algorithm can make; the planner now moves into the query stage to attempt and find a suitable path for the robot to take. These paths are found between two arbitrary values of configurations that are named q_{init} and q_{goal}. These values are then used to find the feasible paths in the roadmap by attempting to connect q_{init} to a node in R so that the feasible path P_{init} can be found while the same is done for q_{goal} to find P_{goal}. If no connections are made in a specific time limit, the algorithm will begin using bounce walks to try and connect to R. Finally, the final path between q_{init} and q_{goal} can be found by concatenating P_{init}, the recomputed path corresponding to P, and P_{goal} reversed [75]. Now with a path calculated, the robot can begin making its movements along the found path.

Overall this method shows good potential with its experimental results showing that it can handle difficult paths reliably and at a decent rate. The key downside to this method is the requirement for the environment to be static which heavily limits the algorithm's viability in many projects including this one. While this method can't be used for the arm being implemented by this team due to the changing environments that the arm will be in; this method has been built upon to allow for more dynamic roadmap construction in an attempt to create a version that can be used in such changing environments as our rover arm.

Dynamic Roadmaps for Robot Path Planning

Now with an understanding of PRM and how it is used to perform path planning in a static environment the focus of study moves to path planning in a changing environment. To accomplish this task the Dynamic Roadmap planner (DRM) was developed using the PRM as a base and modifying it to better handle these dynamic environments [76].

Understanding these changes is key to grasping the concepts needed to ensure a robot can path plan properly in changing environments.

First similar to PRM, a roadmap is generated in the free C-space (C_{free}) that consists of nodes and edges. Unlike the PRM which already knew about its entire environment and obstacles the DRM has no obstacles during the generation of the roadmap. This leads to developing a roadmap that specifically denotes where each node can connect to and creates every possible configuration that the robot can make in the situation with the only constraint being that there could not be any self-collision so the robot cannot have a configuration that involves two or more links of the robot to intersect. This leads to a fully connected map that can be easily changed as obstacles become present in the environment so that less time has to be done recalculating the entire map each time new obstacles emerge. Another aspect that differs from the PRM approach for generation is that the DRM uses the idea of manipulability to help create samples for the roadmap generation [?]. These manipulability measures are taken from a manipulator Jackobian matrix and help the algorithm by giving a quantitative measure of the robot's ability to position its end effector as well as orientate it from a specific position. The formula for this action is

$$w = \sigma_1 \sigma_2 \dots \sigma_n \tag{34}$$

where σ_i is the singular values of the manipulator Jacobian and w is the quantitative manipulability value. The manipulability value is then used as a bias for sampling data for the road map. If the value is high then the robot has a great amount of dexterity and freedom in these areas of the region and thus don't need too many nodes taken as samples to accurately show the areas that the robot can move through. If the value is low then the robot lacks dexterity in these areas and will need denser collections of nodes in these regions to make proper path planning through them. These values are then used in a histogram whose data is converted into a cumulative distribution function similar to the examples shown below in figure ?. Note that the filtered line is the line from sampling the manipulability of the robot without removing instances of self-collision while the filtered line has removed these instances.

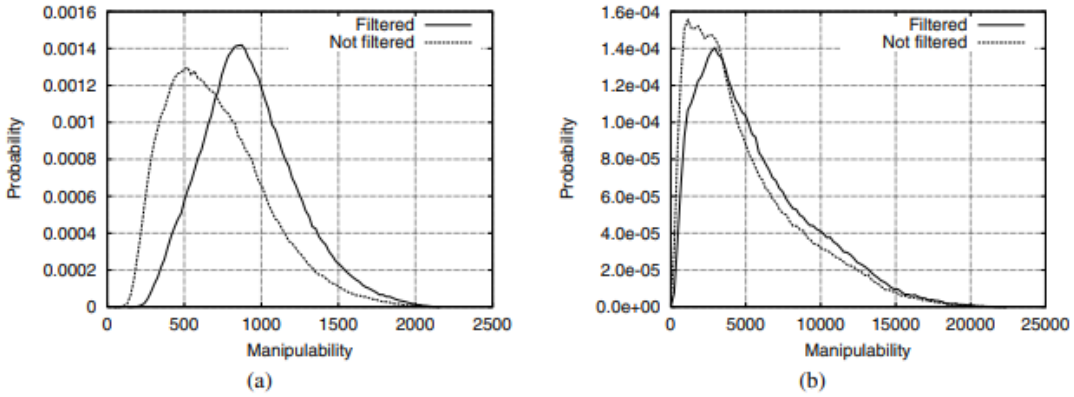


Fig 53. Graphical examples of the probability distribution functions for the manipulability of a robot with six degrees of freedom (6 joints).[76]

These distributions help in the roadmap generation by influencing what samples are randomly generated by the algorithm. The only key issue with this method is that it doesn't take into account the joint limit of each point on the robot. The joint limit is simply the limit of a joint which is found at the max and min of its range of movement. As a joint reaches its limits the manipulator becomes less flexible and restricted. To solve this issue a convention can be adopted into the algorithm to ensure these outliers are handled which is done by defining the manipulability of a configuration close to a joint limit as 0 by including how close the joint is to its limit as a parameter in the manipulability calculations. This makes sure that joint limits have as little negative effect on the generation as possible.

Now that all nodes have been generated, the local planner can begin attempting to connect these nodes. There are many methods that have been developed for this step but the chosen method that was used for the DRM is a simple straight-line planner that, for each arc, attempts to create a straight-line trajectory in the C-space that connects the two configurations [76]. To help the planner, this method also includes the use of a distance function that provides a measurement that represents the difficulty that the local planner would have connecting two configurations. This will greatly help speed up the planner by helping to select pairs of nodes that can be connected in the roadmap. The ideal distance function would be a swept volume of the entire workspace for a trajectory that connects two configurations, but this method is very time consuming and computationally heavy which makes it not fit for an algorithm that is aiming to create a path planning method that is quick and efficient. For this reason, the DRM chooses to use approximations that only use the two configurations that the planner is trying to connect. Four distance functions from other works were chosen to work alongside the algorithm and are detailed below in table 4.

2-norm in C-space:	$\mathcal{D}_2^{\mathcal{C}}(\mathrm{q}, \mathrm{q}') = \|\mathrm{q}' - \mathrm{q}\| = \left[\sum_{i=1}^{n}(q_i' - q_i)^2\right]^{\frac{1}{2}}$
∞-norm in C-space:	$\mathcal{D}_\infty^{\mathcal{C}}(\mathrm{q}, \mathrm{q}') = \max_n \lvert q_i' - q_i \rvert$
2-norm in workspace:	$\mathcal{D}_2^{\mathcal{W}}(\mathrm{q}, \mathrm{q}') = \left[\sum_{\mathrm{p}\in\mathcal{A}} \|\mathrm{p}(\mathrm{q}') - \mathrm{p}(\mathrm{q})\|^2\right]^{\frac{1}{2}}$
∞-norm in workspace:	$\mathcal{D}_2^{\mathcal{W}}(\mathrm{q}, \mathrm{q}') = \max_{\mathrm{p}\in\mathcal{A}} \|\mathrm{p}(\mathrm{q}') - \mathrm{p}(\mathrm{q})\|.$

Table 4. Distance formulas used in the DRM. [76]

where n is the number of joints that the robot has, q and q' are two different configurations corresponding to nodes in the roadmap, q_i refers to the configuration of a specific joint i in the configuration q, and $p(q)$ is the workspace reference point p from the set of reference points A at configuration q [76]. These formulas capture the cost of a connection between two configurations through a measurement that is defined only on the endpoints of the path. These calculations are done quite quickly but lack the ability to account for the motion of the robot as it travels along the designated path; in order to fix this issue some of the formulas are slightly modified to include the midpoint of the path.

$$q_m = (q + q')/2 \tag{35}$$

Using the midpoint, the first of the functions to be altered is D_2^W which simply has the midpoint function added to the end of the formula as shown below.

$$\mathcal{D}_{m2}^{\mathcal{W}}(\mathrm{q}, \mathrm{q}') = \left[\sum_{\mathrm{p}\in\mathcal{A}} \|\mathrm{p}(\mathrm{q}') - \mathrm{p}(\mathrm{q}_m)\|^2 + \sum_{\mathrm{p}\in\mathcal{A}} \|\mathrm{p}(\mathrm{q}_m) - \mathrm{p}(\mathrm{q})\|^2\right]^{\frac{1}{2}}. \tag{36}$$

The second formula that must include the midpoint formula to account for motion is based on the coordinate frame of the end effector for the robot. Letting F_a and F_b denote the two coordinate frames being considered, the distance between the two frames can be found with

$$d(F_a, F_b) = d_x + d_y + d_z \tag{37}$$

where d_x, d_y, and d_z are the distances in Euclidean between the unit vectors along each of the axes respectively [76]. Using this formula, the distance between two configurations can be defined as

$$D_F^W(q, q') = d\big(F(q), F(q_m)\big) + d(F(q_m), F(q')) \tag{38}$$

where q and q' are the configurations being considered, q_m is the midpoint if the path as defined above, and $F(q)$ is the frame of the end-effector corresponding to the configuration q. While this method still has issues such as triangle inequality and using the midpoint in these calculations increases the amount of time needed on computations, the method has shown to save time in other aspects of the algorithm and does an adequate job of accounting for the motion of the arm [76].

With the roadmap constructed, the following step is to map the workspace to that roadmap where the workspace is a representation of the obstacles in the environment. The workspace is represented as a uniform, rectangular decomposition and is denoted as W. Considering that the C-space is denoted by C, the mapping $\phi: W \rightarrow C$ as

$$\phi(w) = \{q | A(q) \cap w \neq \emptyset\} \tag{39}$$

where w is the cell of the workspace, $A(q)$ is the subset of W that is occupied by the robot's configuration at configuration q [76]. This formula results in $\phi(w)$ being equal to the C-space obstacle region of w if w is considered to be an obstacle in the workspace. Since this method doesn't explicitly use the C-space directly, this mapping can be redefined into two mappings for the roadmap with the first mapping the workspace w to the nodes of the roadmap N and the second mapping the workspace w to the arcs of the roadmap E

$$\phi_N(w) = \{q \in N | A(q) \cap w \neq \emptyset\} \tag{40}$$

$$\phi_E(w) = \{\gamma \in E | A(q) \cap w \neq \emptyset \; for \; some \; q \in \gamma\} \tag{41}$$

where q is a configuration and γ is a connection that connects two configurations [76]. An example of this mapping is shown below for a two linked robot.

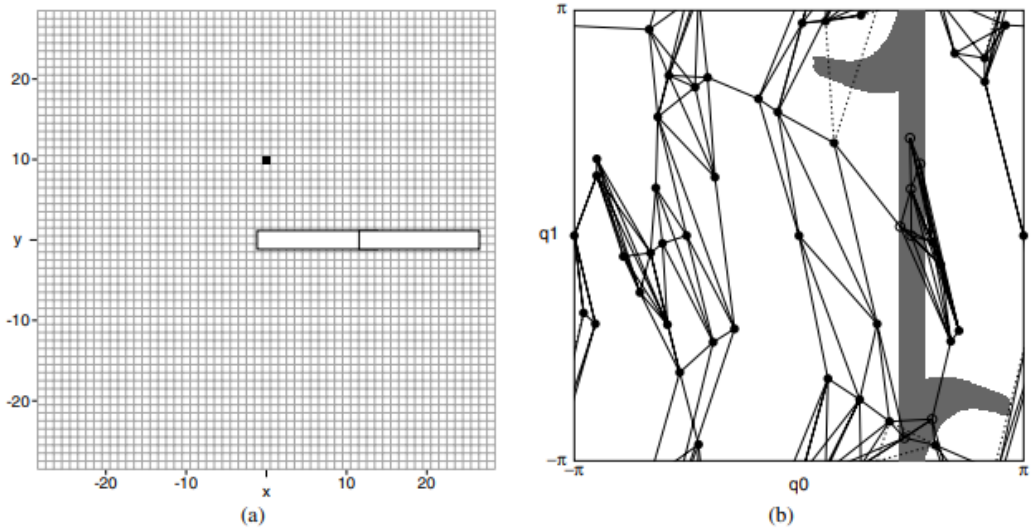


Fig 54. An example of mapping the workspace with the roadmap/C-space. (a) The workspace of the robot with an obstacle at (0,10). (b) The C-space of the robot after mapping. [76]

These mappings are used to account for the obstacles in the workspace since during roadmap generation of the DRM it is assumed that the workspace is clear and without any obstacles.

Lastly before the querying stage of the DRM, which is the same as the PRM, there is an enhancement stage that takes place between the preprocessing and query stages. The key goal of this stage is to handle the issues that the PRM has when dealing with narrow openings in C_{free}. This is done with one of two methods depending on the situation; first is the minimum cut set which is a traditional measure of the graph's connectivity and the second is a quantitative measure of the robustness of the roadmap regarding the addition of the obstacles in the workspace.

For the minimum cut set, first the edge connectivity must be defined as the minimum of the maximum number of arc-distinct paths between any distinct pair of nodes in the roadmap. This is under the idea that regions that have a low connectivity have clusters of nodes that can be easily disconnected with the addition of a small set of obstacles in the robot's workspace. This can be problematic and thus the goal becomes to have the edge connectivity of the roadmap be at least equal to the minimum degree of any node in the roadmap. To accomplish this an unweighted minimum cut set algorithm is used to determine the minimum number of arcs that partition the roadmap into two pieces [?]. With the two partitions arcs can be added between the two pieces until the minimum cut set is equal to the minimum node degree where the algorithm stops.

The second method using the robustness of the roadmap, examines the relationship between the roadmap and the workspace with a quantitative value. P-robustness of a gragh G is defined by whether there is a spherical obstacle in the workspace of diameter p (or less) that can cause G to become disconnected. If there is no such obstacle than the graph is p-robust, otherwise it is not p-robust. This is used to add arcs to the roadmap until the desired level of p-robustness is reached. Starting off with a 1x1x1 cube from the workspace, the algorithm calculates the set of connected components of that cube. If there is more than one component in the set of connected components then add an arc if possible to facilitate repairing the roadmap. This continues with increasing sizes of cubes until a cube of side length $p + 1$ is reached [76].

Both methods have their advantages and disadvantages but by working together can make up for a lot of their weaknesses. Overall, the enhancements can make a big difference in

the connectivity of a roadmap as shown in the figure below. This figure shows the difference between the roadmap and its connections before and after enhancement to show how much more connected it is over the narrow areas that the first map missed connections for.

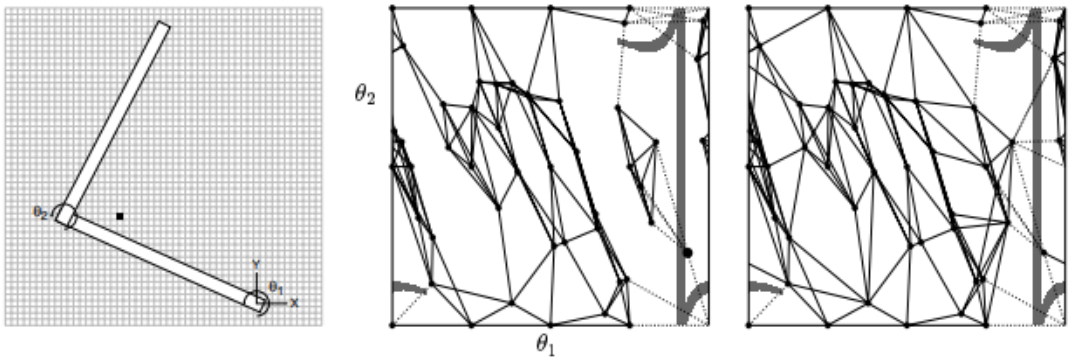


Fig 55. (a) Example workspace with roadmaps for (b) before and (c) after enhancement. [76]

Advancements in the DRM Algorithm

The DRM is a powerful algorithm for capturing spatial data and converting it into a roadmap for path planning robotics in a changing environment. While this algorithm on its own is capable of accomplishing the needs of this project on its own, it is worthwhile to analyze ways to improve the algorithm by finding bottlenecks that limit the real-time capabilities of the robot's movements. The two key advancements that have been made regarding the algorithm is efficient connection of the start and goal configurations to the roadmap and an A* search using an admissible, consistent, and fast heuristic [77].

For the enhancement to the start to goal connection algorithm, rather than using the midpoint metric as the original DRM used a combination of the direct workspace metric as shown below

$$d_2^W(p,q) = \sqrt{\sum_{a\in A} \quad |a(p) - a(q)|^2} \tag{42}$$

along with a weighted L^1metric in the configuration space. This is done because the midpoint metric method, while being more informed, has significant overhead with the amount of calculations that must be done. This leads to a huge decrease in efficiency which is important when the robotic arm is meant to be used in an environment that requires real-

time reactions to obstacles entering the workspace of the arm. To also help with efficiency since the direct workspace metric needs reference points for each of its terminal configurations; the reference points for these calculations are precomputed during the generation of the roadmap and its nodes. These reference points are then saved in the nodes to allow for quick and easy access to the information when the workspace metric needs to be calculated [77]. This leads to the final metrics only needing two computations of forward kinematics as shown in the equations below:

$$d_{combined}(p,q) = 0.9\left(d_2^W(p,q)\right) + 0.1(d_{w1}(p,q)) \tag{43}$$

$$d_{w1}(p,q) = \sum_{i=1}^{n} \quad w_i|p_i - q_i| \tag{44}$$

where w_i is a wight at a specific connection, p is the starting node, and q is the goal node [77].

The graph search using the A* search method also greatly helps the performance of the overall algorithm . This search's main goal is to find the shortest collision-free path for the arm to take to reach its desired destination. The heuristic used is the distance to the goal with the following metric that comes from the direct workspace metric being multiplied by a constant factor as shown below:

$$d_{2c}^W(p,q) = \sqrt{\frac{1}{2}}d_2^W(p,q) = \sqrt{\frac{1}{2}\sum_{a\in A} \quad |a(p) - a(q)|^2} \tag{45}$$

This heuristic is consistent and admissible since it satisfies the triangle inequality and is a lower bound on the midpoint workspace metric [77]. This A* search showed solid improvement in the DRM algorithms with the majority of that contribution being thanks to the pre calculations done during the roadmap generation for the reference points in the previous method since that saved a very large amount of time. These improvements are visualized in the figure below comparing the A* search with the uninformed search used in the original DRM algorithm.

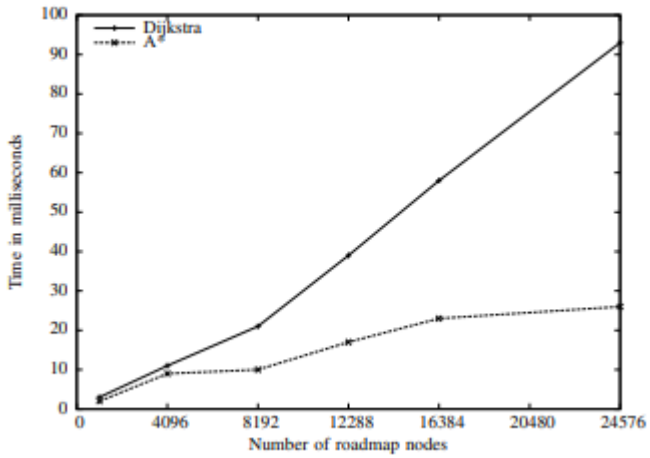


Fig 56. Comparison in running time between the A* search method and the uniform search method originally used in the DRM algorithm. [77]

As can be seen by the figure above, the A* search was over three times more efficient than the original search method. Should the DRM be used for path planning of the robotic arm, it will likely be a must to include these enhancements to the algorithm due to the level of speed-up that is shown.

Current EZ-RASSOR

The current EZ-RASSOR simulation is simple yet effective in modeling the rover's interactions as a real robot would. The design features two barrels for collecting regolith (moon dirt), one on the front and back. The rover has a body, four wheels, and a front camera.

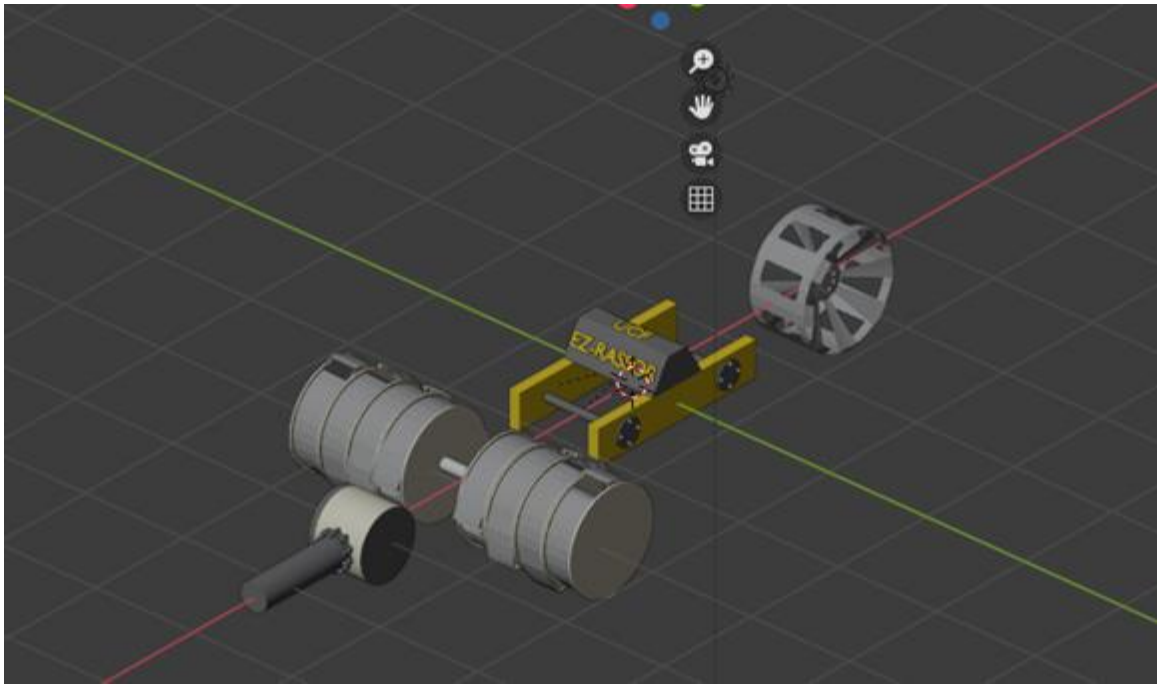


Fig 57. EZ-RASSOR model open in Blender

The Blender model (above), contains all the simple meshes for the visualization of the robot. The model is then assembled with URDF specifications and loaded in the simulation (below).

