Simulated Swarm (White Team)

Group 25

Fall 2021 - Spring 2022

The Simulated Swarm White Team consists of team members Keifer Wheatley, Huy Pham, Bryce Hitchcock, Raymond Price, and Daniel Cisneros.

Project Sponsors: Lockheed Martin (Joseph Rivera and Scott Nelson).

Table of Contents

1 Executive Summary

The senior design class of Fall 2021 - Spring 2022 included many interesting and challenging projects for students to tackle. From a virtual reality game to database-backed mobile applications, there were many to choose from to inspire creative minds to solve their own respective problems. Creativity and complexity are exactly what our group has expressed and exercised when choosing to take on this project.

Our project is in competition with two other groups aiming to solve the same fundamental challenge. The challenge being to make use of drones to solve a real-world problem. Search and rescue missions require precision and haste due to one important factor, time. The faster a target can be found, the higher chance of a positive outcome in a real-world situation. Using drones has proved to be not only a safe way but also a fast way to conduct such a search and rescue mission. This project strives to be an innovation for this problem and a three-group competition can inspire teams to solve this problem in the fastest and best way within a productive and encouraging environment. The research and steps that we have taken in order to complete this project were extensive and are highlighted throughout our design document.

2 Project Overview

2.1 Project Description

Drones are becoming commonplace in many different applications today. Many problems and limitations that arise in certain situations can be solved with drones. Ranging from reconnaissance to material transportation, the applications of Unmanned Aerial Vehicles (UAV’s) are quickly rising.

In this project, we are required to use a simulated urban environment to develop an autonomous swarm of aerial and/or ground vehicles that will search for a predetermined target. Using software such as Gazebo to simulate the environment and then ROS for robotic competency, we will be tackling the problem in a five-man team against two other five-man teams. This will be conducted in a limited amount of time and will require

different techniques and algorithms such as SLAM for constructing a map of the environment and YOLO for real-time object detection.

2.2 Statements of Motivation

Keifer Wheatley

I was motivated to select this project because it integrates many areas of computer science that I am interested in. When I read that the project will be using robotic systems, machine learning techniques, and the latest simulation technology, I knew I had to put this as my first choice. Being able to work with these technologies is going to be a great opportunity to learn and grow as a programmer. I am not an expert when it comes to all of these new technologies, but I always enjoy learning new topics and can't wait to get started. I feel all the classes I have taken in my undergraduate degree have prepared me to work on a larger scale project like this one and I am excited.

The impact of this technology was also a major motivation factor for me. Drones and other autonomous vehicles can be used in so many different ways. We could use the software we are designing for a search and rescue situation. This shows the real-world impact something like this could have. I wanted my choice of project to be something meaningful and fulfilling and this choice checks both boxes.

I am very excited to be working with this team and with the combination of our skills and hard work not only will this project succeed but hopefully we will place first amongst the two other teams.

Bryce Hitchcock

My motivation to sign up for this project came from the opportunities this project has to offer. It can be really difficult to get started in the area of machine learning, computer vision, and robotics, especially when you don't have a mentor or someone to show you the ropes. That was a big obstacle for me because these are areas, I am really passionate about and want to learn more. I've always looked at the cool things that engineers were able to accomplish with these technologies such as Tesla, Boston Dynamics, and even some people on YouTube and wanted to start learning but never had anywhere to start. That's why I think this project would be perfect for me. Having a clear cut, real-world problem that involves the intersection of machine learning,

computer vision, and robotics with an experienced mentor would be an amazing opportunity to learn more about these areas and hopefully be a steppingstone into this industry.

Another big motivation for me to pick this project was the networking opportunities. Lockheed is one of the giants in the software engineering industry, so this project provides invaluable networking opportunities, not only with the sponsors but also with my groupmates. Being in a group with a bunch of like-minded people would be fun and I think it would be a great way to network and gain some insight from others who are in a similar situation as me.

Raymond Price

Machine learning and artificial intelligence has always piqued my interest. I have always had a strong math and coding background and really enjoyed the introductory classes for both topics. Ever since I learned the basics, I have longed for a more challenging opportunity to really deepen my understanding of machine learning. This project, being machine learning intensive, is just what I was looking for and made it an easy first choice. At first, seeing the requirements for this project made me feel overwhelmed and that the learning I will have to do will be too steep. Learning how to simulate a drone swarm along with the machine learning associated is very extensive and will require a lot of knowledge to be obtained. Now, learning these things is what makes me excited and is what motivates me. In the end, I know the final result will have given me invaluable information and the accomplishment of the project will be well worth it.

Furthermore, in our computer science career paths we will need a lot of experience working in teams and meeting our specific requirements. What better way to prove that I can handle these responsibilities than taking on a project from a major company? We will be working bi-weekly with a sponsor and weekly as a team, meaning we will need commitment. If we say we will have a certain aspect of our project done or say we will finish our research by a certain deadline, there are many different factors that will pressure us to get our work done. I look forward to being honest and respectable as both a teammate and a student and therefore will remain motivated to do what I say I will do.

Huy Pham

My motivation for working on this project was the robotics and machine learning aspects of the project. I have always had an interest in working with electronics. In high school, I joined my high school's robotics team, where we would meet weekly to work with different electronics to do simple tasks. While I was there, my club entered the Lockheed Martin competition, where we had to design a robot that could be remotely navigated through a course and push balls into our own goal. We were split into two teams that worked independently from one another within our club. While we were originally just competing against other schools, the team split sparked a rivalry between the two teams. Preparing for this project was one of the most memorable experiences I've had throughout high school. I learned that I like working with others to create something we can show off at the end of the day. I was one of the programmers for that project and it was one of the reasons I enjoy programming now. Now and then, I continue to talk to previous team members to build new robots.

My experiences lead to me working as a coding and robotics intern at a company called STEM-ME, where I created a curriculum for students interested in learning STEM skills. From this opportunity, I was able to help others get a head start in STEM as well as refine my understanding of computer science. This project is beyond what I have worked on before, but I am eager to work with and learn alongside my teammates. While our main goal is application work, my personal goal is to create an application that functions and performs better than the other teams.

Daniel Cisneros

A lot of the computer science classes for undergrads give you the fundamental concepts and theory that can be applied to the real world. I chose to work on this project because I have always been interested in the artificial intelligence, machine learning, deep learning, computer vision, and cyber security divisions. With my background knowledge in math and coding experiences from previous classes I believe this project can be done even if I do not have the necessary experience. Therefore, the idea of implementing these fields in a real-world environment and learning new technologies will give me the experience on how this can be used. Also, I chose this project because I am interested in joining one of the military branches and helping to create a better world with the help of technology.

This project is sponsored by Lockheed Martin, which is an international company that focuses on aerospace, arms, defense, information security, and technology worldwide.

The main goal of this project is to work with robotic systems with the specifications of a float of 5-6 drones that will be able to fly autonomously and find its objective. This also caught my interest because nowadays drones are being used for multiple tasks like security, surveillance, recognition, and I strongly believe that in some future not too far they will be able to do even more tasks like delivering things to humans and help them in their daily tasks. The idea of making a robotic system fly autonomously excites me because I have always been curious on how this is done with the help of AI and computer vision.

Even though there are going to be some challenges like learning new software, implementations, and designs. These same challenges also excite me because in my previous experiences, computer science and other careers in the technology branch requires to always do some research with constant learning to deliver the most efficient and practical way to make good software implementation. I believe this is a big start and with enough preparation, effort, and teamwork a lot of things can be done not only for this project but in general.

2.3 Goals and objectives

Design an efficient machine-learning algorithm for drones to search and find a given target in a simulated environment. ● Efficiently implement swarm behavior.
Locate the target within the time constraint of 3 minutes. ● Find the most efficient methods of locating our target. ● Outperform other teams.
Able to display bounding box around the target ● This is an imaginary rectangle that serves as a point of reference for object detection.
Stretch Goal: a tactical display that keeps track of the coordinates of the drone and maps them out conveniently to be seen like a mini map.