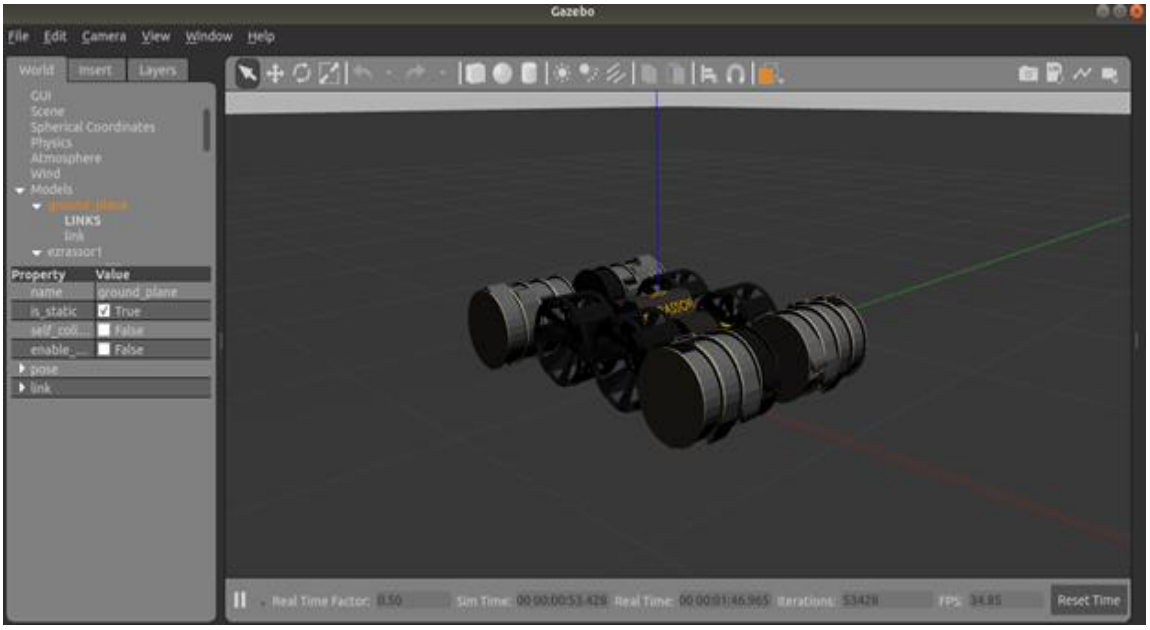


Fig 58. EZ-RASSOR in Gazebo simulation.

This simulation contains keyboard controls as well as autonomous options and swarm controls for multiple rovers. In this state, the rover's function is to collect regolith and deposit it elsewhere. This is all with the objective of creating level terrain in an irregular environment.

All the current EZ-RASSOR designs are available online on the prior UCF teams github. This includes all file architecture and everything necessary to run the simulation in its current state.

Design Summary

The following 2 figures (figure 59 and figure 60) are high level block diagrams that show the flow of information through each component in the implementation design. While the rest of the design is detailed in depth below these figures, these figures are meant to show a brief overview/summary of the implementation design.

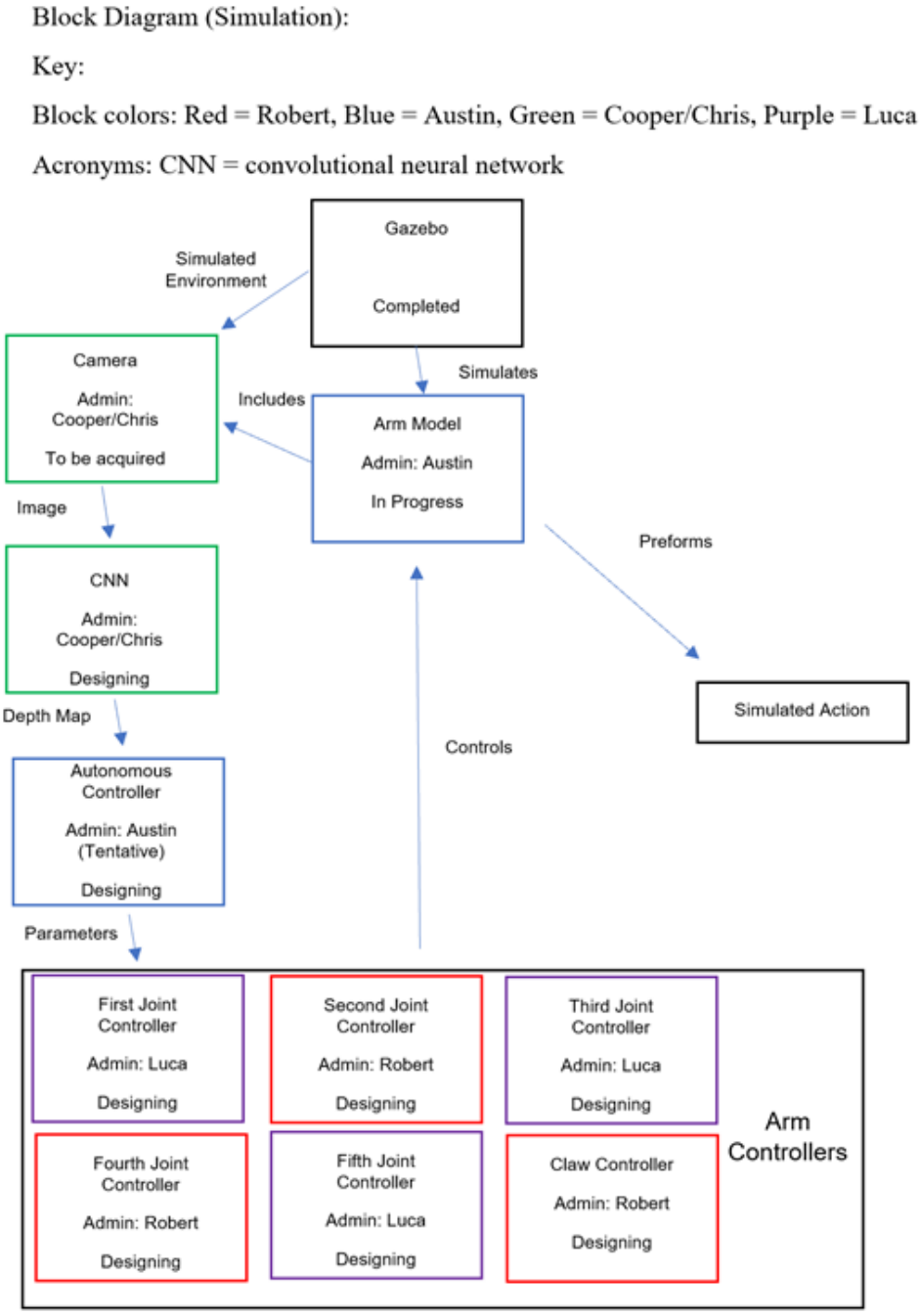


Fig 59. High level block diagram summarizing the design of the simulation implementation

First, figure 59 shows the implementation design that will be integrated with the simulation environment to test that the implementation is viable. The Gazebo simulation first generates including the simulation environment, simulation rover, and simulation camera.

Using the camera, the simulation begins taking stereo color images as input and sends this information to the CNN/Visual Network. This network uses the image to find the center position of the paver and end-effector of the robotic arm as well as monitor motion tracking to account for any sway in the rover or arm. Passing these values to the autonomous controller, inverse kinematics are used to get the velocity controller parameters to get the end-effector to the desired location to place/pickup the paver. Finally, these parameters are passed to the joint velocity controllers that cause the arm to take the necessary actions to complete its task.

Block Diagram (Real-World):

Key:

Block colors: Red = Robert, Blue = Austin, Green = Cooper/Chris, Purple = Luca

Acronyms: CNN = convolutional neural network

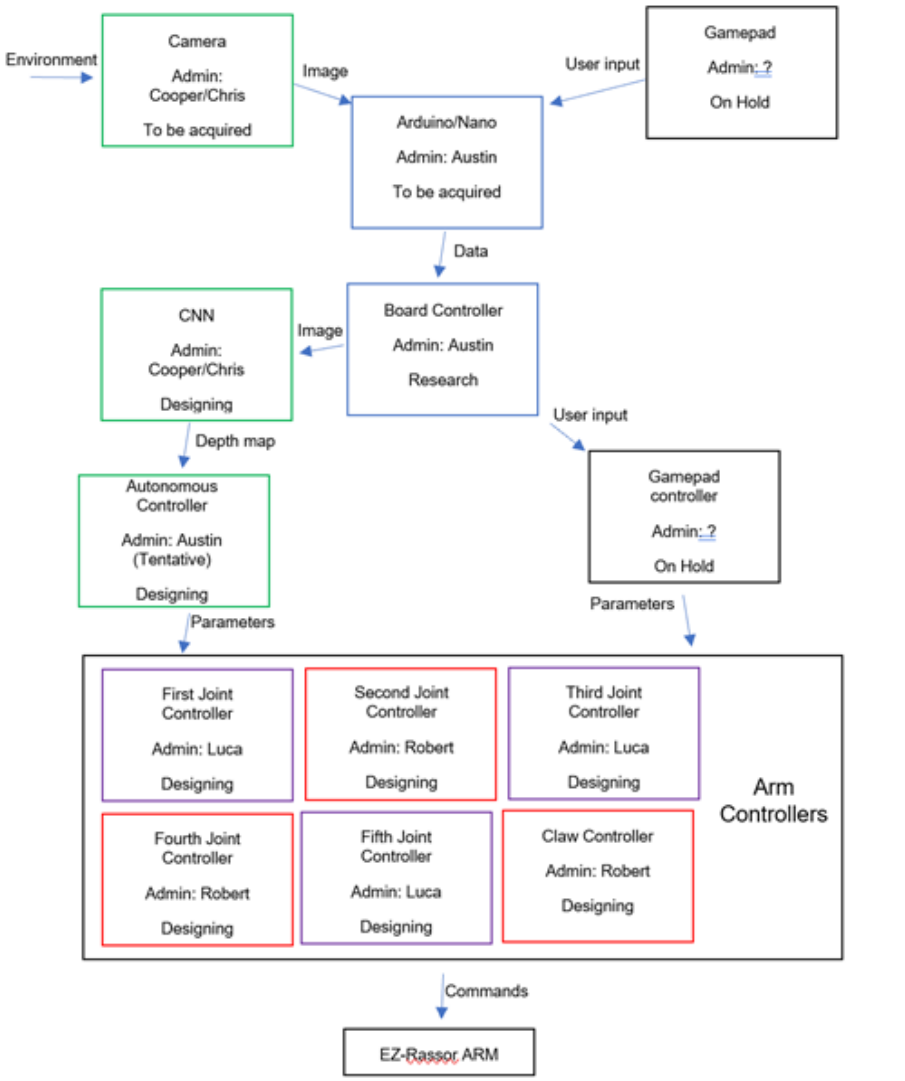


Fig 60. High level block diagram summarizing the design of the real-world implementation.

Secondly, figure 60 shows the implementation design that will be integrated with the real-world rover. This integration will take place after the simulation has been developed and extensively tested to ensure the implementation design is ready for real-world testing. The overall design of the real-world implementation is very similar to the simulation except that the Gazebo elements have been replaced with real world components. The key change that must be incorporated into the real-world is the inclusion/integration of microprocessors on the rover: an Arduino for general functionality and a Jetson Nano for heavy computational programs and neural networks/AI. Accounting for this; the boards will start the autonomous controller and camera at initialization. The camera will then begin taking in images and sending them to the Arduino board. This causes the board to send the image to the Jetson Nano and start the CNN/Visual network for edge detection and motion tracking to get the needed parameters for the autonomous controller as explained in the simulation summary. These values are returned to the Arduino to be sent to the autonomous controller program that creates the velocity parameters for the joint controllers. Finally, these velocity parameters are sent back to the Arduino that publishes the data to the motors causing them to move as intended.

ROS

For our robotic framework/middleware to communicate with the rover, we have chosen ROS and more specifically ROS Melodic. ROS was chosen for a few noteworthy reasons; firstly, being that the previous teams have built the rest of the rover's communication using ROS. While it would be nice to try out some of the other middleware, using ROS will ensure that our code will be compatible with the current repository and won't require extensive time devoted to integrating ROS and another system. Outside of the previous reason, ROS also brings a large amount of functionality and libraries to the rover that allow for a quicker and more reliable development. The choice of ROS Melodic over the newer version, ROS Noetic or even ROS2, allows the project to have a reliable middleware that isn't going through heavy development/changes and is stable.

Simulation Model Phase 1

Design for the simulation when through several phases throughout the approximately three-month work period thus far. Design started with research on the simulation software mainly including gazebo, blender, and ROS architecture. This research process is discussed in detail in the research section. The next phase involved getting familiar with previous teams

work with simulation. The EZ-RASSOR rover had already been simulated by a previous UCF senior design team. The existing simulation and previous work are also discussed in the research section.

Next in the design process was analysis of the arm created by Jacob and determining what functionality it had, to be simulated. The arm (below) is designed to pick up a paver, stack it on the rover, several pavers high, and then remove the pavers and place them together to create a floor. The movement of the arm is designed to balance fluid motion and simplicity of design.

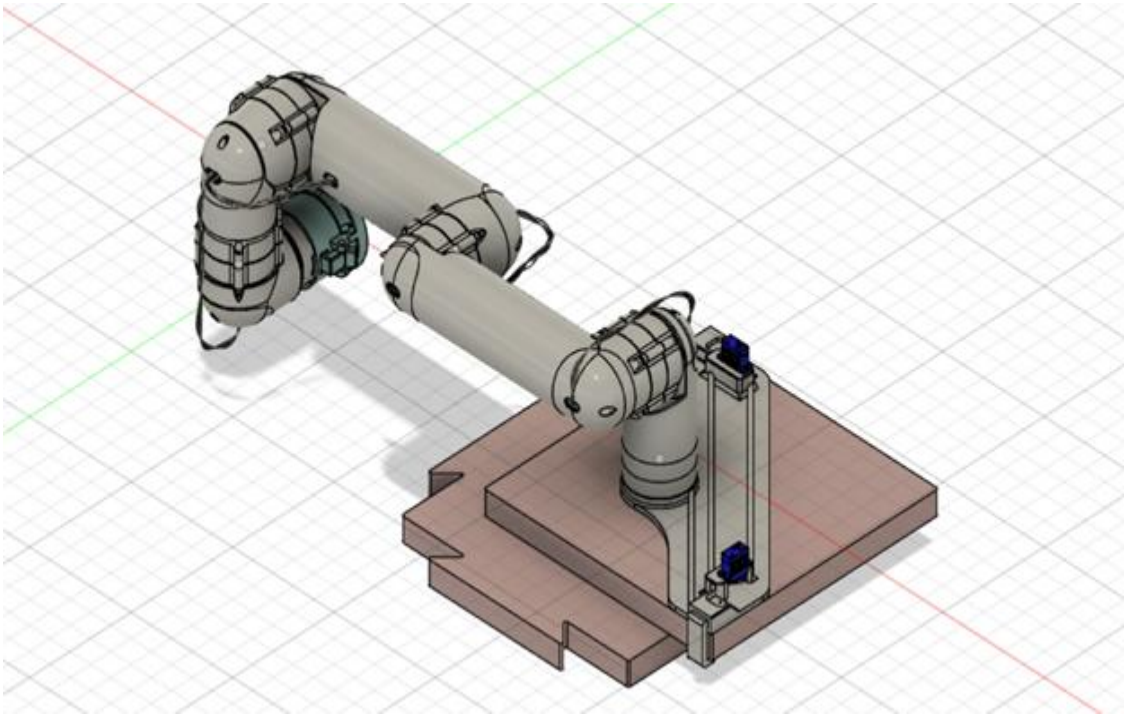


Fig 61. the arm design as seen in Autodesk Fusion 360.

The arm with its five degrees of motion would have six links and five joints as described in URDF (unified robot design format). The drawing (below) was my attempt to visualize the joint design similar to the visualization in figure 61. after looking at the arm design in Autodesk and watching the video demo of the arms movement, this was my rough estimation of the arms chain of links.

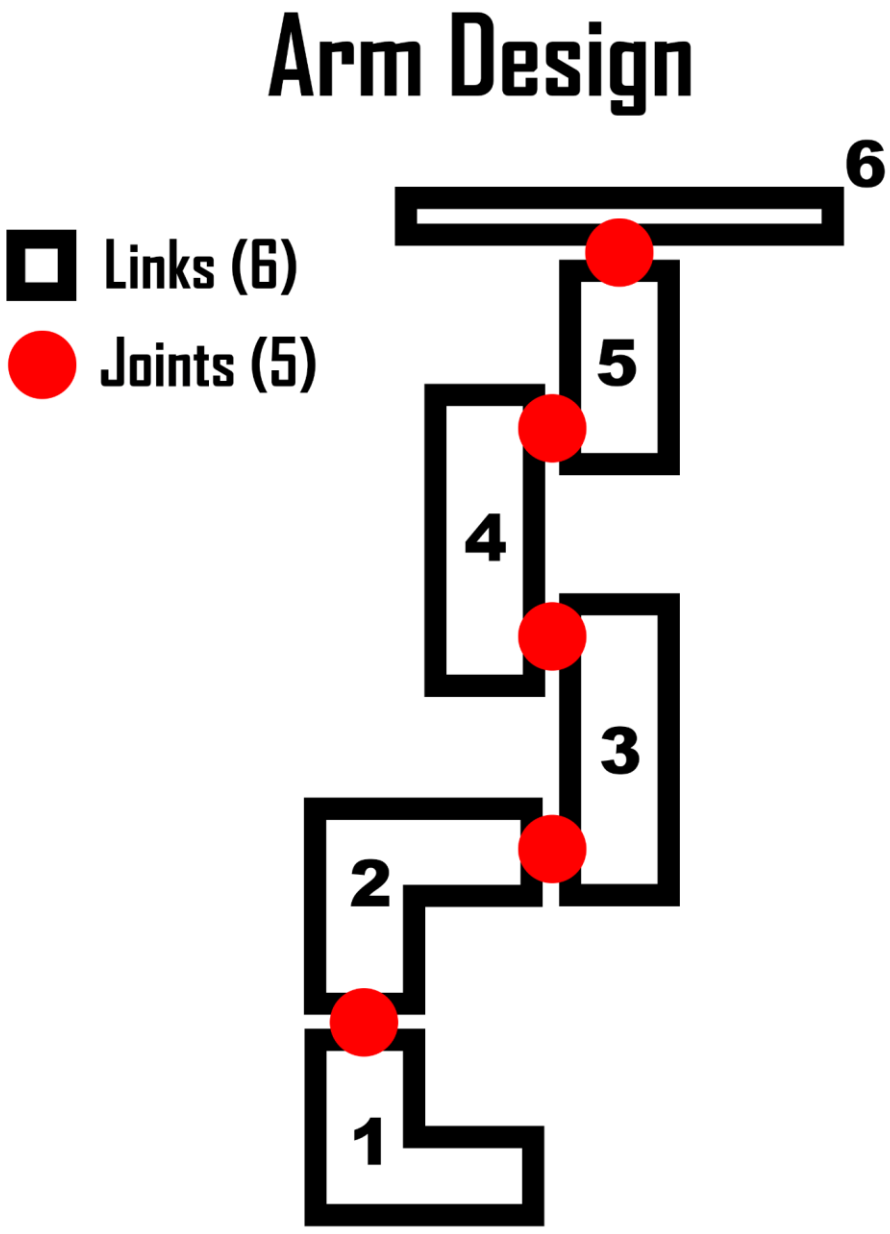


Fig 62. Early arm conceptualization for simulation design.

Joint type and axis are also considerations, when factoring in the starting orientation of the arm and its movement. The arms links rotate against each other which indicates a hinge type of motion, this is best represented by the continuous joint type. The drawing (below) provides a visualization of the arm only factoring the joints, 6 joints total including the first one that is fixed to the base link.

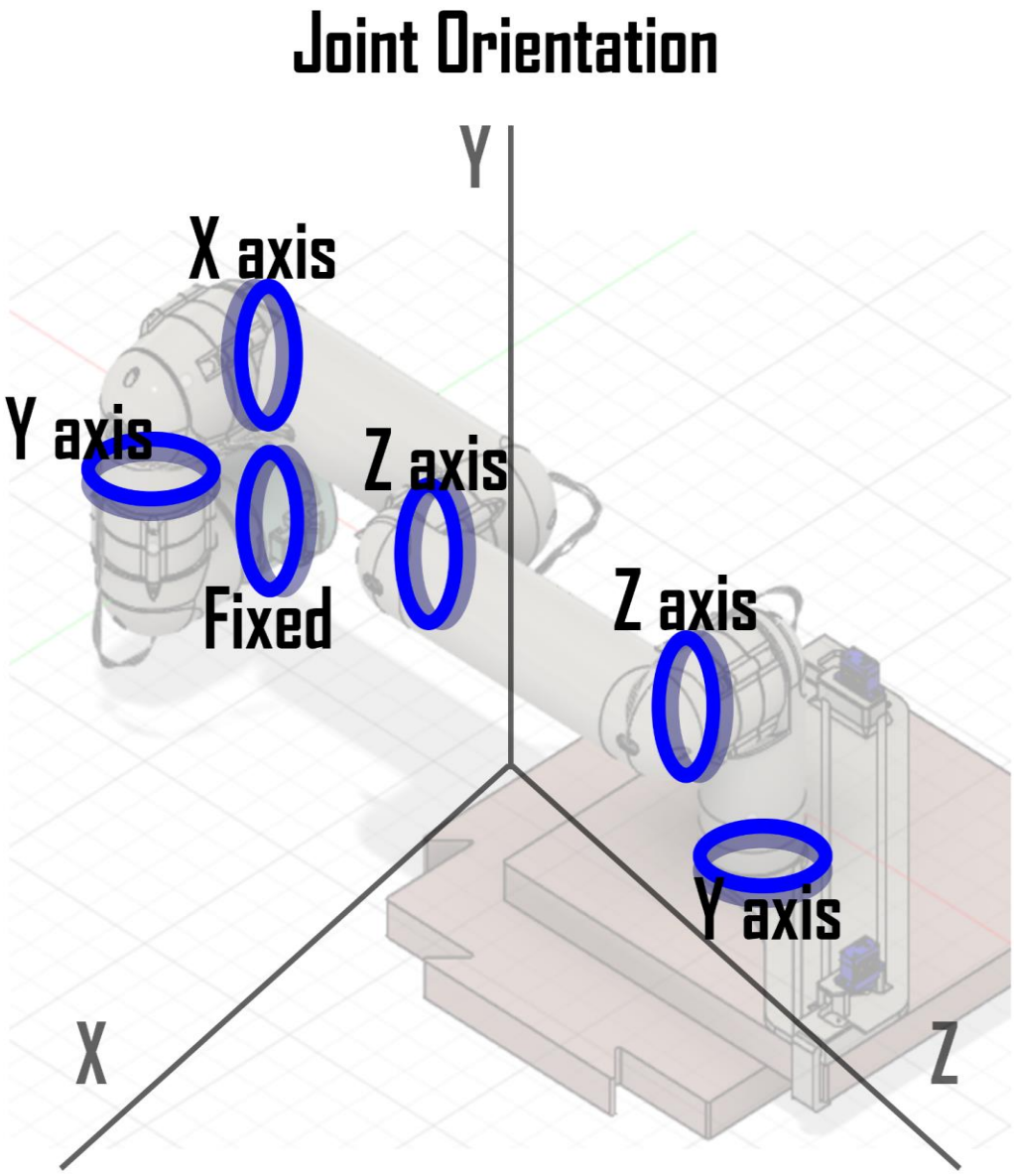


Fig 63. Arm joint orientation illustration.

Initially the arm model created in Autodesk Fusion 360 was used as the texture for the simulation arm; this made design in blender simple. This model was first exported from

Autodesk as a step file. This step file was opened in blender, the modeling tool for use with Gazebo simulations. In blender, the arm file was divided into the six links and the paver included in the file was temporarily discarded. These six links were then oriented and exported as .dae files for use as meshes in URDF.

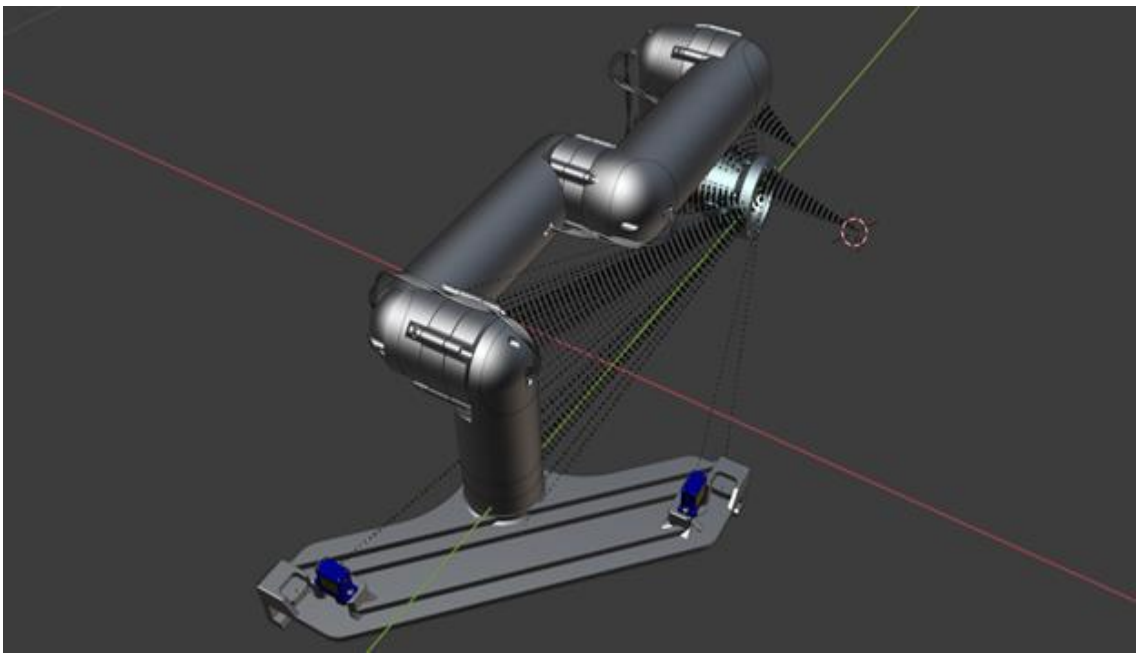


Fig 64. AutoDesk Arm design imported into Blender

The arm with its six pieces was to be attached to the existing rover simulation. This was accomplished by modifying the existing rover simulation files. This took place entirely in the ROS package: ezrassor_sim_description.

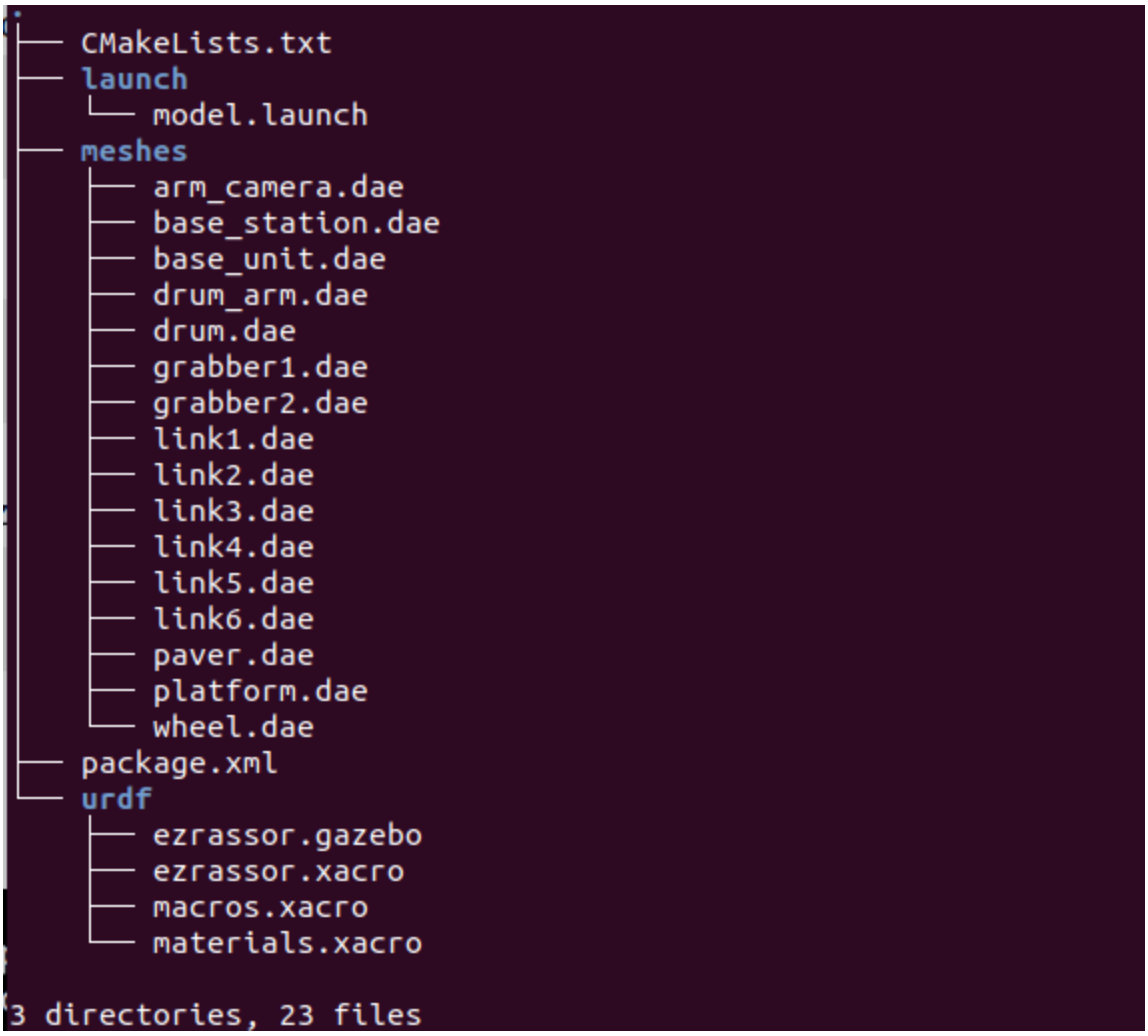


Fig 65. File architecture of ezrassor_sim_description.

Macros.xacro and ezrassor.gazebo were both modified to include a reference to the new arm pieces added to the robot. In macros.xacro, the robot meshes were linked and their scale compared to the rover was adjusted. The size of the arm meshes was considerably bigger than the rover, so they were scaled down considerably.

The file ezrassor.xacro was modified more considerably. Added into this file were detailed specifics of the arm links and joints. The front barrel of the robot was removed, and the arm was positioned to take its place as depicted in figure 66.

```
<!-- Arm links -->

<link name='link1'>
  <pose>0 0 0 0 0 0</pose>
  <inertial>
    <mass value="1.0"/>
    <origin xyz="0 0 0" rpy=" 0 0 0"/>
    <inertia
        ixx="0" ixy="0" ixz="0"
        iyy="0" iyz="0"
        izz="0"
    />
  </inertial>
  <visual name='link1'>
    <origin xyz="0 0 0" rpy=" 0 0 0"/>
    <geometry>
      <link1/>
    </geometry>
  </visual>
</link>
```

Fig 66. Arm link in URDF ezrassor.xacro.

An example (above) of one of the six arm links created in URDF for simulating the arm of the EZ-RASSOR several of the joints connecting the links (below). These joints are created based on orientations resolved in figure 63.

```
<!-- ARM Joints-->

<joint name="joint0" type="fixed">
  <axis xyz="0 0 0" />
  <origin xyz="0.5 -0.1 -0.1" rpy="0 0 0"/>
  <parent link="base_link"/>
  <child link="link1"/>
</joint>


<joint type="continuous" name="joint1">
  <origin xyz="0 0 0" rpy="0 0 0"/>
  <child link="link2"/>
  <parent link="link1"/>
  <axis xyz="0 0 1" rpy="0 0 0"/>
  <limit effort="100" velocity="100"/>
  <joint_properties damping="0.0" friction="0.0"/>
</joint>

<joint type="continuous" name="joint2">
  <origin xyz="0 0 0" rpy="0 0 0"/>
  <child link="link3"/>
  <parent link="link2"/>
  <axis xyz="0 1 0" rpy="0 0 0"/>
  <limit effort="100" velocity="100"/>
  <joint_properties damping="0.0" friction="0.0"/>
</joint>
```

Fig 67. Arm joints in URDF ezrassor.xacro.

The following figure is a graphviz diagram of the URDF file ezrassor.xacro with the added arm links and joints. This was created with the command: “urdf_to_graphix <(xacro ezrassor.xacro)”

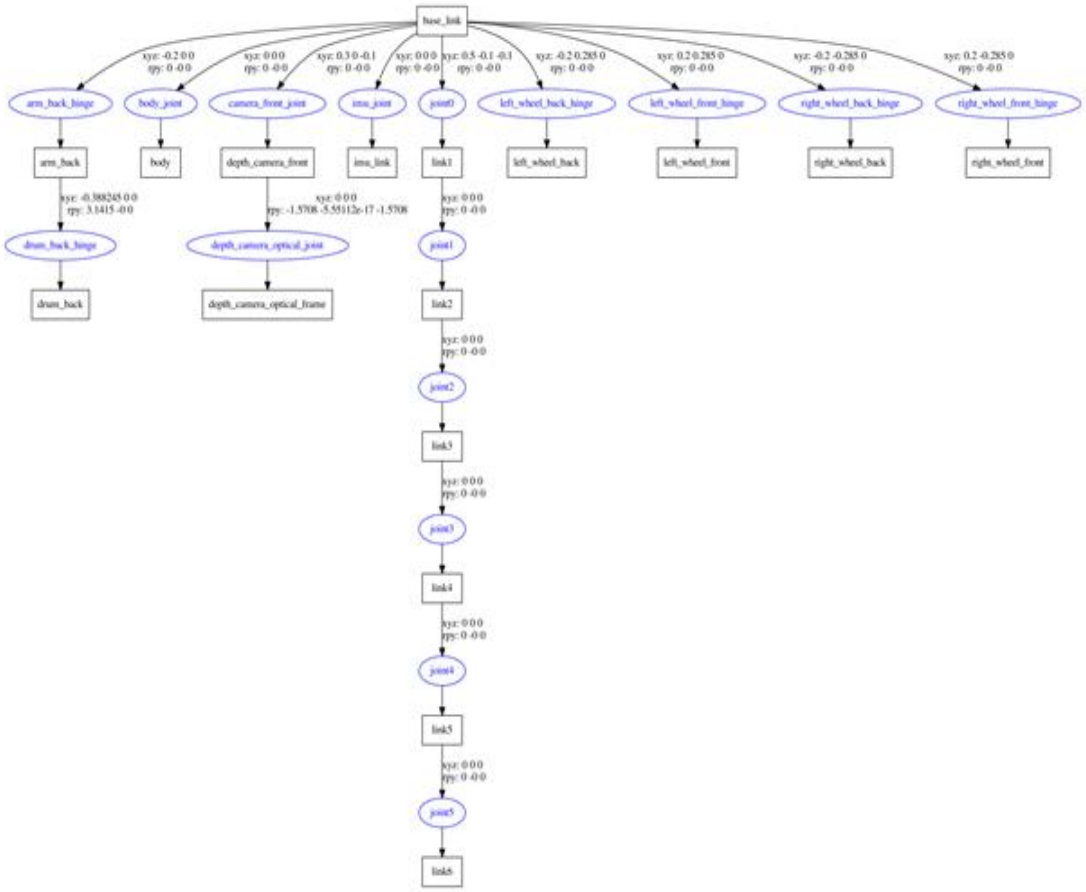


Fig 68. Early graphviz visualization of the robot's architecture.

The longest line of objects shown in the visualization (above), is related to the arm design as it has 6 links all connected in a chain while the other robot links are more central to the base. The degree of separation the end of the arm has from the center of the robot leads to physics considerations not factored as heavily with other links like inertia.

The gazebo simulation, now with the arm, is loaded simply with: "roslaunch ezrassor_launcher configurable_simulation.lauch control_methods:= keyboard"

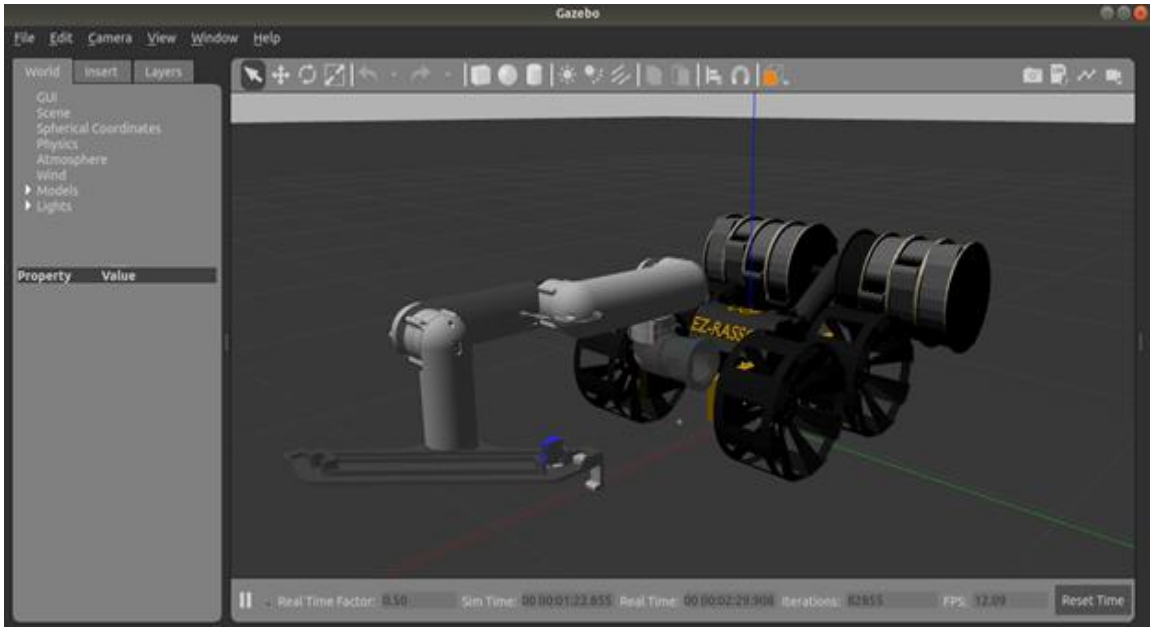


Fig 69. EZ-RASSOR with arm addition in Gazebo simulation.

Open ezrassor.xacro ~/ezrassor_ws/src/ezrassor_sim_description/urdf Save

```
</transmission>

<transmission name="joint1_tran">
  <type>transmission_interface/SimpleTransmission</type>
  <joint name="joint1">
    <hardwareInterface>hardware_interface/VelocityJointInterface</hardwareInterface>
  </joint>
  <actuator name="joint1_motor">
    <hardwareInterface>hardware_interface/VelocityJointInterface</hardwareInterface>
    <mechanicalReduction>1</mechanicalReduction>
  </actuator>
</transmission>

<transmission name="joint2_tran">
  <type>transmission_interface/SimpleTransmission</type>
  <joint name="joint2">
    <hardwareInterface>hardware_interface/VelocityJointInterface</hardwareInterface>
  </joint>
  <actuator name="joint2_motor">
    <hardwareInterface>hardware_interface/VelocityJointInterface</hardwareInterface>
    <mechanicalReduction>1</mechanicalReduction>
  </actuator>
</transmission>
```

Fig 70. Transmissions for arm joints in URDF ezrassor.xacro,

The model (figure 69) was the first prototype of the simulated EZ-RASSOR with the arm attachment. Controls were added with ROSpy and the joints were given transmissions

(figure 70) both by Robert, but calibration of the arm movements had yet to be synchronized.

Simulation Model Phase 2

Through the design process, and discussion with teammates and sponsor Mike Conroy, it was decided to create new meshes for the arm visuals rather than using the AutoDesk models due to the unnecessary size of the models that bogged down the running of the simulation unnecessarily. Initial use of the AutoDesk models provided the benefit of early visualization of the arm in Gazebo but it was in need of updating.

With the AutoDesk arm as reference (figure 64), new, simpler, shapes were created in blender to visually represent the arm in the simulation. These shapes, the six, arm links, were created using blenders shape tools, most of them starting as cylinders. The blender models of these arm segments are pictured below.

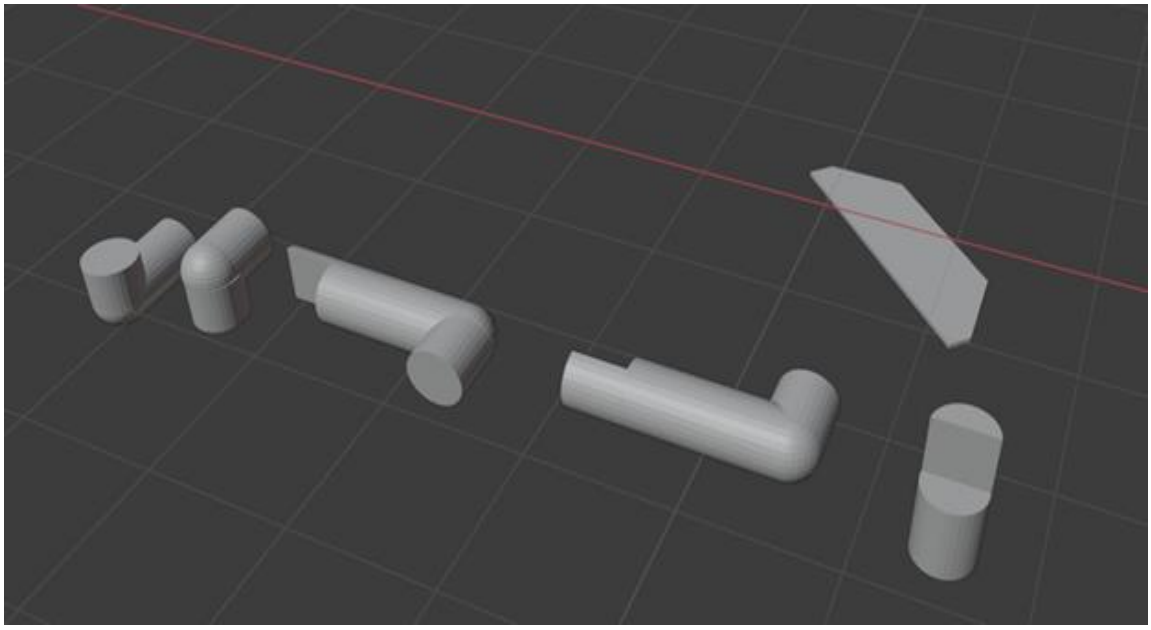


Fig 71. Blender arm segments.