2.4 Initial Ideas

Bryce Hitchcock

As building prototypes and having hands on access to the drones and models is very important for the completion of this project, my idea is to start with learning as much about ROS and gazebo as we can through online resources. Also, we need to have a common GitHub repository with all our code and hopefully local environments set up for everyone to be able to develop and test their own solutions. I think this will help out in speeding up development. As for the actual drone swarm project, I think a fair amount of time should be spent in looking up different autonomous vehicle and object detection algorithms and weighing their advantages and disadvantages. On top of this I think looking into previous work on drone swarm technology either from academic research, real-world applications, or previous Senior Design groups would be very helpful.

My idea for our solution would be to have each drone breakdown the target area into individual segments and report to each other which segment they intend to search. They will then individually search each of their own respective areas to determine where the target is and report back to the other drones when the target is found.

Raymond Price

This project is going to require very creative minds. In order for the most innovation and creativity a deep understanding of what we as team members need to do is required. I think the first place that we should start is researching about our project. Using the project information from the team that came before us, we can form a baseline or a starting point that we can build from. We also need to become accustomed to the new languages and software that we will be using such as ROS, Gazebo, YOLO, and SLAM. Utilizing free online courses to learn this material is another idea I have as a solution to this.

Keifer Wheatley

My idea for this project is to first become familiar with all new technologies we will be using to run our simulations. If we are able to learn as much as we can about ROS, Gazebo and our algorithms it will help a lot during our development process. After thorough research we will be able to begin implementing our ideas and training our models. My idea is that each agent breaks off and scans an area of the environment sending back information to all other agents. Doing this we will be able to perform a full sweep of the environment swiftly and locate the target. The problem is reducing agent overlap and overall time to perform the maneuvers. This is what will have to be thoroughly tested through trial and error on Gazebo.

Daniel Cisneros

My idea for this project will be to assign each drone a role but also, they will have a similar implantation to complete their mission which is finding its objective. For example, these tasks can be divided by making one of the drones a leader, another drone will be able to confirm the objective, another drone can go back to the initial state point and with the help of graph theory give the most efficient route to the target, if necessary. Another drone can mark the last position where the target was last seen if it's moving and predict the next location, another drone will be able to expand the field view and obtain even more data than the others. Again, all these drones will have the ability to communicate with each other when they have the most relevant information in real time. Also, these drones can change their task according to the environment.

Before trying to implement all these roles a lot of research needs to be done. A good entry point will be to start learning as much as possible about ROS which is an open-source for robotic systems and Gazebo which is also an open-source software for robotic simulations. This can be done by completing some courses and gaining experience inside their environment. Research about different types of algorithms for mapping, moving objects, sensing, exploration, training model, collision avoidance, and communication by choosing the most efficient solution according to the project specifications.

Huy Pham

My idea for this project is to have a centralized computer that calculates an optimal flight path for each of the drones to follow. As the drones scout out more of the area using their built-in sensors, the computer updates the path of each of the drones as well as generates a comprehensive map of the environment. Once most of the borders and walls of the environment are added and the target is not located, the drones can make a final sweep of the zone to map out areas that were unaccounted for. The main problem is implementing the machine learning algorithm that the drones will follow. The system would have to make sure to avoid collisions or overlap between drones. This may be very difficult to implement because of the real-time updates that would have to occur in unknown environments and being aware of the other drones to avoid collisions.

2.5 Broader Impacts

Once successfully implemented, our project will help demonstrate the searching methods that drones will follow while searching and mapping out an area and locating a target. This will be done in a stimulating environment with a machine/deep learning algorithm that will perform a search solution with effective swarm navigation and inter-agent communication. In a dire situation, it is important for the drone to be processing and pursuing the most effective path. As the use cases of drones increase, the need for more powerful search algorithms increases. Instead of having a physical pilot, creating a more autonomous method that can operate with minimal experience and minimizes human error.

One of our stretch goals is to create a tactical display that shows the GPS location of the target. This is an example of the data collection capabilities that drones can be utilized in.

In optimal circumstances, our project will help in speeding up the decisions of autonomous search drones as well as strengthen their effectiveness in critical moments. With the addition of the tactical display, we hope we can help improve drone data collection in any way we can.

2.6 Issues

Legal Issues

Here are some patents related to our project (oldest to newest). This list is not exhaustive.