Now that the rover had the new arm meshes, the speed of the simulation loading time was approximately quadrupled.

One significant problem remained, while the arm now had functioning movements binded to keys, these movements were still uncalibrated, meaning they didn’t move or appear as

intended. URDF defaults the joint for each link to be at the origin of the link. This results in all links being placed on top of each other.

Fixing this issue started as a guess and check process of tweaking the joint and link origin values in the robot description to reach the desired movement. This proved to be too tedious to be feasible, so I stepped back and implemented a new approach, a mathematical analysis of the robot's exact dimensions in Blender divided by the simulation scale (figure 73).

Fig 72. Macros file in URDF.

The macros file contains the references to the robot's meshes (above). The scale of the original rover was 35%. For consistency, this scale was kept and the new additions to the rover were modified to match this scale.

Robot Dimensions: as described in blender measurements

Wheel Diameter = 1.02

Body = 1.77 x 0.77

Barrel diameter = 1.0

Barrel arm diameter = 0.63 length = 0.94

Arm:

- Diameter = 0.4
- Link 1 = 0.82w 0.53l
- Link 2 = 0.61w 0.53l
- Link 3 = 1.6w 0.59l
- Link 4 = Link 3
- Link 5 = 0.887l
- Link 6 = 1.0w 0.3l

Scale = 0.35

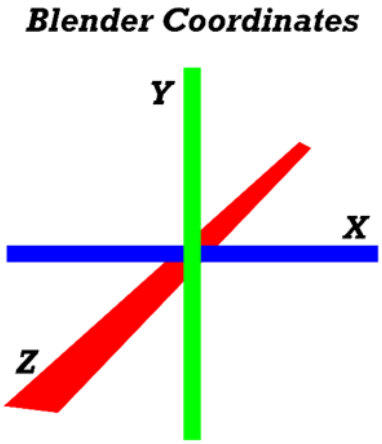


Fig 73. Work on robot dimensions.

By this mathematical analysis, an accurate approximation of each joint was established and the appropriate position of the meshes (below).

Relative Arm Positions: in URDF and Gazebo

Joint Positions: <link size>/2 *scale

Joint 0 = fixed

Joint 1 = -0.12y 0.09z

Joint 2 = 0.037y 0.16z

Joint 3 = 0.25x 0.037y

Joint 4 = 0.25x 0.037y

Joint 5 = 0.155z

Visual Positions:

Link 1 = fixed

Link 2 = 0.185z

Link 3 = 0.26x 0.08z

Link 4 = 0.26x

Link 5 = -0.155z

Link 6 = -0.155z

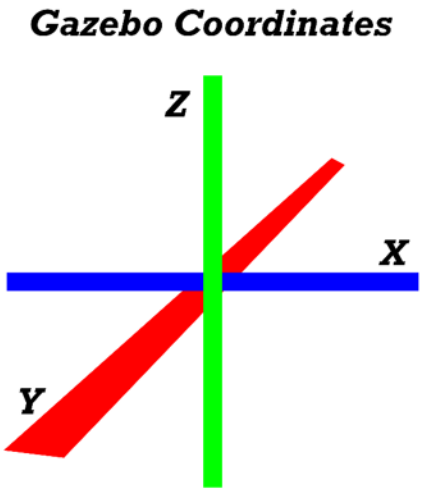


Fig 74. Work on relative positions of links and joints.

Now, the simulation had a working arm attached but it still needed work to better approximate the final rover design as pictured in figure 61. The rover, as pictured in figure 69 still needed the proper attachment of the arm to the base (currently floating above the rover), the arm camera housing, and the platform above the base for placing pavers on, all pictured in figure 61. The arm for the front barrel, previously removed, was readded to the simulation and the arm was attached to the end of it. On the other side of the barrel arm end was added a model of the camera housing made in Blender. A new model more accurately depicting the 6th link was modeled. This version of the rover is displayed below.

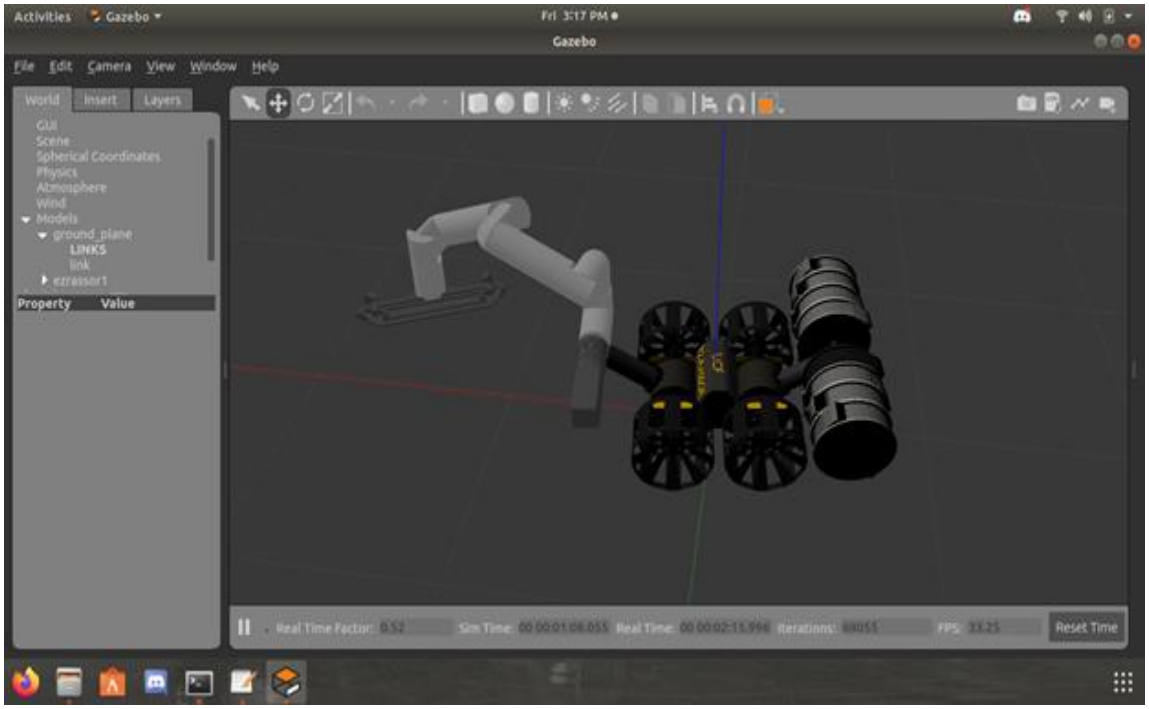


Fig 75. EZ-RASSOR with camera housing and proper arm attachment in Gazebo.

The final phase of design for this semester involved the addition of the paver platform to the model and the addition of the two grabbers that give the arm the functionality to pick up pavers. Keyboard control and calibration for the arm grabbers have yet to be simulated. The final version of the model with all appropriate elements is pictured below.

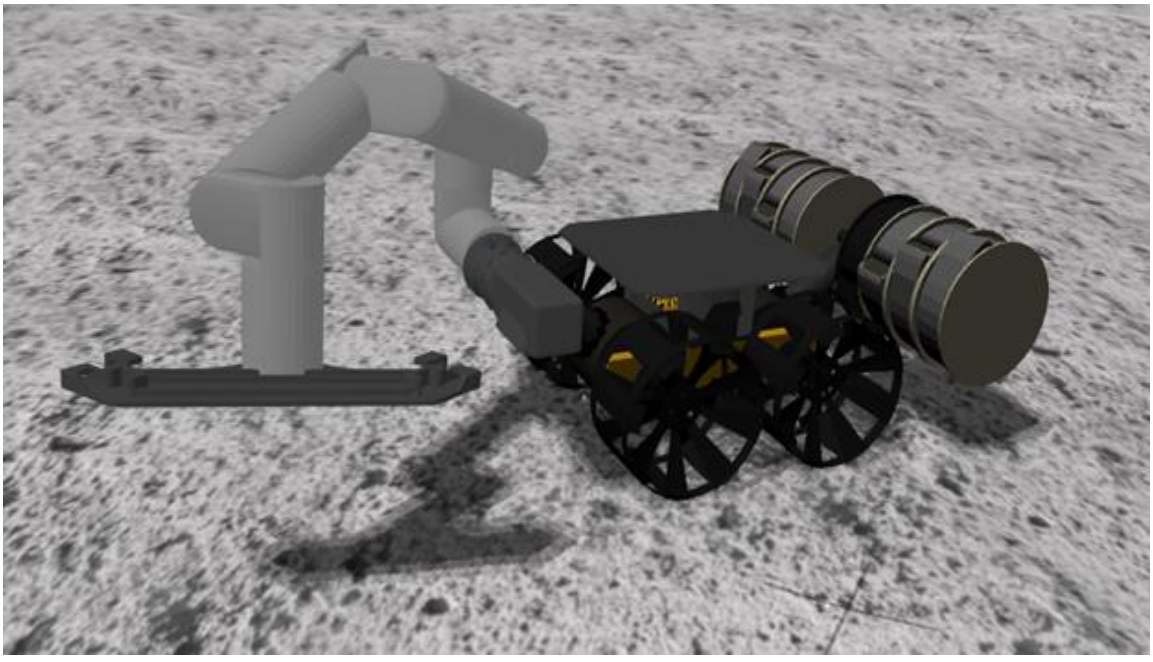


Fig 76. EZ-RASSOR simulation with paver platform and grabbers added.

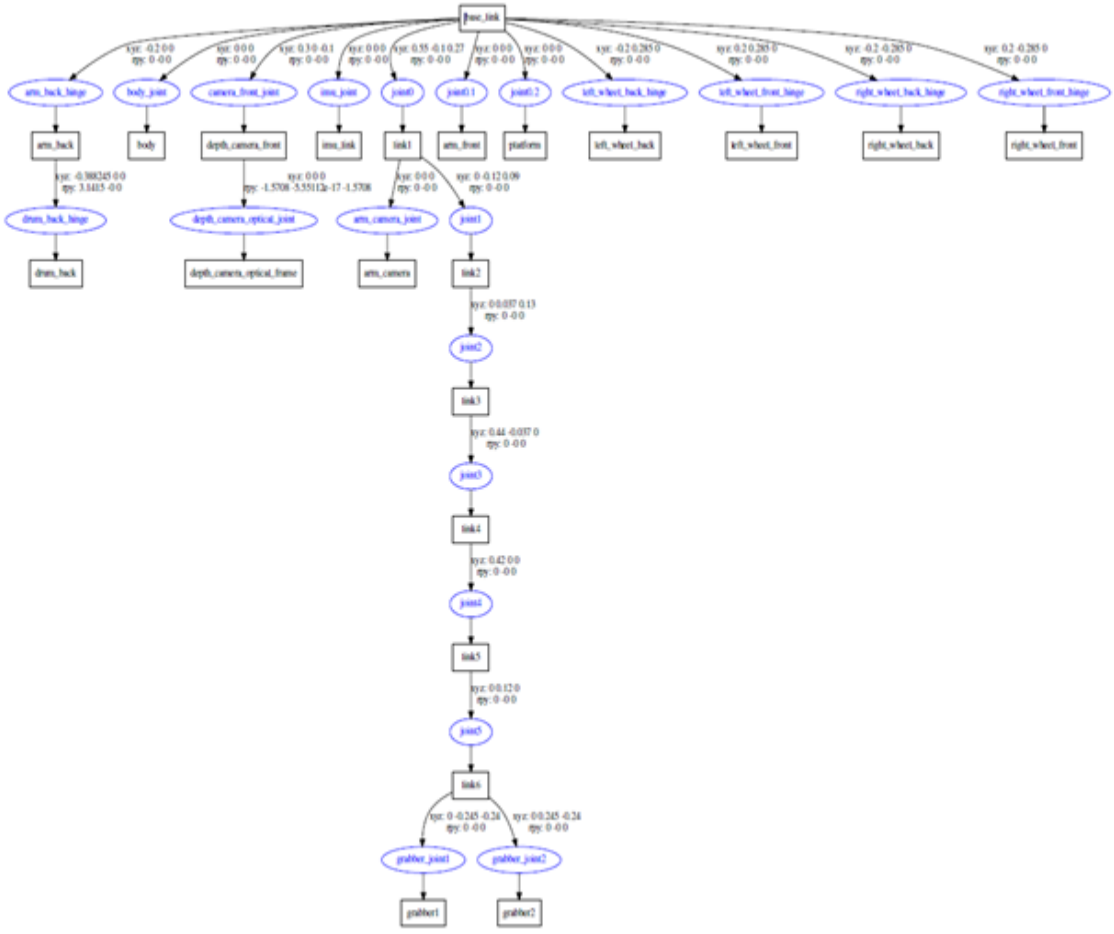


Fig 77. Final graphviz visualization of the robot's architecture.

The model now has all the correct appendages (figure 76 & 77) but is still not complete. The simulation still needs all the appropriate physics applied to the arm and other new appendages. Further analysis of the AutoDesk model will provide insight into the individual arm segment weights. With proper weights for the new appendages and added collisions for the arm segments, we will be very close to a fully functional simulation of the EZ-RASSOR.

Our team decided early on to prioritize progress on implementation of the simulation this semester so that we would have a place to test the robot vision and autonomous controls when they are ready; both are still in research phase, but we plan to make incremental progress fine tuning the simulation in the summer so that come fall, we can hit the ground running with testing. Unless we get access to a physical EZ-RASSOR, the simulation will

be our only method of testing our work so it is imperative that this environment be functional and active before any testing can commence.

Simulation Model Phase 3

The final steps of the simulation rover model completed in the early parts of the fall involved minor tweaks to the link adjustments and the addition of link weights and collisions for all the components. Most links were given collisions mapped to the visual meshes but this caused issues with the addition of the paver causing inconsistent physics. The platform holding the paver on top was given a box collision that worked more consistently in Gazebos environment.

Simulation Camera Integration

Following the model integration with gazebo and basic manual controls using the keyboard; the next major step was to begin development on the visual network and autonomous controller for the arm which needs a camera to use. The real-world designed arm has a camera integrated into it at the base link of the arm so no work will have to be done adding a camera to the arm and will only require programming to access its images using ROS topics. For the simulation however, the camera must be specifically added and placed in the proper position to simulate the actual camera on the arm.

Since the arm will be using the intel real sense D435i camera, it is important to simulate that type of camera to the best of our ability. This means using a simulated camera that not only produces color stereo images like most stereo cameras but can also produce point cloud images using depth sensing which is a prominent feature of the selected real-world camera. To achieve this in the simulation the camera model will be added before functionality is added using the sensor tag along with the Openni Kinect plugin.

The camera model is the first that needs to be implemented into the simulation in order for the functionality to be relevant. This involves adding 2 links and two joints which are shown below in figure 78 in the form of code to create them in the Gazebo simulation.

```
<!-- Arm Camera Added -->
  <link name="depth_camera_arm">
    <collision>
      <origin xyz="0 0 0" rpy="0 0 0"/>
      <geometry>
        <box size="${cameraSize} ${cameraSize} ${cameraSize}"/>
      </geometry>
    </collision>

    <visual>
      <origin xyz="0 0 0" rpy="0 0 0"/>
      <geometry>
        <box size="${cameraSize} ${cameraSize} ${cameraSize}"/>
      </geometry>
      <material name="green"/>
    </visual>

    <inertial>
      <mass value="${cameraMass}" />
      <origin xyz="0 0 0" rpy="0 0 0"/>
      <box_inertia m="${cameraMass}" x="${cameraSize}" y="${cameraSize}" z="${cameraSize}" />
      <inertia ixx="1e-6" ixy="0" ixz="0" iyy="1e-6" iyz="0" izz="1e-6" />
    </inertial>
  </link>
  <joint name="camera_arm_joint" type="fixed">
    <origin xyz=".4 .1 .15" rpy="0 0 0"/>
    <axis xyz="0 1 0" />
    <parent link="base_link"/>
    <child link="depth_camera_arm"/>
  </joint>
  <joint name="depth_camera_arm_optical_joint" type="fixed">
          <origin xyz="0 0 0" rpy="${-pi/2} 0 ${-pi/2}"/>
          <parent link="depth_camera_arm"/>
          <child link="depth_camera_arm_optical_frame"/>
  </joint>
  <link name="depth_camera_arm_optical_frame"/>
```

Fig 78. Code for adding the necessary model components for the camera including: the link depth_camera_arm, the joint camera_front_joint, the joint depth_camera_arm_optical_joint, and the link depth_camera_arm_optical_frame.

Briefly explaining the model, the depth_camera_arm is a link that acts as the main housing for the camera model and used as the middle link between the base frame of the arm and the optical frame of the camera. Most of its values are currently set to default values for testing purposes and will be updated as the full information for the camera is integrated into the system. This link includes collision information to tell the simulation where around the model can collide with other parts of the model, visual information to determine the physical shape and appearance of the visible model in the simulation, and internal information that contains a mass value and inertia values. Connecting the depth_camera_arm link and the base link of the arm is the camera_arm_joint which is a very basic joint that has no motor functionality as of the writing of this document. This joint exists in the case that this decision is changed in the future and the camera is added functionality that would allow for it to rotate and view the backend of the rover. It consists of an origin value, as do almost all joint and link tags, that dictates the origin of the joint in relation to its connecting pieces which are the parent and child tags. The second joint, depth_camera_arm_optical_joint, is very similar to the first joint with no new notable

information other than its connection is connecting the depth_camera_arm link and the depth_camera_arm_optical_frame link. Lastly there is the depth_camera_arm_optical_frame which is just a link that is meant to represent the face of the camera which is used in calculating the camera's location in the simulation when generating images. Without this frame the simulation would not be able to properly orient the origin of the image which would cause off center images that can be detrimental to the success of our visual network.

Moving on to the functionality, there is not a lot to write in code thanks to the resources available to ROS sensors. Figure 79 below shows the code necessary to add functionality to the camera in the simulation. Each part of the functionality is discussed below figure 79 in order to construct a full understanding as to how the component works.

```
<!-- Arm Cameras -->
<gazebo reference="depth_camera_arm">
  <sensor name="depth_camera_arm_camera" type="depth">
    <update_rate>30</update_rate>
    <camera>
      <horizontal_fov>1.29154</horizontal_fov>
      <image>
        <width>640</width>
        <height>480</height>
        <format>R8G8B8</format>
      </image>
      <clip>
        <near>0.105</near>
        <far>10</far>
      </clip>
    </camera>
    <plugin name="depth_camera_arm_controller" filename="libgazebo_ros_openni_kinect.so">
      <baseline>0.2</baseline>
      <alwaysOn>true</alwaysOn>
      <updateRate>0.0</updateRate>
      <cameraName></cameraName>
      <imageTopicName>color/image_raw_arm</imageTopicName>
      <cameraInfoTopicName>color/camera_info_arm</cameraInfoTopicName>
      <depthImageTopicName>depth/image_raw_arm</depthImageTopicName>
      <depthImageInfoTopicName>depth/camera_info_arm</depthImageInfoTopicName>
      <pointCloudTopicName>depth/points_arm</pointCloudTopicName>
      <frameName>/depth_camera_arm_optical_frame</frameName>
      <pointCloudCutoff>0.105</pointCloudCutoff>
      <pointCloudCutoffMax>10</pointCloudCutoffMax>
      <distortionK1>0</distortionK1>
      <distortionK2>0</distortionK2>
      <distortionK3>0</distortionK3>
      <distortionT1>0</distortionT1>
      <distortionT2>0</distortionT2>
      <CxPrime>0</CxPrime>
      <Cx>0</Cx>
      <Cy>0</Cy>
      <focalLength>0</focalLength>
      <hackBaseline>0</hackBaseline>
    </plugin>
  </sensor>
</gazebo>
```

Fig 79. Code for adding the arm camera functionality to the simulation.

The sensor tag is a specific tag used in gazebo models to dictate some form of sensor whether it is a camera, laser, internal motion unit, etc. For our purposes, we will be using the sensor tag's ability to be used to add a basic depth camera. To accomplish this the sensor was added to the ezrassor.gazebo file which is the main file used to generate the model of the rover in the simulation. The sensor tag has a few parameters that must be defined to allow for the tag to perform its intended function.

First of these parameters, outside of the gazebo reference parameter that references a specific link in the model, is the name of the sensor and the sensor type with the name being "depth_camera_arm_camera" (the naming style is to match the style of the other camera names in the simulation) which can be used for uniquely identifying the specific sensor and the type is set to "depth" to denote that the sensor is a depth camera. Next was the update rate which denotes the max number of times per second a new camera image is taken from the gazebo environment; for our camera we have chosen the standard update rate recommended by the gazebo website which is 30 images per second. Following the update rate is the camera's basic information which includes the size/format of the raw images, horizontal field of view, and clip distance which denotes the lower and upper bounds for the distance that the camera can see. These values are all based on the intel real sense D435i specs to simulate the camera as closely as possible.