- WO2016203322A2: Claims rescue drone that can fit on aircraft consisting of certain parts.
- EP2942688A1: Claims flying drone with a memory for storing forecasted flying trajectory defined by 3D coordinates points, and the drone also has a control and command computer for driving drone directional and motorization.
- US6856894B1: Claims a learning autopilot that can perform complex maneuvers on the vehicle when controlled by its ground station.

WO2016203322A2 is very specific and doesn't apply to our drone. EP2942688A1 and US6856894B1 are generic, and our project won't exactly fit the constants of these patents. With the approval of the sponsors, we should be outside the effects of any patents.

Ethical Issues

Our project has the potential to be applied in more hazardous applications outside of this project. It could be used to create an effective way of harming others in wartime. If a weapon were to be added onto the drone, it could become a very dangerous and efficient search and destroy device. It could be used on a personal drone to follow others and assist in nefarious plans such as robberies and other crimes. The worst aspect is that the drone could remotely be deployed and increase the scope of damage that someone could inflict.

While there is a chance that this project could be used in more malicious ways, the project itself is just a simulation for a competition. It is not the first of its kind and the simulation is more focused on the search rather than the destroy function. There will be a lot more improvements that need to be made until the project can be effective in real-life situations.

Privacy Issues

There aren't many privacy issues involving this project. Unlike most real-life drone experiments, we won't be flying the drones through other people's property/airspace to perform the search. If this project were to be implemented in the real world, the drones would have to search in private property or request permission to search public areas. There is especially a problem if the drone causes harm to a civilian. The majority of our project will be simulated which doesn't require formal access to an environment and won't affect the lives of civilians.

3 Specifications and Requirements

3.1 Project Requirements

- Utilize Robot Operating System Software (ROS) to implement a system that an actual aerial or ground drone could utilize in a search mission.

- Use an urban simulated environment through Gazebo to imitate realistic features and obstacles that an aerial or ground drone may face, and also have a simulated drone work through this environment.

- Develop and utilize existing machine learning algorithms, such as YOLO, for modern object detection. Use this to prove that with computer vision a detected bounding box can be formed with a specific confidence score and accuracy.
    - Bounding Box around target - must demonstrate IoU > 0.5
    - Confidence Score - must be above 0.7

- Implement search patterns and algorithms for the autonomous search of a given object within a specific time constraint.
    - Time constraint of 3 minutes to ID target

- Must use a specified number of drones to imitate swarm behaviors. These agents must collaborate and work together to find a target in a synchronous way.
    - At least 3 drones

- STRETCH GOAL:
    - A tactical display - Show pseudo-GPS coordinates of the target on a grid (mini map)

3.2 Figures and Diagrams

Figures

The drone will need to search in an urban environment as shown below and will need to overcome obstacles with computer vision and pathfinding algorithms. These obstacles may be physical objects that the drone may have to avoid, or it may be a visual impairment that may impact the autonomous functionality such as a shadow. The simulated drone will have cameras to take constant photos that will later be used to train a machine learning algorithm in order to perform this object detection.

Image: Visual of Simulated Drone in a Simulated World [Lockheed Martin Presentation]

Target

In the simulated world, our drone's objective is to find the set target within a set time frame. Therefore, an important decision we have to make is what we should make the target of our search algorithms. There are a number of factors that we should take into account when choosing our target. The target we end up using for this project will be a bb-8 drone from the movie Star Wars.

The reasoning behind our choice largely comes from the target being the previous team's target. The target was given to us from the previous team's project files which makes deploying the model much more straightforward. The previous team has also already tested the bb-8 as a target, so it has some functional backing from the previous teams. We are also using the environment of the previous team so the target should be compatible with the environment we have chosen.

The drone will be flying around an urban environment and taking constant photos in order to map and identify where the target is. Once the target is within view of the camera the machine learning algorithm must be able to draw a bounding box around it and identify the position it is in. There will be obstacles and other things that will impact this; therefore, the algorithm must be trained to efficiently account for these hindrances. Below is a figure depicting a bounding box drawn around the target object within the urban world, with shadows and other objects behind it.