Finally for the sensor tag is the plugin tag where a shared library known as a plugin can be inserted into the simulation to facilitate integration between the camera and ROS. For our simulation the plugin chosen was the Openni Kinect plugin which is a plugin that was originally designed for use with a Microsoft Kinect camera but has since been expanded to support most depth cameras and is reliably maintained. Similar to the sensor tag, the plugin tag also comes with quite a few parameters that must be understood and defined for the plugin to perform as intended.

Starting with the plugin parameters is the baseline or the distance between the left and right cameras on a stereo camera which is currently set to the default value of 0.2 until the actual camera's baseline can be calculated. Next are a few parameters which are quite straightforward with alwaysOn being set to true meaning the camera is always on, the update rate is set to 0 so that it doesn't throttle the base update rate set in the sensor tag, and the camera name which is left blank since it is not needed for this simulation. Now we reach a few parameters that automatically create topics that the camera will publish for accessing images and camera info for color images, depth images, and point cloud images. These topics can be called on by rospy subscribers to obtain the images/info as they are generated to be used in other parts of the program such as the visual network for the autonomous controller. The frame name is used for viewing the images from the camera in software built for the simulation such as rviz and the pointcloud cutoff and cut off max are

the minimum and maximum distance for point cloud points, respectively. The distortion values of K1-K3 and T1-T2 are used to represent any distortion in the camera and should match the values given in the distortion tag of the camera model which for this method is not needed so all of these values are set to 0. Nearing the end of the plugin parameters are a few more parameters for calibrating the camera including the principal points (CxPrime, cx, and cy) which are set at their default values of 0 so that they are at the center of each image, the focal length of the camera which is also at its default value of 0, and lastly the hackBaseline which should be set to its default value of 0 since any other value will cause a misalignment between the sensor image and the frame that image is supposed to be attached to.

After all of this the camera was ready to be tested to ensure that all of the image/info topics are being properly sent. This was done by using rviz while the simulation was running in the background allowing for the arm camera's optical frame to be selected as well as the topics for the images. The results showed that the color image, depth image, and pointcloud were actively being sent over the defined topics in the plugin tag and were ready for use in the visual network.

Robot Vision Summary

We have shown the research of many different methods of object detection, edge detection, robotic simulation visualizations, etc. We had many different options to choose from for each step of the vision process, and now that we have concluded the research we now know which method to choose.

For the edge detection, we will be using the Canny Edge detection method that uses built in methods from the OpenCV library. This can be done in a few lines of code as we tested this out early on in the semester with the picture of the Rubik's Cube from the Canny edge detection section. We chose the Canny method because it is very well known and used widely in the professional world, giving us enough information to pull from if we ever run into a problem.

For object detection we have decided to use the algorithm and function used in the Object Detection section. It is the easiest and fastest way that we found to detect edges efficiently. This algorithm, in combination with the box-bounding equation will do a great job at locating items that we need the EZ-RASSOR arm to navigate to. This sort of detection will make it a lot easier for the implementation phase of this assignment.

After using Canny to detect edges in the scene and using our object detection algorithm, the data will be fed to the autonomous controller so it can begin to manipulate the joints

and move the arm. As the arm begins to move downwards to pick up the paver, our motion tracker, most likely through the optical flow method, will continue to feed data to the autonomous controller to keep track of how the arm is moving in relation to the surroundings to keep the rover stable and to make any necessary adjustments so the arm can accurately lift the paver. Once the arm has grabbed the paver, the arm will lift the paver and the camera will begin to view the area where the paver will be placed.

At this phase, the Canny algorithm will run again to find edges in the scene, and the object detections will look for if another paver is already placed on the ground. If there is another paver, image data for how the paver is oriented will be fed to the CNN to help the controller determine how it needs to place the paver down. The CNN can most likely be built with keras and trained over the summer on one of our local systems. If there is no paver, a boxed area can be calculated and then fed to the CNN determining where to place the paver. Output from the CNN can be sent to the controller where it will begin to move the arm downward to place the paver. Once again, the motion tracker will keep track of arm speed and direction to make necessary updates for the paver to be properly placed in its target destination.

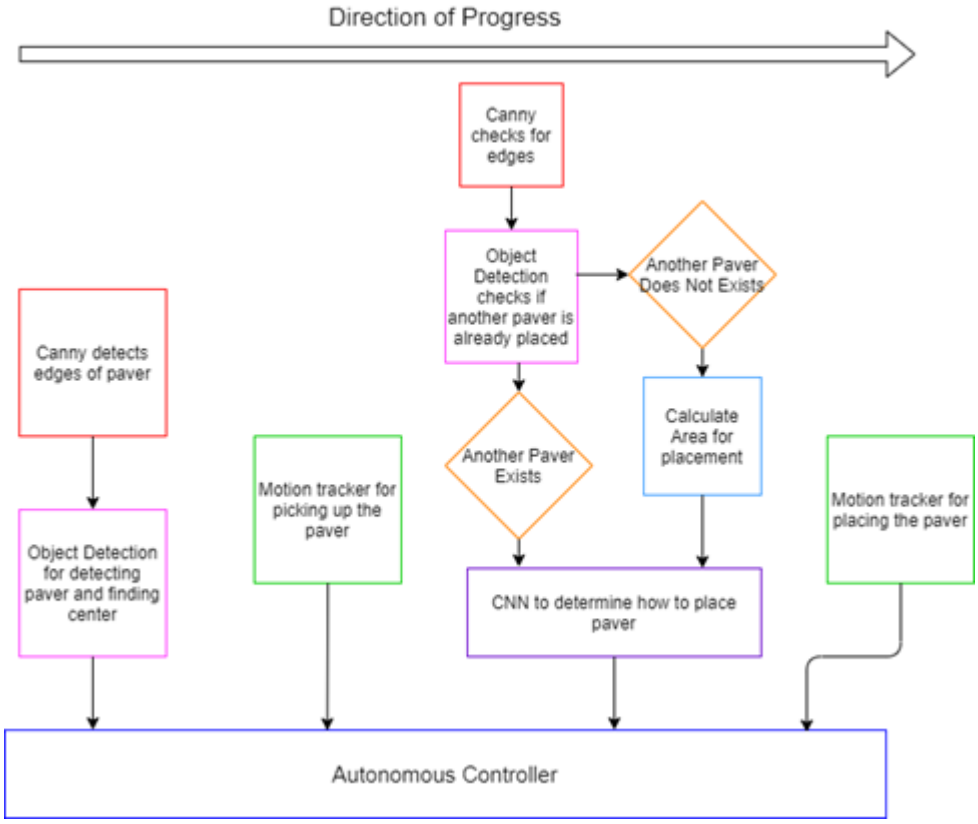


Fig 80. Diagram for how we plan for our visual network to be designed.

Communication

Our design method for communication is to build off the existing architecture as shown below in figure 81 starting with manual controls using the keyboard. Each joint will have two keys mapped for its function as shown in table 5.

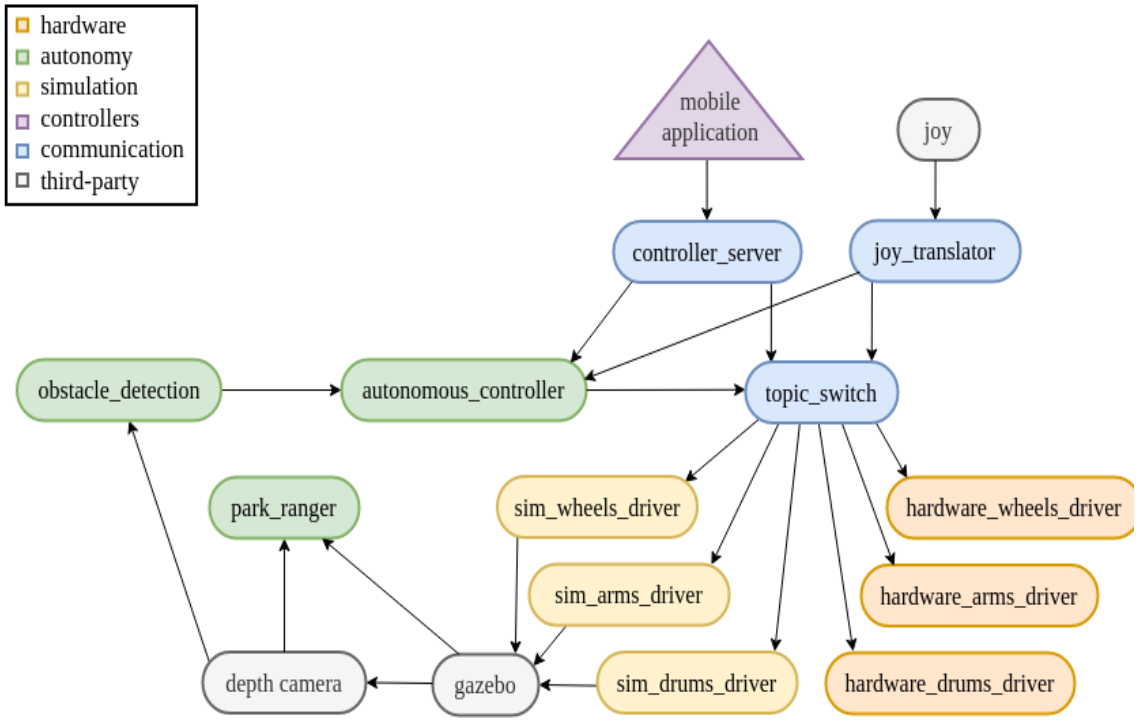


Fig 81. Current Architecture of EZ-RASSORcommunication. [78]

Key	Action
Q	Move First Joint Forwards
A	Move First Joint Backwards

W	Move Second Joint Forwards
S	Move Second Joint Backwards
E	Move Third Joint Forwards
D	Move Third Joint Backwards
R	Move Fourth Joint Forwards
F	Move Fourth Joint Backwards
T	Move Fifth Joint Forwards
G	Move Fifth Joint Backwards
Z	Move Claw First Joint Forwards
X	Move Claw First Joint Backwards
C	Move Claw Second Joint Forwards
V	Move Claw Second Joint Backwards

Table 5. Keyboard controller map for arm.

This mapping will allow us a simple controller for testing that each joint responds and functions properly for basic controls. Whenever a key is pressed, a POST request is sent to the controller_server where it is read and routed to the correct node using a publisher that has a subscriber in the driver node. Once the driver node has the data it can perform the action and return information regarding the progress of the action.

After manual controls work without issue, the autonomous controller can be built that utilizes the visual neural network. These controls will start upon loading the launch file with autonomous movement as an argument. When the controller_server loads, it will start the autonomous controller and begin receiving messages containing the topics and parameters for the next movement generated by the visual neural network and autonomous controller. Once the controller_server has these messages they can be sent to their respective drivers similarly to the method explained in the manual controls.

Should time permit, there is a gamepad app that could have controls for the arm integrated into. The app would follow the same method for communication as the keyboard controller but will need some slight modification to the app itself to send the correct information in the POST request. First buttons for each joint's movement forwards/backwards will have to be added to the graphical user interface. Then adding calls to the controller server to send the necessary data obtained from the arm's functionality buttons to perform actions. Once the POST is received by the controller_server, it will be mapped to the already existing topic switchers made during the keyboard controls.

Fig 82. EZ-RASSOR controller app that arm functionality will be added to. [78]

Also should time permit, there is a joystick controller that allows for more precise movements to be made in relation to a control pad's manual movements and could have arm commands implemented into it. This would require mapping each position on the joy stick to a movement for the arm and using buttons to change which joint is being manipulated. Once these movements are mapped, the joystick controller can send its POST request to the controller_server to proceed with standard manual controls.

Autonomous Controller

With a functioning camera and visual network along with basic functionality to the arm's joint motors; the next key part to be designed is the autonomous controller for the arm. This controller's job will be to facilitate autonomous movement by the rover arm to pick up and place pavers in the intended location/orientation. There are two implementations currently being considered with the first being what we plan to do for sure and the second being more of a stretch goal option/enhancement of the first method should the first method be insufficient.

The first method for the autonomous controller that is intended to be the main method for the controller is by using reverse kinematics to move the arms joints based on the imaging data given by the stereo camera as researched in the "Arm Path Planning using Stereo Vision" section of this document. As images are taken from the camera and fed through the visual network; the visual network will return key pieces of information to allow for the use of reverse kinematics for each joint such as the center of the paver and the center of the end-effector of the arm. Using these positions along with known starting angles, reverse kinematics can be used, along with the arm's DH model depicted in figure 83, on one joint at a time to move the end-effector to a position where, by moving the other joints that haven't moved yet, can reach the designated location for pickup/placing. These functions are very similar to equation 21 in the path planning for the stereo vision section of this document, and show the relations between each joint angle and its positions in the environment space. While this method will certainly not be the quickest, it doesn't need to be too quick since our sponsor said that the method can take a few minutes without causing an issue along with the physics factors that should be considered. Being that this rover arm will be operating in possibly low gravity environments it is important for the autonomous controller to be mindful of its own movement as moving the arm too fast could cause the rover to tilt or rock back and forth which would be detrimental to the overall performance of the rover. For this reason, the autonomous controller will also take input from a motion tracking algorithm and the internal motion sensor (IMU) to ensure that movements being made by the rover in response to the arm movements are considered and used to refine the kinematic calculations for the joint movements.

The second method, which is currently either a stretch goal or enhancement for the first method should the first method be insufficient, is using the DRM algorithm to create roadmaps that can be used to plan the path of the robotic arm in a more dynamic and thoughtful way. Using the first method there is no consideration for things like obstacles in the environment which is fine for the purposes of this implementation since the locations where pavers are to be placed should be free of any obstacles. However, while it is fine for

this specific implementation to ignore obstacles; it is worthwhile to consider the things that this arm may be modified to do in the future that could include working around obstacles. These obstacles could even be human beings some day so having a path planning algorithm that is used in conjunction with the inverse kinematic equations and can be calculated and ran at a fast pace is certainly something that is worth the consideration for this project. This method would involve using the DRM methods explained in the previous sections of the document. First the environment's spatial data would be obtained using the point cloud imaging that the arm camera can produce. This data would be run through a roadmap configuration loop to produce a viable roadmap that has connections between the majority of its configurations. Then obstacles would be added in through mapping the workspace of the arm to the configuration roadmap generated in the step before. With obstacles accounted for, the algorithm could then have its enhancements done using the minimum cut set and p-robustness methods to ensure that narrow spaces in the C_{free} are properly connected and represented. With this the roadmap can be queried for viable paths for the arm to take which would then be fed into the inverse kinematics equations to convert those positions into joint movements for the arm.

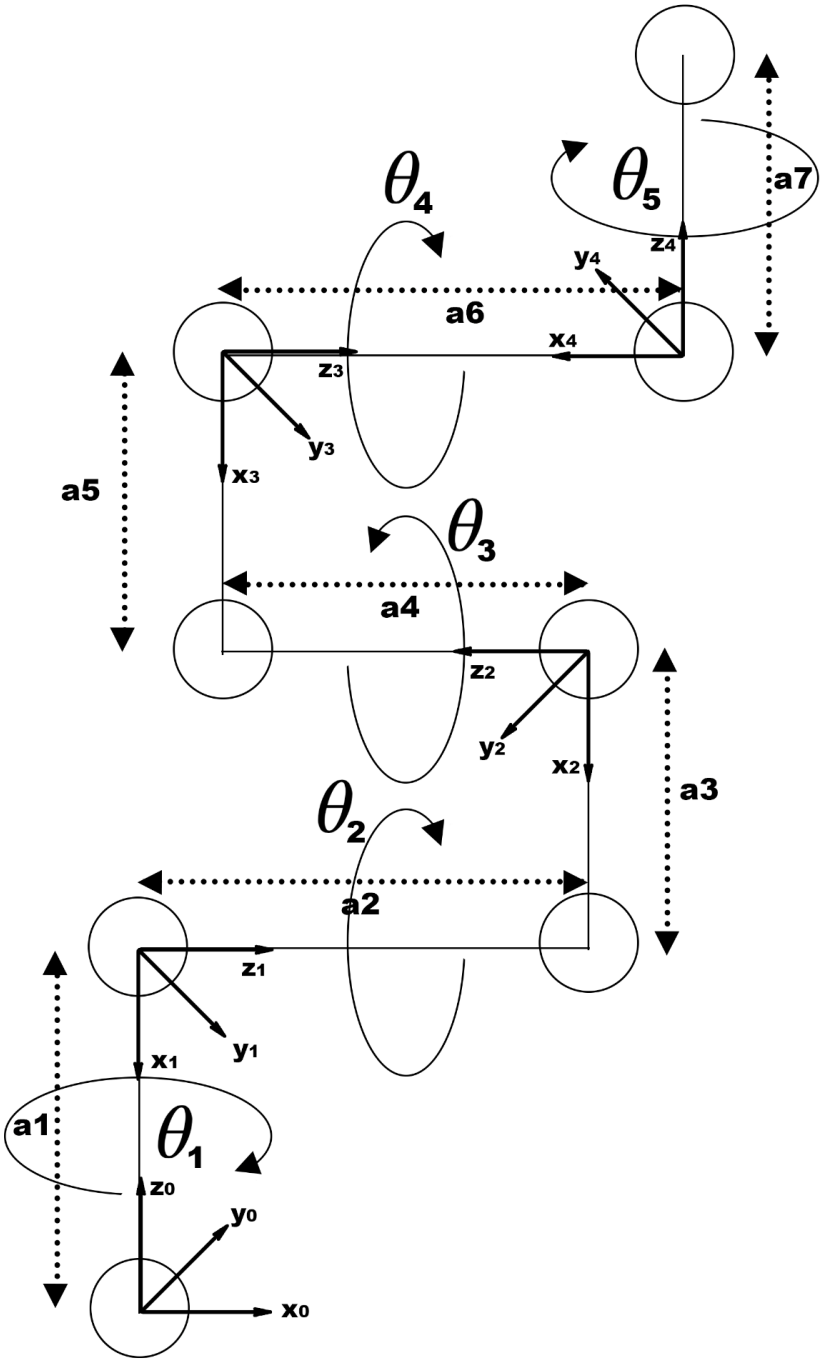


Fig 83. DH model of our arm that has 5-DOF.

Running Tests

Initial Challenges and Difficulties

In order to get most research done, the group has to run tests to determine which methods of object and edge detection works best, or even whether or not it is necessary at all. Having all of this information is good to have at our fingertips, but we won't know for certain until the testing simulation is up and running.

This is the difficult part about this step of the project. Being stalled out due to waiting on a certain project to work, and getting through the difficulties of running the specific programs on our own personal computers. Operating system functionality has been an issue for some of the members in the group and we are still trying to find a way around that.

Simulation

Future testing for the majority of this project will take place in the Gazebo simulation environment. For testing the simulation itself, this also occurred in Gazebo as well, but was compared to the expected function of the EZ-RASSORas it would in the real world. The displayed Gazebo simulation was compared to given source material and described functionality. If the rover did not look right or did not move correctly, the design code was edited, and the simulation was re-run to see changes. This process was repeated until the simulation produced the desired movement and look of the rover.

Keyboard controls for the arm movement were primarily added for testing purposes. In the final design, moving the arm by rotating it's 5 joints with different keys by hand is not feasible and the arm movement will be nearly or entirely automated. Key movements of the arm allowed for testing of the movement and can also give indications about necessary precautions for the autonomous movement like max rotational speed to avoid unbalancing the rover.

Although building the simulation is not actually required in this semester's deliverables, our group has decided that it is essential to get a head start on testing before the semester ended so we are not bombarded with testing and integration in the fall. Since we don't have access to a physical EZ-RASSOR robot, we have been working effortlessly on trying to build a fully functional simulation by the end of the semester to hit the ground running at the start of Senior Design 2. As stated in other sections of this paper, since we have the whole summer to do additional research, we plan on taking advantage of this time by perfecting the simulation and ensuring that there are minimal hiccups along the way due to there being no time constraints over the summer months. The only downside is that this

project might be on the back burner over the summer for some of our group members as a good majority of us have summer internships/ full time jobs to have to give most of our time and effort towards.

Physical Robot Implementation

While we do not currently have a working physical EZ-RASSOR robot working with the robotic arm attached, we do plan on being able to implement our program with it in person by the next fall semester. Since our physical senior design class is set to be in person, we plan on being able to go to the Florida Space Institute lab, the lab that is sponsoring this project, and work with the robot hands on. Currently the arm is being prototyped in Germany by another EZ-RASSOR team member, but the whole team is hoping for the final design to be done prior to August.

Using the methods and programs that we have been researching and the computer simulated tests. We hope to have a smooth transition from the simulation to real-world implementation. Our group members have done a fantastic job at modeling the EZ-RASSOR robot and robotic arm. If all things go as planned, the transition in the fall should be effortless, barring any unexpected disasters.

Hardware

Intel RealSense D435i

The camera that our group will be using in this project is the Intel RealSense D435i stereo vision camera. The use of stereo vision cameras make this project much easier compared to if we were to implement these programs using a regular mono-vision camera. The major difference that makes Stereo vision cameras can detect every object in the frame compared to mono-vision cameras that only detect objects that it has been trained to detect [79]. This obviously gives us an extreme advantage in not only time in the implementation stage of this project, but also it could give us a faster runtime when running the actual edge and object detection software.

Another major advantage for the use of stereo vision cameras is the fact that it implements the visual field of two cameras simultaneously rather than one [80]. This makes the camera have vision similar to human eye sight, which gives the robot a sense of depth that a regular camera just doesn’t have. It's similar to the difference in depth perception that you have when you close one eye. Having two fields of vision to bring together will make the EZ-RASSOR robotic arm that much more efficient and accurate. An example image of how the cameras operate is shown below [80].

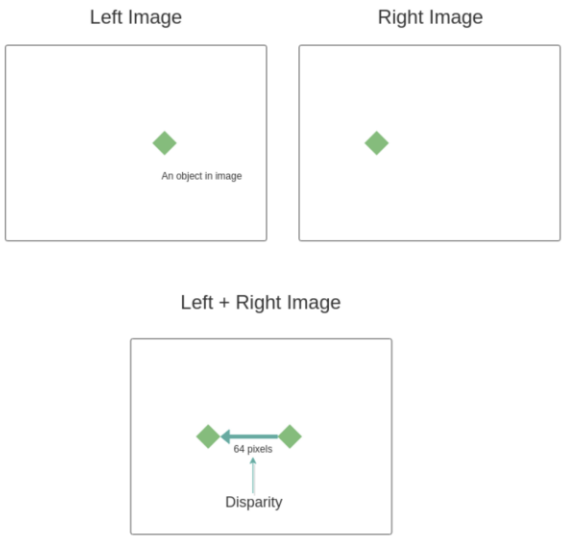


Fig 84. Depiction of disparity in stereo image.

As you can see, the combination of the two images from the different cameras form a new image, the image on the bottom shows the difference in position between the two cameras. By combining these two images, we get a better idea of where the object in the main image is.

Arduino Mega 2560 REV3

Arduino is an open source electronic platform that uses boards and software to convert some sort of input into whatever output desired. The board is programmable using the Arduino programming language.

This is a microcontroller board with 54 input and output pins, 16 analog inputs, 4 universal asynchronous receiver-transmitters (UART), a single type A B USB port (5V), a power jack, a 16 MHz crystal oscillator, and ICSP header, and a reset button [81]. This particular Arduino board is designed for projects that use more RAM and sketch memory, making it ideal for robotics projects such as ours.

Like many other Arduino boards, programming the Mega 2560 requires the Arduino integrated development environment (IDE). To get started, the board must be plugged into a PC using an A B USB cable (USB printer cable). If the connection is successful, a green LED should turn on labeled PWR. This connection provides power to the board and allows programming. Next, specify the board and port in the “Tools” tab of the IDE. Now to upload any sketches, just use the upload button in the IDE. A few seconds after the upload, the pin 13 LED will blink in orange indicating the board is running.

Fig 85. A picture of the Arduino Mega 2560 we will use.

Nvidia Jetson Nano Developer Kit

Our project will need some sort of way to process the image from the Intel RealSense Depth Camera D435i to let the arm see what it must interact with. This is where the Jetson Nano comes in.

This device is an AI computer that we will use for the visual neural networks. The developer kit is a non-production Jetson module attached to a reference board. It is used to create and test software, but isn't intended for production use. For the finalized product, a Jetson module will be required.

The kit has a 40 pin expansion header, a micro-USB (5V), 1 gigabyte ethernet port, 4 type A USB 3 ports, an HDMI and DP port, a barrel jack, an M.2 key E connector for wireless networking, and a micro SD card slot. A minimum 16 gigabyte UHS-1 micro SD card and a 5V micro-USB power supply is required to operate the developer kit. To start, simply flash a system image to the SD card and insert it into the Nano. Then insert the monitor display, the mouse and keyboard,the ethernet cable, and the micro-USB power supply. Alternative configurations are possible such as powering the computer with a DC power adapter using the barrel jack or using a wireless network. After inserting the power source, the unit will power on automatically [82].

Fig 86. A picture of the Nvidia Jetson Nano developer kit we will use for testing.

RepRap Arduino Mega Polulu Shield 1.4

The RepRap Arduino Mega Polulu Shield (RAMPS) 1.4 is a low-cost 3D controller board to be used as a shield for the Arduino Mega board. It is essentially a board that serves as the interface between the Arduino Mega and any electronic devices attached to the Ramps board.

The RAMPS board has the following features: 5 stepper motor driver outputs (3 Axis + 2 Extruders, with dual motor driven Z-Axis), 3 PWM MOSFET's outputs, 3 thermistor inputs, 6 limit switch inputs, I2C and SPI outputs, and room for additional expansion boards.

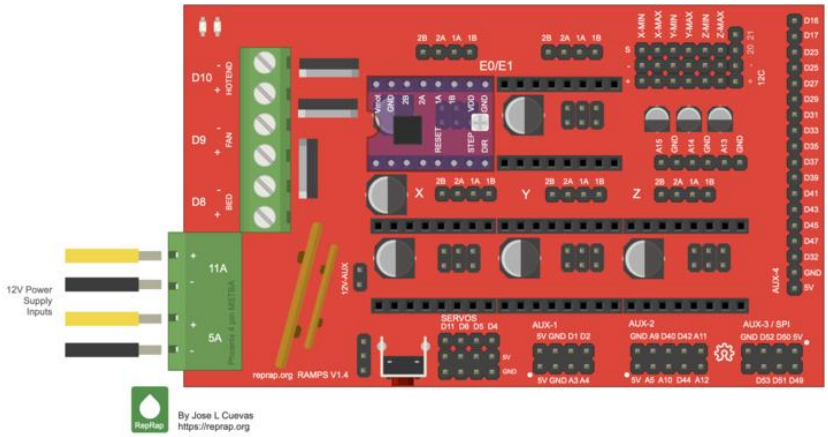


Fig 87. A diagram of the RAMPS board that will contain the arm's stepper motors and limit switches.

Integrating Jetson Nano Developer Kit with RealSense Camera

To establish a visual network with Jetson, synchronization between the Jetson Nano developer kit and the RealSense camera is required.

For this process, we will use the scripts provided by another GitHub repository. First, clone the mentioned repository and switch into its directory.

$ git clone https://github.com/JetsonHacksNano/installLibrealsense

$ cd installLibrealsense

Then simply run the installation script:

$ ./installLibrealsense.sh

To run Intel RealSense Viewer application, just run the following command in the directory:

$ realsense-viewer

Integrating the Jetson Nano Developer Kit with the Arduino Mega 2560

Sending Data from the Mega 2560 to the Jetson

The first step is to program the Mega 2560 using the Arduino IDE. It is possible to install the Arduino IDE on the Jetson itself to make programming the board more convenient, but it isn't required.

Next, connect the Arduino to the Jetson Nano using a USB connection.

Now if pyserial hasn't already been installed on the Nano, then do so by running this command in the terminal:

$ sudo -H pip3 install pyserial

Then we can create a script in python that establishes a connection with the Mega 2560 using the USB. It is critical that we import serial and time. The line that establishes the communications should look something like this:

arduino = serial.Serial('/dev/ttyACM0', 115200, timeout=1)

The first parameter is the USB connection, the second is the baud rate (make sure it is the same as the rate specified by the Arduino program), and the third parameter is the connection timeout time.

Implementation

This section of the document is dedicated to documenting notable changes made during the actual implementation of the project.

Visual Network

Our final implementation for the visual network did not involve all of the methods researched and mentioned in the robot vision summary section. Far less information was needed for our autonomous controller as there were fewer variables and scenarios to consider than we had anticipated. When picking up pavers, we were able to assume that another rover would place pavers on the back in the same orientation every time, so the visual network would not be needed for picking up the paver. We also would not be placing multiple pavers during one trip. The rover would place a paver, go back to receive another, and then drive to the previously placed paver and place the next. This meant only one paver would need to be detected at one time, and we wouldn't need the visual network to determine where the rover needed to drive to next.

Since continuous adjustments to path planning and speed of the arm would not be necessary when the arm began to move, the motion tracking was no longer needed. We also did not need the canny edge detection as the CNN would be enough for detecting the paver and finding the edges by producing a bounding box around the paver. So the final design for our visual network was reduced to just a CNN that would output a bounding box on the paver which could then be used for getting coordinates on where the next paver needed to be placed. The way we achieved this came about in two versions.

Version One

The first version of our implementation involved training a pre-built CNN model that would output the coordinates of the corners to produce the 2D bounding box, and then using the features from that box we could calculate a 3D bounding box to give us the depth of the paver. Using a pre-built model allowed us to avoid designing our own custom model and instead use one of the many options that are available and proven to be highly accurate. We were able to do this by using tensorflow and keras, which are python

libraries that make setting up training and testing of CNN models simple, and they also give you access to the pre-built models previously mentioned. Using Google Colab, we were also able to efficiently train the model, as Colab gives you access to high-end GPU's Google has access too, speeding up the training phase to around one to five minutes instead of the hours it can often take if you train on your local GPU.

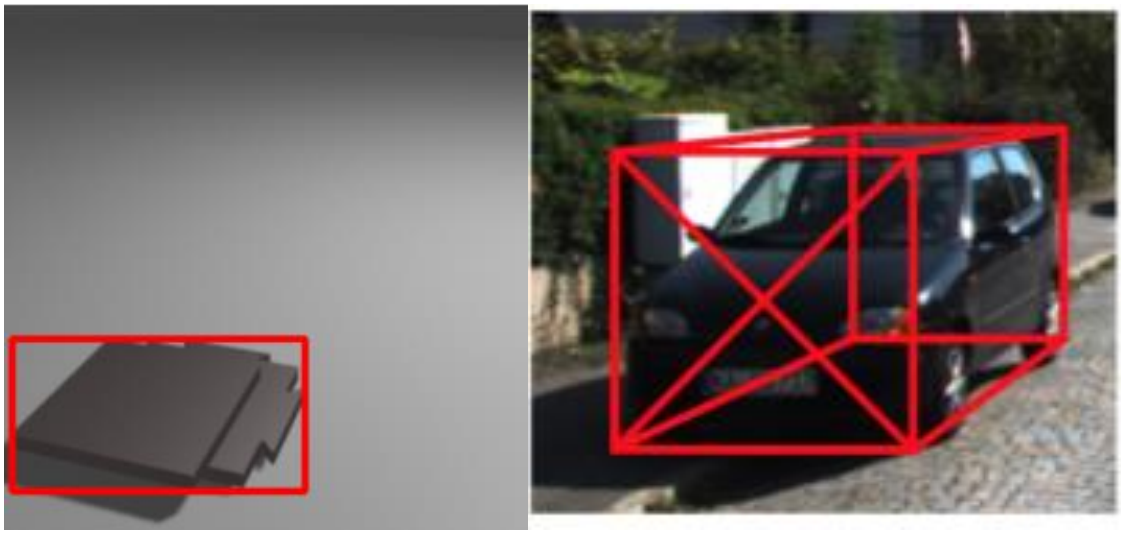


Fig 88. Images showing 2D bounding box made using label data and an example of what the 3D bounding box should look like.

For training, we went through several iterations of varying amounts of testing images that included images with a paver in view and images without a paver in view. Along with the images, we also included labels for the start x, start y, end x and end y coordinates that would produce the correct bounding box for each of the images. Several models were used, but the final model that produced the best results was the VGG16 model, using four-thousand eight-hundred images with a split of eighty percent training data and twenty percent testing data. These results were not up to the standards we needed for our final implementation, so we began looking for other methods.

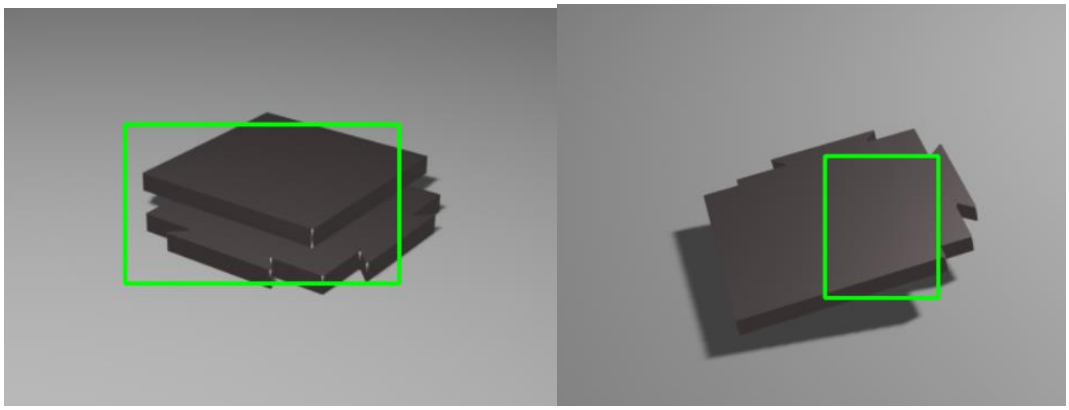


Fig 89. Examples of best results from training our VGG16 model with 4,800 images split 80% training, 20% testing, 10 epochs, batch size of 15 and a learning rate of 0.0001.

Version Two

When searching for solutions to our issues, we found that instead of training a model ourselves, we could actually use a function provided by OpenCV that takes the frozen weights and configuration that you pass it to load a pre-trained model, and then we could pass the image taken from the camera on the rover into the model which would produce a multidimensional array that included all of the objects found in the image, the confidence scores for them being an object, and the coordinates needed to create a bounding box around the object. Since this finds all of the objects it detects in the scene, this method is typically used by setting a threshold and only displaying the boxes of the objects whose confidence score surpasses that threshold, but this resulted in many boxes being drawn in our final image.

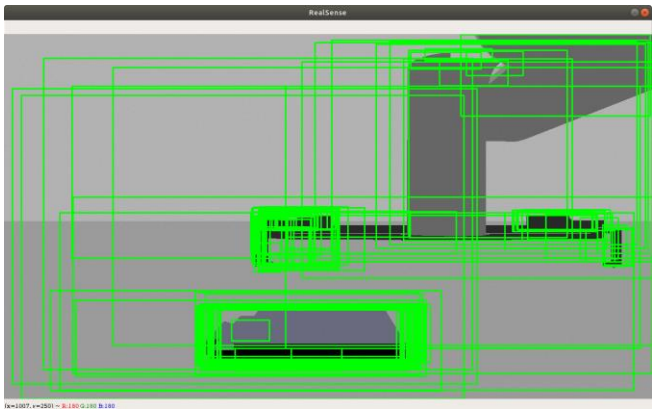


Fig 90.. Result of passing image from rovers camera into model and drawing all boxes that exceeded a score 0.01

Increasing the threshold would obviously get rid of some of these boxes, but another aspect to consider was that we only needed one object to be detected, which was the paver, so instead of drawing all boxes with a score above the threshold, we can simply draw the first that is above the threshold. This did give us what we were looking for with only a single box being produced, but when the paver was located at certain positions in the image, no boxes would be produced. The issue was that in these positions, the model was producing scores that were significantly lower than the threshold. We then removed checking for the threshold, and instead draw the box of the object with the maximum score.

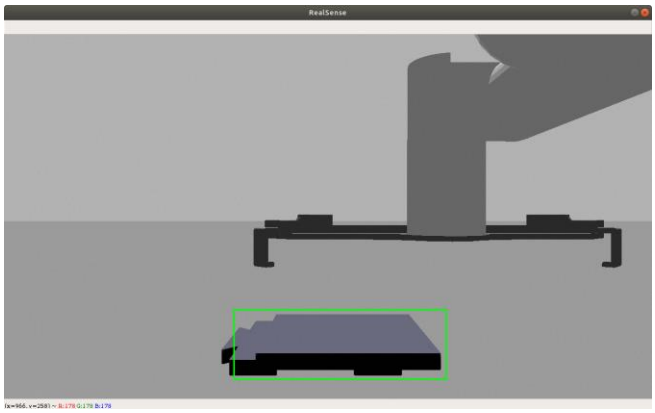


Fig 91. Result of drawing bounding box after removing threshold check

Now we needed to get the depth of the paver since we're dealing with 3D space. Similar to how we found a simpler method for producing the 2D bounding box, we also found a simpler method for getting the depth than producing a 3D bounding box. Since we are using the realsense stereo camera, we are able to get a depth image along with a color image. This provides us values in millimeters for any parts of the image we need. Using the coordinates given for the 2D bounding box, we can calculate the center of the object, and use the x and y coordinate of the center to find the depth at that pixel.

Fig 92. Result of drawing 2D bounding box, calculating center and getting depth at that pixel position.

Finally, we need to convert the coordinates we have from pixel coordinates to physical cartesian coordinates. The realsense API provides a function that uses the camera's information such as the height and width, and the pixel coordinates you pass to the function, and then deprojects those coordinates to give you physical coordinates. This gives us the x, y and z coordinates needed to pass to the autonomous controller, which then using the known dimensions of the paver can calculate where the next paver needs to be placed.

Autonomous Arm Control

For the final implementation of the arm's autonomous controls inverse kinematics were used through the moveit framework for ROS. This framework is dedicated to facilitating the movement of robotic manipulators through the use of inverse kinematics mapped to the design of the manipulator/arm. Using the moveit setup assistant and a URDF file of the manipulator to be used, a moveit configuration ROS package for that manipulator is created that contains aspects such as groups of joints that should move together, the defining of the end-effector, and even integration with sensors on the robot to take in information from the environment directly to moveit's planning scene. Speaking of the planning scene, moveit was chosen for the robust features that the framework includes such as the planning scene mentioned above that uses the visual information obtained by sensors to populate the planning scene with any obstacles that could hinder the arm's movement. Utilizing this planning scene ensures the arm considers collisions from all objects around it during the planning of its trajectory. Another useful control given by the moveit framework is fine control over the arm's speed in a dynamic manner. For the placement of our pavers on locations such as the moon where gravity is much lower than here on earth, having this control over the speed of the arm ensures that tipping of the rover when carrying a paver using the arm can be avoided as well as the speed increased when a paver is not being held. These are just some of the aspects of the moveit library that allowed for the autonomous control of our arm; some others not mentioned are pre-planned pose goals, as well as easy access to dynamic pose goals for either the manipulator or each joint.

Controller Integration

Halfway through the semester the group ended up integrating the Robotic Arm into the existing EZ-RASSOR controller. We did so by using the React Native framework in Javascript to make a simple user interface for users and students to interact with. Because this is an educational project, the controller was meant to have a very simple and straightforward layout so you can operate the robot with little to zero knowledge.

The controller interface consists of two D-PADs that control specific joints. You will see that the left D-PAD controls the main joint in whichever direction you choose, the middle D-PAD controls the joint at the top of the grabber, and there are buttons to rotate the grabber, as well as open and close the grabber itself.

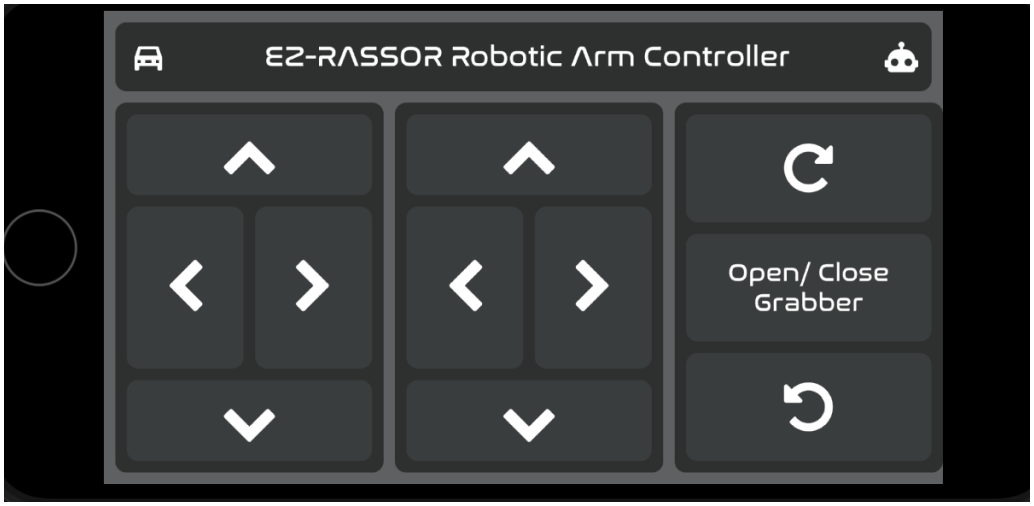


Fig 93. Final UI for Robotic Arm Controller

As you can see there is a Robot in the top right. If a user clicks that button, they would have three commands to choose from that autonomously moves the arm to specific locations to do specific things. There are 3 functions, the first being pick up paver which navigates the arm to the back of the Paver and prepares to pick up a paver from the pay load. It will go down, retrieve the paver and prepare to place. The Place Paver command will place the paver in the spot that the virtual network sees fit. The Return Home function will return the arm to its initial resting position.

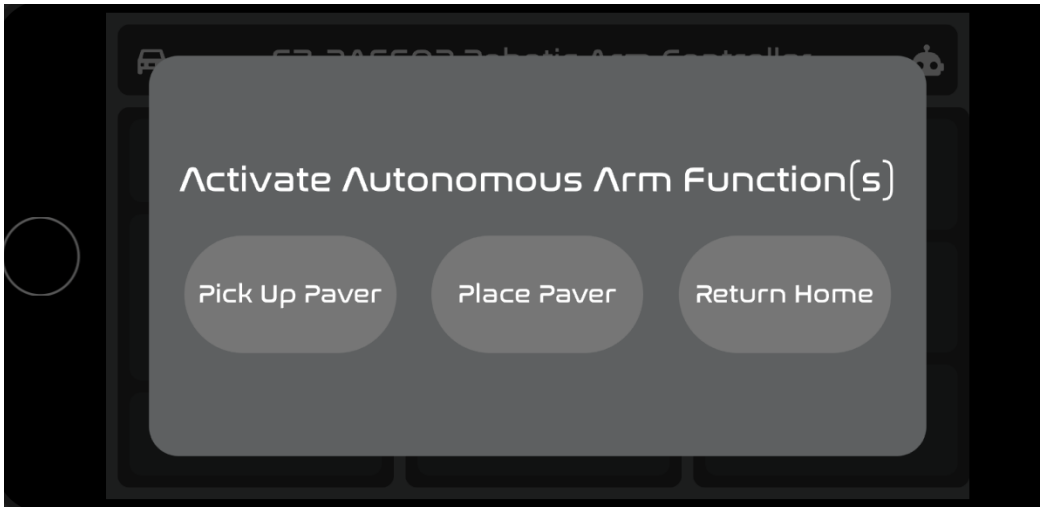


Fig 94. Autonomous Arm Functions

Facilities and Equipment

Working with NASA and the Florida Space Institute (FSI) there are certainly many resources that can be advantageous for the group to use. Unfortunately, due to the covid pandemic those resources for the time being have been severely limited. While this has not overly hindered our design, we are hopeful that come fall more of their resources will become available since doing work on a real-world rover may require things that are difficult for students to obtain. Granted the plan to work on the real-world rover will only come after the implementation of the simulation rover.