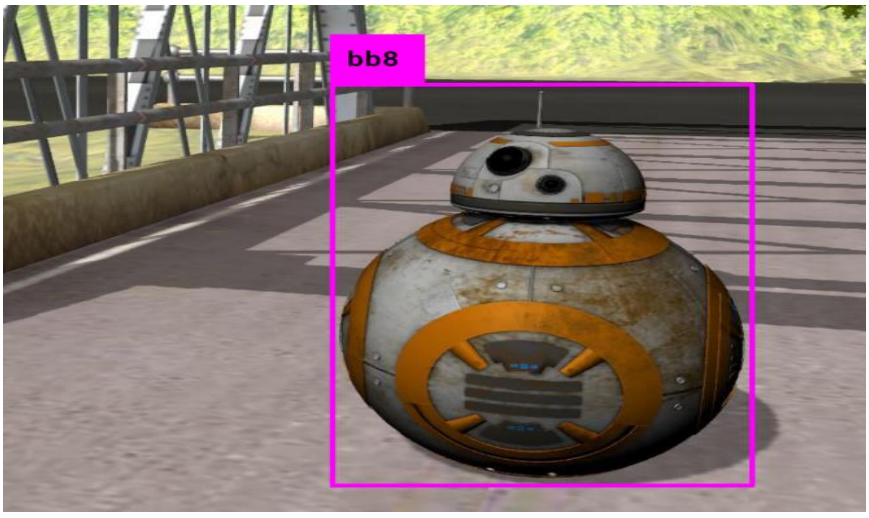
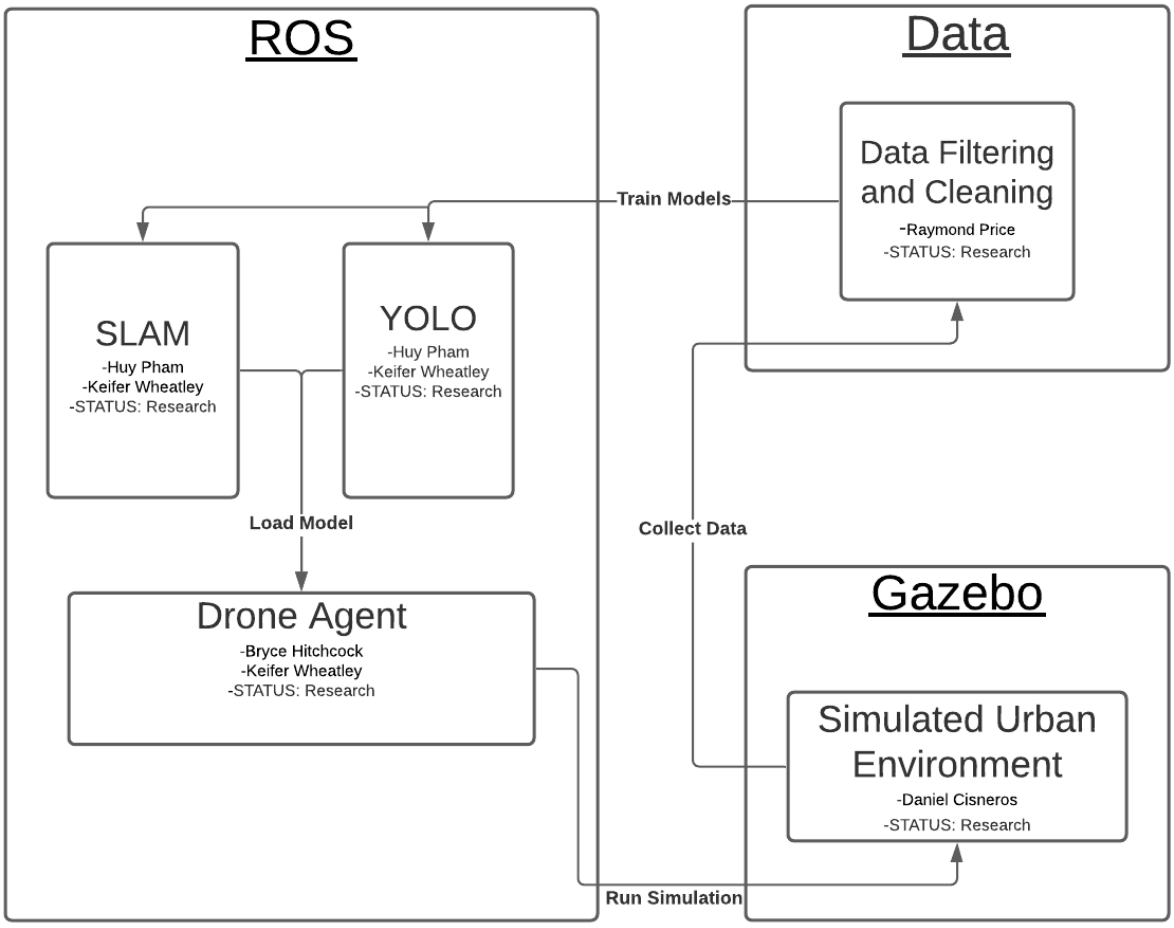