One such piece of equipment that is currently being discussed about acquiring is a 3D printer. The physical rover that this team is working on, the mini-RASSOR, has a small model that can be created using a 3D printer for the model parts along with some small machinery components such as bearings and motors. This modeling also extends to the arm and paver that will be attached to the rover so having access to a cheap and easy to test physical version of the arm, rover, and paver will be crucial to developing an implementation that can be incorporated with a more expensive and durable model. Right now, the team is discussing possibly using one of the sponsor's older 3D printers to use for this purpose; should this not come to fruition we will likely seek a grant from the space institute as recommended by the sponsor to handle the large cost of such equipment.

Fig 95. Example of 3D printed bearing for the arm.

Another important piece of equipment that would be of huge help to the project is stronger computers to facilitate better testing with the simulation. For all of the members there have been performance issues that have had to be dealt with so far by trying to limit the work done by the simulation. In a long-term project like this these fixes only curb the issue rather than fix it along with having to spend time that could be dedicated to other parts of the project. With all of these reasons it is being considered by a few members to consider investing in a permanent Linux machine that is stronger than the current ones being used since some members currently only have access to a duel boot Linux/Windows machine.

Project Milestones

- February 4 – March 1: Setting up communication with all necessary parties, preparing Github, and researching/experimenting with ROS, Python, C++, Robot Vision, Board Integration, and Gazebo/Blender (complete)
- February 15: TA check in 1 (complete)
- February 16: Meet with the creator of the arm to learn the full specifications (complete)
- February 18: Initial project design document due (complete)
- February 19: Meet with the previous senior design teams to plan integrations with their systems as well as discuss our plans for the arm software, Meet with NASA swampworks (complete)
- March 5: Begin blender model/controls of arm for simulation (complete)
- March 11: Project status and doc review with professor (complete)
- March 20: Finalize design for blender model of arm and integration into simulation (complete)
- March 21: Finalize basic arm controls for simulation and integration into simulation (complete)
- April 8: Finalize visual network/autonomous controller design (complete)

- April 9: Finalize board controller design(complete)
- April 21: Finalize design document (complete)
- April 28: Final design document due (complete)
- May 5 – August 22: Summer Break
- May 7: Begin visual network implementation for simulation environment (in progress)
- June 20: Midsummer update with sponsor

- July 16: Finalize integration of visual network for stereo camera in simulation
- July 30: Finalize integration of controller for autonomous arm control in simulation
- August 21: Extensive simulation testing to ensure software is ready to begin real world implementation
- August 23: End of summer update with sponsor
- August 27: Begin integration to real world rover
- September 3: Controller for Arduino/Nano integrated
- September 10: Integrate basic arm functions into rover
- September 24: Integrate visual network for camera
- September 30: Integrate autonomous control for arm
- October 1 – November 26: Testing/Bug fixing for rover, gamepad integration if possible
- October 15: Critical design review
- November 26: Final project complete

Challenges

There have been multiple challenges that the group has come across since the beginning of this project. With major projects and research papers like this there will more often than not be more problems than solutions. With every mistake there is an opportunity to learn, and we have had our fair share of mistakes. This section will be a running list of the mistakes and challenges that each member of the group has encountered so far.

Austin Dunlop

Simulation design faced two major challenges: simulation speed and joint calibration. It was found that the simulation, even in a full Linux environment, was taking too long to load. It was resolved that the use of unnecessarily high-resolution models for the EZ-RASSOR's arm were causing the high loading time. Replacing these models with new ones added a new task/challenge to the simulation design process.

Another challenge for the simulation design process was properly calibrating the joints. The simulation was scaled to a different size and was to include elements developed by different groups but required a unified and synchronous movement of all these elements. Calibrating these movements required analysis of EZ-RASSORelements in blender, AutoDesk, ROS, and Gazebo. These measurements were all standardized and documented (fig.19 &20), then implemented in the URDF.

Cooper Urich

So far, my main focus has been researching different edge and object detection methods. Although there are numerous research papers and a plentiful amount of information on these subjects, the most challenging part of my research has been comparing the different speeds of the edge detection methods. It seemed like the more accurate the method was, the longer it took to compute. With a project like this one, speed and accuracy are both very important as you don't want computations to take too long, but you also don't want any faulty edges. Also what is so challenging is that without having everything up and running, it is hard to tell which edge detection method would be the best.

Another major challenge and setback that I personally have encountered is figuring out how to install the Linux operating system onto my computer. Previous EZ-RASSOR groups advised us to Dual-Boot instead of running Linux on a virtual machine. Personally, I ran into a major issue with trying to Dual-Boot my system, as I had made a mistake downloading Ubuntu Desktop (a program that runs Linux) and accidentally overwrote my

native Apple Macbook OSX operating system. This caused me to end up buying a new Desktop Computer to avoid the issue, so it ended up being a fairly costly mistake.

Since this project is mostly going to be written in python, it has taken a lot of research just on how python is written syntactically. Because I have not taken any python courses before, it is slightly intimidating to go from no experience to having to build an entire robotic arm purely written in a language you barely know. Thankfully, there are hours upon hours of tutorial videos online to learn. It also helps that the python's syntax is very easy to pick up, being similar to other object oriented programming languages.

A challenge that I am beginning to feel half way through this project is just how hard it is to get 30 pages on a specific research project. Although this project is by far the most detailed project I have done to date, I am beginning to have trouble finding enough research in my specific field to reach the minimum page requirement. That is one of the most challenging aspects of this project in my opinion, because the actual research and implementation is not technically that difficult, but it is taking the time to sit down and write it all out that is the most difficult.

Knowing what to research without overstepping on my teammates research topics was always a struggle for me during this research paper. There have been times where I wanted to get into another topic for research but didn't know if another member had already done the research and instead of jumping right into the research I had to ask the other members and wait for a response. This caused some significant delays in writing, as I couldn't be as efficient as I wanted.

Luca Gigliobianco

One of the biggest challenges present at the moment is waiting to design the joint controllers. The reason I have to wait until I get to dive into more technical aspects of the project is that the arm model, camera, visual network, and autonomous controller must all be made/acquired in the simulation. Once these are mostly complete, I will be able to work on the arm controls. The same situation applies when testing must be done in the real world. Although this should be faster since we have preparation from the simulated environment. I am also in charge of connecting the Arduino board and the Nvidia computer when testing in person. I also must wait to do this until we are finished working with the simulations.

An additional hurdle I encountered was difficult setting up a dual boot with Ubuntu 18.04 LTS and Windows 10 on my desktop. I successfully partitioned the drive and flashed the system onto a flash drive. However, whenever I tried to install Ubuntu on the partition, the installation process kept on failing. I found out that this was a problem with my

motherboard, and that during the installation process the Ubuntu system installer would only accept input from USB 3.0 ports. This means that I needed 3 USB 3.0 ports available, one for the flash drive, and two for the mouse and keyboard. Unfortunately, My computer only has two USB 3.0 ports, so installation was not possible on this machine. To get around this, I simply did the same thing on my laptop without any issues.

This pivots into another concern I had: hardware requirements for the simulated environment. My laptop is an Inspiron 5570 and it has an Intel Core i5 (8th gen) 8250U, 8 GB of RAM, and integrated graphics. These hardware specifications could be heavily strained by the use of an environment like Gazebo. I have done some testing and found that loading models in Gazebo is slow, but manageable. I am concerned for future simulations that will have more resource intensive requirements.

Finally, the other challenges come from lack of experience and knowledge in certain topics. I will have to learn how to use the Arduino IDE and how to use sketches in it, familiarize myself with the python libraries used, and get a good grasp of how Gazebo operates among other things. This obstacle is the most trivial since this can be overcome with practice and research.

Christopher Jackson

The biggest challenge for when starting this project was learning all the information needed for understanding ROS. While I had heard of ROS prior to the project, I had no experience using it, so concepts such as topics and subscriptions and launch files were hard for me to grasp at first. I spent a decent amount of time at the beginning of the project doing research on the ROS website, looking into the tutorials they provide to help learn each the beginner topics that could get me started, but there is a lot of information you need to go through before you start to have an idea of how everything works. Luckily, my team was able to help fill in some of the gaps for where I was struggling.

Another challenge I faced related to ROS was setting up the environment for running all the necessary software. I already had a virtual machine with Ubuntu 20.0 installed, so initially I thought I had most of the work done already. After looking into some of the ROS documentation, I downloaded ROS Noetic into my virtual machine, only to realize afterwards that the previous teams used ROS Melodic, so I needed to reinstall ROS with the correct version. Unfortunately, ROS Melodic does not support Ubuntu 20.0, so this meant I had to completely start over installing all that I needed. Thankfully, right at this time we met with one of the previous teams who informed us that dual booting was a much better choice for the project than using a virtual machine, so I fortunately avoided once again downloading the virtual machine and ROS, only to have to restart. The dual booting

experience for me was not as difficult as it was for some of my other team members, but it did take quite a while to set up, and in the end after some alterations to my pc build, I ended having to repeat the process again.

The main and most obvious challenge was related to the research for the visual network. Although I did have some prior knowledge with taking Intro to AI in the Fall 2020 semester, the information I obtained from that course was not quite enough to give us everything we needed for this project. Taking robot vision alongside this course did help tremendously in aiding in my research, as many of the topics we learned about helped me have a starting point or an idea of what further research I needed to do. Even with this help though, concepts in robot and computer vision are by no means simple. It took many hours of researching scholarly sources and articles to truly understand in depth each topic, and even more time needed to be spent determining whether what I was researching would be useful for us, and if it was, how it would fit into the overall scheme of our visual network. The time spent was well worth it, however, as I feel very confident in the work I and my partners have put into our research.

Robert Forristall

As with every project there will be challenges and hurdles that must be overcome in order to see the completion of the project's goals. This project is no different with each member having their own challenges to face in regards to the designing and minor implementation that has been done prior to the completion of this document. Some of my personal challenges are listed below and discussed regarding how they impacted the work that I completed and their solutions if one was found.

The biggest challenge for me was certainly the difficulty of being a project manager on a project where I have little experience both in the project topic, for this instance was robotics, and how to manage a project of this scope. It was hard at times and felt overwhelming with all the expectations that I felt placed on me that were not the fault of anyone but myself. Having low self-confidence made this task difficult with things like having to manage communications with many individuals from teammates to people that are associated to the project, maintain an organized list of milestones and goals for the project without overestimating or underestimating the feasibility of said goals, along with delegating out roles which is especially hard for me because I don't like having to tell people what to do even if it may be necessary to keep the project moving slowly. While at first I felt overwhelmed, with time and a good and supportive team I have grown to feel more confidence in my ability to be a good project manager for not only myself but for the sake of my fellow group mates.

Following my own personal struggles comes my struggles with physical hardware that limited my productivity heavily. Both of my current computers, my Windows laptop and Linux desktop, are currently quite a few years old and were relatively inexpensive machines even when they were new. These older computers have had countless issues loading and managing programs such as my laptop having issues with even Microsoft word at times being slow and difficult to use effectively and my desktop trying to load the very computationally intensive simulation environment with rover. Some of these challenges have since been dealt with to certain degrees like creating a lower resolution model for the arm in the simulation to facilitate better performance during testing but, there are still complications that can be detrimental to the constant progress needed for this project. My goal is to invest in new computers once my personal home sells both for my own sake as it is time for an upgrade but also for the sake of this project's success.

What We Would Do Differently

Through a semester of research and documentation, there are some things that we have seen go well. There have also been some things that we have learned did not work, not only for the good of the group, but for the good of the project as well. This section will go over what each member thinks we could have done better and, if given another chance, would do differently.

Austin Dunlop

The main things I would do differently are to do with more efficiently accomplishing my work on the project. As stated in the design section, there were two major setbacks in simulation design that required revision. Setback one was using the Autodesk models for the simulation instead of creating new models. This could have been avoided by creating new models from the beginning.

The second setback was calibrating the joints of the arm to move properly. Part of the issue that made this process take such effort was that I went in with minimal knowledge of the relationship of the joints and links in the unified robot description format (URDF). If I spent the time to fully understand the process of robot creation in URDF from the beginning instead of beginning by tinkering with the previous team's design, it could have saved a lot of time spent tediously troubleshooting.

Cooper Urich

There have been many things that I think our group could have done better, but the main and most important would be our efficiency and speed of communication. While writing

and researching my sections, I found that I spent just as much time checking if no one in the group had written what I was writing as I did actually doing my research/ writing. If we were able to start from scratch and redo this project, I feel like the first thing we would do is message in our chat/ discord every time we wanted to start a new section. This would make sure that we knew what person was researching something in a given day.

Another thing we could have done better with is keeping our teammates up to speed on the process of our own research. Personally, I got nervous a few times when I didn't know the progress of my teammates' work. Thankfully it was always done and turned in on time, but I feel like if everyone in the group stated in the Discord what they were doing every day for research/ writing it would have made it a lot easier to see where everyone was at.

Another thing I would go back and change is the pace at which the group contributed to this final design document. Initially we planned for about 5 pages per week, or about 1 page per day. As the semester progressed and other classes started becoming more challenging, we began to stray away from this pace which caused us to slow down significantly. My most important piece of advice is to start early and keep at it. I personally fell behind towards the middle of the semester and had to do two pages of research per day instead of one like I originally planned.

Christopher Jackson

A major thing that I often neglected to keep in mind when working on the project is how much work we already were able to pull from the previous teams that have contributed to the EZ-RASSOR project before us. Many times, I would be researching what we would need to develop our visual network for the project, completely forgetting that we already had access to the full documentation the previous teams have made showing off what ideas they had and what they ended up implementing into their design. Of course, regardless of if we had this information or not, we would still have to do plenty of our own research, as what we are trying to achieve with the arm design is quite different than what was needed for systems like the GPS-Denied systems, but there was still useful information provided by the previous teams, and at the very least, many headaches related to the development that were already discovered by those teams.

The other thing I would have done differently going into the project is to have a closer collaboration with those of us working on the visual network. While we did try to stay in decent contact with each other, during the beginning stages of research, we mainly went separate directions finding what we could to fit the project's needs. Although this helped us divide learning things we would for different aspects of the visual network, it would have really helped when coming up with a finalized design to have been in closer

communication with each other. The advice I would give to future teams, and probably what most other teams and my partners would say as well, is that communication is one of, if not the most important factor in having a successful project. You can have a team full of skilled individuals, but if none of them communicate, you will never be able to get much of anything done. Luckily, the research we did ended up providing a lot of useful information going forward with development. I am confident as we begin our implementation in the summer, that our design for the visual network will set us up for success.

Luca Gigliobianco

If I had to do this project again, I would most definitely come in with better computer hardware. At the start of this project, I was delayed in installing certain necessary software, like Ubuntu 18.04 on a separate hard drive partition because of conflicts with my motherboard. On top of this, the one device where I could dual boot Linux (my laptop) is lacking in processing and GPU power. This makes it extremely difficult and resource draining to run simulations on Gazebo.

For the most part, I would say that any computer with dedicated graphics and a sufficiently powerful CPU can handle the requirements for this project. Unfortunately, my motherboard on my desktop had an issue where I could not dual-boot a Unix installer. For Senior Design 2, I will hopefully comeback with upgraded hardware components that can easily handle the resource requirements needed for simulations.

Something else I also would also improve upon is my reading of the former team's documentation. While the focus of our EZ-RASSOR projects is the arm, the previous teams have contributed countless amounts of research and documentation on their findings. Looking back, so many topics would have been much simpler if I had consulted with what the other teams have written. This includes instructions on installation as well as research on softwares and hardwares used.

Finally, an aspect I think all the team members would change is how often contributions were made to the final design document. While Senior Design 1 makes us keep some sort of pace by submitting earlier versions of the documentation, not consistently adding to it will end up becoming a huge burden later. It is better to put any findings and work done into the documentation as you go, so you don't have to struggle in the end to make it all fit.

Robert Forristall

Going into this project involved making quite a few mistakes along the way that could have been avoided if there were just things I knew before undertaking this task. It is worthwhile

to analyze these things I wish I knew to be used as advice for anyone that plans to attempt to work on a project similar to this one as well as develop on top of the work done by this team. Note that these topics will be about both the specific project discussed in this document but also some aspects of the class for which this project is done since there is a very likely chance that future students will develop on top of this project as said by the sponsor.

To start off its best to discuss some things worth noting about the class itself that this project is associated with; Senior Design has been the most informative and exciting classes that I have taken at UCF thus far. To succeed in this class there are a lot of steps that need to be taken solely on the will of the individual which leads to the first bit of advice that can be given to others for this class that also goes with any project; do what you need to do to stay motivated. Losing motivation on a project or class can be detrimental but is far more impactful when it comes to a class like Senior Design and by extension a project of this size. The 30 pages a person can be daunting and learning about something that is completely new to you can be overwhelming but keeping a level head and doing a little each day so as to not fall behind or burn yourself out is crucial. From personal experience, I studied and researched so much at the start of the project with the intention of including as much as I could in my writing that not only did, I burn out a little on my motivation but also overloaded myself with things to write about that it became even hard to decide what was most important to discuss. In the end I had to take a step back to focus on other aspects of school as well as myself to come back to the work of this class/project with a renewed motivation to continue learning and growing. The second biggest advice that can be given for this class is to take advantage of the resourceful minds of not just the professors and TA's but also the sponsor and people related to the project that would likely jump at the opportunity to help. The professors, Dr. Heinrich and Dr. Leinecker are extremely knowledgeable individuals that are there to not only help with the success of your project but also inform you about things that no other class goes over regarding the real world. These topics include things about graduate school, ethics in computing, monetary topics regarding both personal and business financials, and so much more that can be greatly used to the advantage of any student. As for the sponsor and other individuals associated with the project; seeing the project succeed is their ultimate goal and will go to great lengths to give the resources and advice needed to get started. For this team it was daunting at first to begin this project but after talking with not just the sponsor but some fellow students in their second year of senior design that mentality was subdued with the insight and encouragement given to the team.

Going into the project itself there were countless things I wish I had a better understanding that could be discussed but this will focus on the most impacting aspects. One such

noteworthy topic to discuss is Github; having done only a few projects with other groups as well as sticking to smaller solo projects I had never fully put in the time to learn about all the things that should be done for a repository. From making up a wiki, to handling the creation of a .gitignore file it is certainly worth the time for any individual to learn these skills if they are to get into coding. While I have begun studying these topics myself since as the project manager it will likely be my job to oversee this repository I wish I didn't have to spend the time now to learn it and wished it was something I focused on earlier in the semester or even before Senior Design. Following the issues of learning the ins and outs of Github is doing the same but with the operating system Linux. Similar to Github, I had minimal experience on Linux and how to run a lot of the things I am used to running on a Windows machine since it was never a huge requirement up till this project outside of small tasks on a virtual machine. Linux has been a game changer to learn as it has taught me a lot about why so many invest in a strong linux machine as well as prefer it for programming over other operating systems like Windows and MacOS. In the end, a new permanent Linux machine has become something I hope to invest in for the future due to the many advantages it has as well as programs that are built to run best on it as our simulation program Gazebo does.

What The Group Has Learned

There have been many things that we have learned throughout the process of this research paper. We have carefully laid out all of the different aspects of the project that we have researched heavily over a semester in Senior Design 1, however there are things that we have learned that were not impactful enough for their own sections, or they are things that we noticed during research that, if a group were to replicate this project or attempt to research a topic similar to this, should be aware of before stepping into this research.

Austin Dunlop

As the lead on simulation implementation, I learned a lot about the simulation software this semester. My education started with blender, the modeling software for the rover arm meshes. I learned how to navigate blenders workspace and create objects. I then learned about ROS, the software architecture for the robot. I learned some about ROS from my initial research phase but far more from working with ROS building the rover model. This was a lot like learning a new programming language, which turns out to be easiest when you're trying to build a program.

Part of building the simulation model included multiple tests to see how the model looked and functioned in the Gazebo environment. This gave me experience using the Gazebo simulation tool. ROS, blender and Gazebo are three different programs used to create the

simulation and products of all three programs are used in the final simulation working together.

Finally, probably the biggest thing I learned working on this project was adapting to using a Linux operating system that was required to run the simulation efficiently. I have not used Linux before, primarily being a windows user. I had experience using Linux command prompts and Linux file system navigation with an ubuntu shell in windows but never as the primary operating system. I learned how to dual boot ubuntu Linux with windows on my laptop and continue to adapt to using the Linux OS. Probably the biggest change from windows to Linux that I had to adapt to was using the command prompt more for basic OS functions and finding new Linux versions of Windows programs I often use.

Cooper Urich

My main topic of research for the majority of this project was object and edge detection. You will see that I go over research topics regarding box bounding, training object detection software, and multiple different types of edge detection. Previous to starting this project I had taken a robot vision class here at UCF (CAP 4453) and we had gone over all of the different types of edge and object detection but we were required to implement everything from scratch. Needless to say that previous to researching all the steps to the different methods, I wish I would know just how easy it was to implement all of those methods in real time. As you will see in my research, I take a deep five into all of the different steps and run time of each method. I wish I would have known how easy it was to test all of these for myself from the beginning. Having learned this towards the tail end of the semester saved me some time, but knowing this initially would have saved me a lot of headache searching for speed tests and comparisons.