Image: Example Bounding Box Drawn Around Target [previous team]

Block Diagram



The block diagram shown above depicts the general organization of our project's assets and dependencies as well as each team member's roles in these areas. The general outline of our workflow involves using ROS to build our drone agents that will eventually be used in the simulated environment found in Gazebo. These drone agents will be trained on data that is also accumulated from the simulated environment found in Gazebo and filtered and cleaned to be used in training. This data will then aid in training the detection and pathfinding models/algorithms to be used in the next iteration of model testing.

3.3 Software and Tools Used

Establishing a software development environment can be critical to the success or failure of a project. There are many tools and software development options that exist and identifying, selecting, and establishing the software and tools that optimize performance and satisfy the needs of both the development team and the project requirements can be daunting. “Over the last 20 years the set of software tools available to developers has expanded considerably [59]”. The software and tools required to establish a cohesive and efficient development environment can vary based on the system being developed, the system being used to develop the software, the test environment, the configuration management environment, etc. Bulajic, et al, recommends an effective software development flowchart illustrated in the figure below.

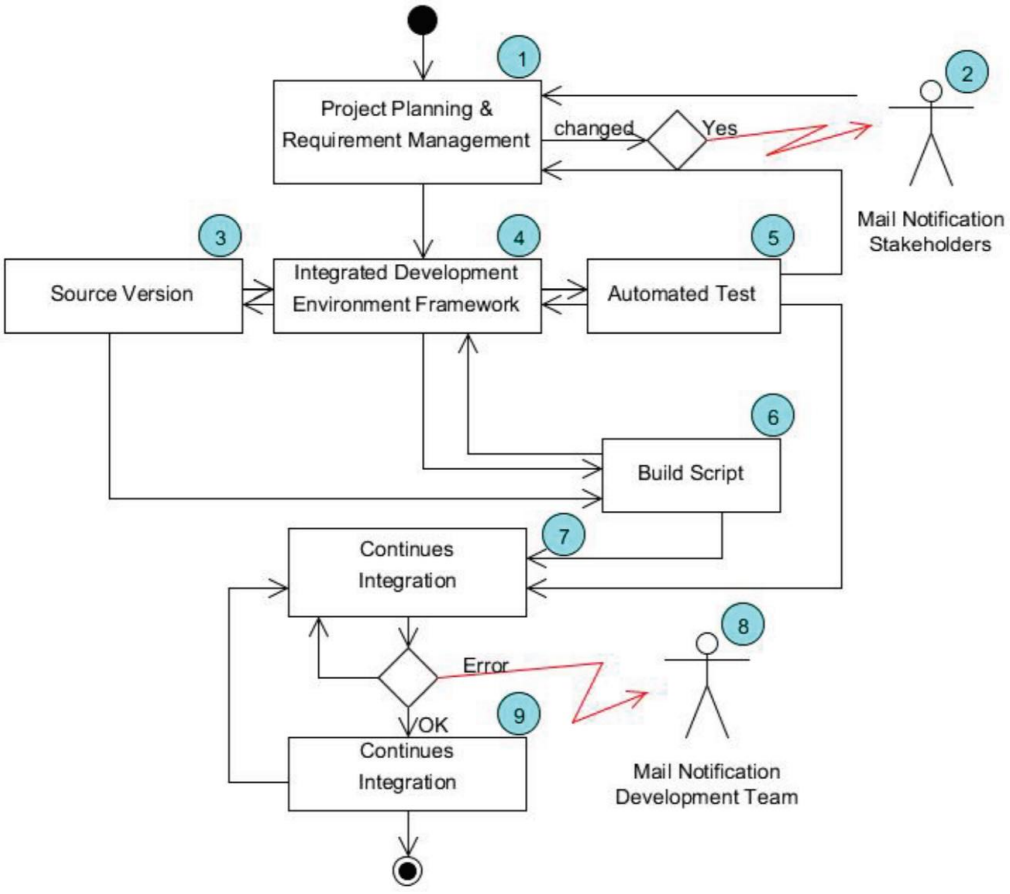


Diagram: An Effective Software Development Environment Model [60]

In addition to these considerations, the system itself needs software and tools in order to have a successful operational system. The system requires an Operating System, libraries, development and debug tools, and other resources to aid in the software development process. In the following sections, the operating selection and the software development will be discussed and described.

Operating System Selection

There are many different criteria for selecting an Operating System for a project. Not all operating systems can perform the same types of tasks equally. Operating systems can vary greatly between those used for a personal computer application versus those used for high performance real-time or near real-time applications. The most common operating systems are Windows, Mac OS, and Linux [61]. These are primarily computer operating systems that provide functions specific to computer use, such as hardware functionality control, memory and disk storage management, and graphical user interface and multi-media control. Linux is an operating system used by programmers and applications developed around the world [61].

For this project there were two operating systems that needed to be selected. The first operating system provides the development environment that the team would use to develop our target hardware solution. And the second operating system would be used on the target hardware environment used in the project. Given that we likely won't be using actual target hardware, the target hardware will be simulated on our development environment as a target hardware application, but we will still need to select an appropriate operating system that would be used on target hardware in order to provide realistic results for the project.