Another major topic that I have learned throughout this process is just how important it is that the team is in constant communication. I have heard of other teams not having weekly meetings and how much of a disaster that is for their group. I believe that our group has done a fantastic job at communicating with each other. Learning this at a very early step in this process is paramount if you want to set your team up for a successful semester. For future groups attempting to recreate this project or do a similar project, focusing primarily on communication would be your number one priority. There have been times where I go to research a specific topic and it turns out one of my other groupmates already did that research. If it wasn’t for our groups constant communication, I would have done unnecessary research and writing.

Christopher Jackson

Researching each of the topics on computer vision has taught me a lot about how the different techniques are used in practice. Taking courses such as Intro to AI and Robot Vision has helped a lot with my understanding of specific methods and concepts of object detections, motion processing and machine learning, but learning them in these courses has mostly shown their use in specific contexts. This project has so far helped me really think about how each technique can play a role in achieving the end goal of the project. Even more so, it has challenged me to think deeply about how each of the individual techniques can be combined to one cohesive visual network. Although it took time, it was not too complicated to figure out the uses of each method individually but determining how each method would play a specific role in completing each task required, such as object detection and classification and motion detection so the proper data could be fed to our autonomous network required a lot of communication and planning between me and those involved with the visual network design.

This project has also provided a great opportunity for learning better teamwork skills, as well as an opportunity to learn about communicating with a stakeholder outside of the development team, that being our sponsor, Mike Conroy. While I have had experiences through other courses with team projects, all of them have been limited to a semester or part of a semester in length. This meant that less work had to be done in thinking about all aspects of a full project, such as project scope, detailed project planning such as building diagrams that represent a plan for the flow of how a system will work, and goal-oriented planning that requires setting deadlines that the team must meet as individuals but also as a collaboration with other members. This project so far has required each of those previously mentioned steps, and I can easily say that without staying on top of each of them, the team would not have been as successful. Our project sponsor has also provided us with a lot of advice and has helped push us in the right direction along the way. There have been several cases where part of our planning required us to get further information on details for what the rover already has implemented or what will eventually be implemented for the rover and arm, and he has also helped guide us in terms of scope of the project. In cases when he was not able to provide us the information we needed, he has always directed us to who could help us with the problem we had.

My time so far with the project has been overwhelmingly positive, and I now understand the things that were mentioned during the Senior Design Bootcamp about this being an experience that none of us will forget. Working with my team so far has been easy and seamless, and we each have done anything we could to make sure we all are successful individually and as a group. I am looking forward to how the project progresses as we begin to implement each of the features. What has been accomplished so far with the models in the simulation and manual movement of the arm in the simulation has already put us on a

good start to a fully completed project, and I anticipate by the end of the summer we will have a more realized visual network working with the simulation and will be ready for moving towards a physical implementation in the Fall.

Luca Gigliobianco

From my time participating in this work for Senior Design, I have learned that large scale projects require an immense amount of planning, communication, and hard work. When I chose to partake in this project, I knew that I would have my work cut out for me, but the actual experience of working on this has been extremely fruitful and beneficial.

To start off, this project has been the first time I ever dual booted an OS by partitioning storage. To use the robot operating system (ROS) and other Linux-specific libraries required for the project, we had to install Ubuntu 18.04 LTS alongside our native OS. While this may not be very hard to do or learn, it is still a very useful skill for programmers to know if they ever need to use an operating system without access from a virtual box or a shell. This was also my first experience with Blender, and how it is used to make three-dimensional models.

I also experienced a robotics simulation program in the form of Gazebo. This has been the team's platform of choice for testing out EZ-RASSOR arm models, as well as seeing simulations of the entire unit. Additionally, we have added controls for the arm simulation in here. Gazebo uses many well received physics engines and serves as a good tool of practice for when our team has to test using real components.

The most technical information I have learned however is related to the Arduino Mega 2560 board and the Jetson Nano. The Mega 2560 is a microcontroller board that will be used to link the visual network with the physical components of the arm. Arduino boards are the heart of many robotics projects and connect all the pieces of the puzzle. The Jetson Nano is a small computer that will be used to process the images taken with the RealSense camera. This project has also taught me basic information about AI and robot vision, including edge detection, neural/visual pathways, etc.

More importantly, the non-technical lessons I have learned have been much more valuable. Big collaboration efforts like this stress the importance of clear and proper communication between team members. Understanding your team members and their responsibilities is paramount to properly synchronizing efforts. This also expands into proper planning and preparation, to organize the members.

Robert Forristall

Throughout the research and design for this project, there have been many lessons learned along the way that I would have likely never learned outside of this project and class. It has been an amazing experience that I plan to carry with me fully and one that can become even greater over the next half of the year as we go fully into the implementation of our design and goals. With that there are a few big points that I believe are worth discussing from my point of view.

Starting with the research; I never imagined how interesting some of the research done during this project could be. My focus in computer science is artificial intelligence as it has always been a huge interest of mine, so my original plan was to try and find a project to scratch that itch. After getting assigned to this group to design the arm I was sure that sticking to the artificial intelligence of the system was my best bet but over time I came to have quite the interest in the actual robotics side of the machine along with the simulation environment that is being used with it. While I still look forward to some of the AI work that is to be done such as the autonomous controller, it has been amazing to learn about all the ins and outs of programming robotics from their operating systems to path planning algorithms and even to the simulation and its models/communications.

Outside of the research, the biggest thing I learned was about all the things that come with managing a project. This is my first time managing a project like this so there was a steep learning curve in not only the project itself but how to be a good manager for the project and support my teammates as best I can. I'm certainly not the kind of person that easily finds himself able to manage something like this; being a quieter person that prefers to listen to the ideas and thoughts of others while keeping mine to myself unless asked due to my anxiety has always weighed heavily on me. When the opportunity came to try and step out of that heavy shadow, I thought I could use the experience both for the future of other projects and for my own confidence. This led me to have to be able to speak up, ask questions about the project, and openly communicate the plans of the project while listening to the advice and insight of my groupmates to incorporate in any decisions that were made. Along with the self-improvement learned was how to physically managing a team including but not limited to stuff like documentation through Gantt charts and project related diagrams, organizing meetings with the group as well as members from FSI and previous EZ-RASSOR team members, and handling the day to day questions and reminders for the group.

References

[1] "Who has Walked on the Moon? – NASA Solar System Exploration", *NASA Solar System Exploration*, 2021. [Online]. Available: https://solarsystem.nasa.gov/news/890/who-has-walked-on-the-moon/. [Accessed: 18- Feb- 2021].

[2] "NASA Accepts Challenge to Send American Astronauts to Moon in 2024", *NASA*, 2021. [Online]. Available: https://www.nasa.gov/feature/sending-american-astronauts-to-moon-in-2024-nasa-accepts-challenge/. [Accessed: 18- Feb- 2021].

[3] N. Davis, "Should We Colonize the Moon? And How Much Would It Cost?", *Pacific Standard*, 2021. [Online]. Available: https://psmag.com/environment/colonize-moon-much-cost-81543. [Accessed: 18- Feb- 2021].

[4] J. Hoft, "Report: Russia to Begin Moon Colonization by 2030", *The Gateway Pundit*, 2021. [Online]. Available: https://www.thegatewaypundit.com/2014/05/report-russia-to-begin-moon-colonization-by-2030/. [Accessed: 18- Feb- 2021].

[5] "How long will the Earth remain habitable?", *Image.gsfc.nasa.gov*, 2021. [Online]. Available: https://image.gsfc.nasa.gov/poetry/venus/q79.html. [Accessed: 18- Feb- 2021].

[6] D. Lewis, "A Million Years, Give or Take – Now I Know", *Nowiknow.com*, 2021. [Online]. Available: http://nowiknow.com/a-million-years-give-or-take/. [Accessed: 18- Feb- 2021].

[7] Mordvintsev, A. M., & Rahman K., A. R. K. "Introduction to OpenCV-Python Tutorials — OpenCV-Python Tutorials 1 documentation". 2013. [Online]. Available https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_setup/py_intro/py_intro.html#intro [Accessed: 23- Feb- 2021].

[8] Librealsense. (2017, December 04). Retrieved March 07, 2021, from https://github.com/IntelRealSense/librealsense/blob/master/readme.md

[9] Laspy. (n.d.). Retrieved April 15, 2021, from https://pypi.org/project/laspy/

[10] *Introduction to NumPy*. (n.d.). *www.W3schools.com*. Available: from https://www.w3schools.com/python/numpy_intro.asp [Accessed: 23- Feb- 2021].

[11] Verma, S. *How Fast Numpy Really is and Why? - Towards Data Science*. *Towardsdatascience.com,* 2019 [Online]. Available https://towardsdatascience.com/how-fast-numpy-really-is-e9111df44347#:%7E:text=Because%20the%20Numpy%20array%20is,leap%20in%20terms%20of%20speed [Accessed: 23-Feb-2021].

[12] A, Choudhury. "10 Best Python Libraries For Computer Vision". *www.analyticsindiamag.com,* 2019 [Online]. Available https://analyticsindiamag.com/10-best-python-libraries-for-computer-vision/ [Accessed: 25-Feb-2021]

[13] Virtual machine. (n.d.). Retrieved February 26, 2021, from https://www.vmware.com/topics/glossary/content/virtual-machine

[14] Comments, 2. (n.d.). How to dual-boot Linux and Windows. Retrieved February 18, 2021, from https://opensource.com/article/18/5/dual-boot-linux

[15] "Catkin Package Summary", *ROS.org*, 2017. [Online]. Available: http://wiki.ros.org/catkin. [Accessed: 16- Mar- 2021].

[16] "urdf/XML/model - ROS Wiki", Wiki.ros.org, 2021. [Online]. Available: http://wiki.ros.org/urdf/XML/model. [Accessed: 27- Apr- 2021].

[17] "urdf/Tutorials/Using Xacro to Clean Up a URDF File - ROS Wiki", Wiki.ros.org, 2021. [Online]. Available: http://wiki.ros.org/urdf/Tutorials/Using%20Xacro%20to%20Clean%20Up%20a%20URDF%20File. [Accessed: 27- Apr- 2021].

[18] "urdf/XML/joint - ROS Wiki", *ROS.org*, 2018. [Online]. Available: http://wiki.ros.org/urdf/XML/joint. [Accessed: 16- Mar- 2021].

[19] "urdf/XML/model - ROS Wiki", Wiki.ros.org, 2021. [Online]. Available: http://wiki.ros.org/urdf/XML/model. [Accessed: 27- Apr- 2021].

[20] B. Guru, "Blender Beginner Tutorial - Part 1", *Youtube.com*, 2019. [Online]. Available: https://www.youtube.com/watch?v=TPrnSACiTJ4. [Accessed: 16- Mar- 2021].

[21] "ROS/Introduction" https://wiki.ros.org/ROS/Introduction

[22] "Changes between ROS 1 and ROS 2" https://design.ros2.org/articles/changes.html

[23] "Player Manual" http://playerstage.sourceforge/doc/Player-3.0.2/player/index.html

[24] “YARP: Welcome to YARP” https://www.yarp.it/git-master/index.html

[25] “OROCOS Project Documentation” http://docs.orocos.org

[26] S, Maitra. “What Canny Edge Detection algorithm is all about?” *Medium.com 2019* [Online]. Available https://medium.com/@ssatyajitmaitra/what-canny-edge-detection-algorithm-is-all-about-103d94553d21 [Accessed: 25 - Feb - 2021]

[27] “Gaussian Filter” *Wikipedia.com* [Online] Available https://en.wikipedia.org/wiki/Gaussian_filter [Accessed: 27-Feb-2021]

[28] *Gaussian Blur Wikipedia.com* [Online] Available https://en.wikipedia.org/wiki/Gaussian_blur [Accessed: 28-Feb-2021]

[29] Sambasivarao. K, “Non-maximum Suppression (NMS)*”. Towardsdatascience.com [Online]* Available https://towardsdatascience.com/non-maximum-suppression-nms-93ce178e177c [Accessed: 27-Feb-2021]

[30] S, Sahir “Canny Edge Detection Step by Step in Python — Computer Vision”. *Towarddatascience.com* [Online] Available https://towardsdatascience.com/canny-edge-detection-step-by-step-in-python-computer-vision-b49c3a2d8123 [Accessed: 28-Feb-2021]

[31] “Roberts Cross Edge Detector” *homepages.inf.ed.ac.uk* [Online] Available https://homepages.inf.ed.ac.uk/rbf/HIPR2/roberts.htm [Accessed: 1-Mar-2021]

[32] U, Sinha “The Sobel and Laplacian Edge Detectors” *www.aishack.in* [Online] https://aishack.in/tutorials/sobel-laplacian-edge-detectors/ [Accessed: 03-Mar-2021]

[33] G.T. Shrivakshan “A Comparison of various Edge Detection Techniques used in Image Processing” IJCSI International Journal of Computer Science Issues Vol. 9, Issue 5,

[34] Ashish, “Understanding Edge Detection (Sobel Operator)” *medium.com* [Online] https://medium.datadriveninvestor.com/understanding-edge-detection-sobel-operator-2aada303b900 [Accessed: 06-Mar-2021]

[35] Kim, D. (n.d.). Sobel Operator and Canny Edge Detector. Retrieved March 11, 2021, from https://www.egr.msu.edu/classes/ece480/capstone/fall13/group04/docs/danapp.pdf

[36] S.K. Katiyar “Comparative analysis of common edge detection techniques in context of object extraction” Department Of Civil Engineering MA National Institute of Technology, India

[37] R, Winastwan “How to Create a Simple Object Detection System with Python and ImageAI” *www.towardsdatascience.com [Online]* https://towardsdatascience.com/how-to-create-a-simple-object-detection-system-with-python-and-imageai-ee1bcaf6b111 [Accessed 05-Mar-2021]

[38] Shah, A. (2018, June 17). Through the eyes of Gabor Filter. Retrieved April 11, 2021, from https://medium.com/@anuj_shah/through-the-eyes-of-gabor-filter-17d1fdb3ac97

[39] Bobick, Aaron. “CS4495-15-MotionModels.” Georgia Institute of Technology.

[40] Astua, C., Barber, R., Crespo, J., & Jardon, A. (2014, April 11). Object detection techniques applied on mobile robot semantic navigation. Retrieved March 09, 2021, from https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4029636/

[41] 13.3. Object Detection and Bounding Boxes. (n.d.). Retrieved April 02, 2021, from https://d2l.ai/chapter_computer-vision/bounding-box.html

[42] Lobo, Niels Da Vitoria. “Optical Flow.” University of Central Florida, 2021.

[43] Lui, Ce, et al. “Motion and Optical Flow.” University of Washington.

[44] “How to Use Background Subtraction Methods.” *OpenCV,* docs.opencv.org/master/d1/dc5/tutorial_background_subtraction.html.

[45] Tamersoy, Birgi. “Background Subtraction.” The University of Texas at Austin, 29 Sept. 2009.

[46] Stauffer, Chris, and W.E.L Grimson. “Adaptive Background Mixture Models for Real-Time Tracking.” Massachusetts Institute of Technology, 13 Feb. 2009.

[47] Point cloud. (n.d.). Retrieved March 13, 2021, from https://en.wikipedia.org/wiki/Point_cloud

[48] Triangle mesh. (2020, September 30). Retrieved April 06, 2021, from https://en.wikipedia.org/wiki/Triangle_mesh

[49] Non-uniform rational b-spline. (2). Retrieved April 06, 2021, from https://en.wikipedia.org/wiki/Non-uniform_rational_B-spline

[50] Poux, F. (2020, April 21). 5-Step guide to generate 3D meshes from point clouds with Python. Retrieved April 06, 2021, from https://towardsdatascience.com/5-step-guide-to-generate-3d-meshes-from-point-clouds-with-python-36bad397d8ba

[51] K. A. Johannessen and E. Fonn 2020 J. Phys.: Conf. Ser. 1669 012032

[52] Poux, F. (2020, November 21). How to automate lidar point cloud processing with python. Retrieved April 08, 2021, from https://towardsdatascience.com/how-to-automate-lidar-point-cloud-processing-with-python-a027454a536c

[53] “Sending Commands from Rviz.” Sending Commands from Rviz - Dev Documentation, ardupilot.org/dev/docs/ros-rviz.html.

[54] “Integration.” *ROS.org*, www.ros.org/integration/.

[55] “Move Group C++ Interface.” *Move Group C++ Interface - moveit_tutorials Noetic Documentation*, ros-planning.github.io/moveit_tutorials/doc/move_group_interface/move_group_interface_tutorial.html.

[56] “Move Group Python Interface.” *Move Group Python Interface - moveit_tutorials Noetic Documentation*, ros-planning.github.io/moveit_tutorials/doc/move_group_python_interface/move_group_python_interface_tutorial.html.

[57] “Integration/Unit Tests.” *Integration/Unit Tests - moveit_tutorials Kinetic Documentation*, docs.ros.org/en/melodic/api/moveit_tutorials/html/doc/tests/tests_tutorial.html.

[58] S.K. Katiyar “Comparative analysis of common edge detection techniques in context of object extraction” Department Of Civil Engineering MA National Institute of Technology, India

[59] “Description - ROS-Industrial.” *ROS*, rosindustrial.org/about/description.

[60] “Wiki.” *Ros.org*, wiki.ros.org/Industrial/Tutorials.

[61] Castrounis, Alex. “AI, Deep Learning, and Neural Networks Explained.” *InnoArchiTech*, AI Advice, Guidance, and Due Diligence | InnoArchiTech, 25 Feb. 2021, www.innoarchitech.com/blog/artificial-intelligence-deep-learning-neural-networks-explained.

[62] Hardesty, Larry. “Explained: Neural Networks.” *MIT News | Massachusetts Institute of Technology*, 14 Apr. 2017, news.mit.edu/2017/explained-neural-networks-deep-learning-0414.

[63] Lobo, Niels Da Vitoria. “Introduction to Neural Networks for CAP 4453.” University of Central Florida, 2021.

[64] Kizrak, Ayyuce. “Comparison of Activation Functions for Deep Neural Networks.” *Medium*, Towards Data Science, 7 Jan. 2020, towardsdatascience.com/comparison-of-activation-functions-for-deep-neural-networks-706ac4284c8a.

[65] Saha, Sumit. “A Comprehensive Guide to Convolutional Neural Networks - the ELI5 Way.” *Medium*, Towards Data Science, 17 Dec. 2018, towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53.

[66] Lobo, Niels Da Vitoria. “Convolutional Neural Networks for CAP 4453.” University of Central Florida, 2021.

[67] Wang, Liqiang. “Artificial Intelligence CAP 4630 Lecture 14. Deep Neural Network Architectures.” University of Central Florida, 2020.

[68] Liu, Jerry, et al. “DSIC: Deep Stereo Image Compression.” Uber ATG, 2019.

[69] Gadelha, Matheus, et al. “Shape Reconstruction Using Differentiable Projections and Deep Priors.” University of Massachusetts, 2019.

[70] Du Y-C, Muslikhin M, Hsieh T-H, Wang M-S. Stereo Vision-Based Object Recognition and Manipulation by Regions with Convolutional Neural Network. Electronics. 2020; 9(2):210. https://doi.org/10.3390/electronics9020210

[71] G.R, S., Kumar, N., Hari, P., & Sasikumar, S. (2018). Implementation of a Stereo vision based system for visual feedback control of Robotic Arm for space manipulations. *Procedia Computer Science, 133*, 1066-1073.

[72] Howie Choset, H.C. (2010). *Robotic Motion Planning: Configuration Space*[Powerpoint slides]. The Robotics Institute, Carnegie Mellon University. https://www.cs.cmu.edu/~motionplanning/lecture/Chap3-Config-Space_howie.pdf

[73] Oussama Khatib O.K. The Jacobian. *Introduction to Robotics*(pp. 81-124).

[74] Matula, D. (1993). A linear time 2 + &&egr;ε approximation algorithm for edge connectivity. *SODA '93*.

[75] Lydia E. Kavraki, & Jean-Claude Latombe. (1998). Probabilistic Roadmaps for Robot Path Planning. *Http://Www.Cs.Rice.Edu/CS/Robotics/Papers/./Kavraki1998prm-for-Robot-Path-Plan.Pdf*.

[76] Leven, P., & Hutchinson, S. (2002). A framework for real-time path planning in changing environments. *INTERNATIONAL JOURNAL OF ROBOTICS RESEARCH*, *21*(12), 999–1030.

[77] T. Kunz & U. Reiser & M. Stilman & A. Verl, (2010). Real-time path planning for a robot arm in changing environments. *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2010, pp. 5906-5911, doi: 10.1109/IROS.2010.5653275.

[78] "EZ-Rassor Architecture" https://github.com/FlaSpaceInst/EZ-RASSOR/wiki/architecture

[79] M, Danziger "The core differences between detection abilities of the two methods" *www.foresightauto.com* [Online] https://www.foresightauto.com/stereo-vs-mono/#:~:text=A%20mono%20vision%20system%20enables,just%20landed%20in%20front%20of [Accessed 07-Mar-2021]

[80] "What is a stereo vision camera?" *www.e-consystems.com* [Online] https://www.e-consystems.com/blog/camera/what-is-a-stereo-vision-camera/ [Accessed 12-Mar-2021]

[81] Arduino Mega 2560 rev3. (n.d.). Retrieved March 03, 2021, from https://store.arduino.cc/usa/mega-2560-r3

[82] Jetson Nano developer kit. (n.d.). Retrieved March 04, 2021, from https://developer.nvidia.com/embedded/jetson-nano-developer-kit