Development Operating System

Within our team, we initially installed ROS and Gazebo into our differing operating systems: Windows, Linux, and macOS. The Windows and macOS versions of ROS and Gazebo required a large number of dependencies and resulted in unresolved errors. Some of the errors we had were unrecognized commands and bad substitutions.

Windows

ROS has recently come out with support for Windows. Within our team, three of the five members were Window users: Bryce, the robot specialist; Raymond, the machine learning specialist; and Keifer, the Pathfinding Specialist.
Setting up ROS required supplementary software and many dependencies. While following tutorials for the installation, there were many errors that were unexpected. The installation process had numerous steps that may require further research or their own tutorials. The large number of steps made it difficult to backtrack and locate the problem in the installation. In addition, the community and documentation behind Windows are minuscule compared to the support for Linux. This made finding solutions very difficult. Ultimately, they opted in for a virtual machine running Linux.

macOS

Daniel, our simulation specialist, was the only macOS user on our team. His ROS installation worked but still had persistent errors that couldn't be resolved. In contrast to the Windows users, he managed to install Gazebo onto his machine, however, his machine had a newer M1 chip, which operates on a different architecture. He needed to use an open-source 3-D graphics engine, Orge, for his Gazebo to function. The engine didn't patch for the M1 chips and after consulting forums, it was concluded that updates were being worked on, but, ultimately, he opted into using a virtual machine that ran Linux because of the current lack of support for the M1 chip and the simplicity of the installation in comparison.

Linux/Ubuntu

Huy, the project manager, was the only Linux user and ran Ubuntu 20.04. In contrast to Windows and macOS, Linux had an abundance of support and documentation for ROS and Gazebo. The installation process for Linux can be completed quickly through

technologies' website tutorials and doesn't have as many unexpected problems and errors occur. Any problem that did appear was easily resolved and was generally already listed as a concern in the tutorial with a suggested link to a solution. For these benefits, our team has decided to completely work off of Linux now. As listed before, the members of the team who don't run Linux as their primary operating system will emulate Linux using a virtual machine.

Target Environment Application Operating System

The Target Hardware Application Operating System must be selected to provide the most fully functional and application specific operating system for our project. There are a number of operating systems designed specifically for our application using drones. However, the selection needs to also consider the concept of robotic vision, machine learning, and other intelligent decision point applications. The following paragraphs will provide some details from the operating system research for our simulated swarm of drones.

FlytOS made by Flytbase is an operating system made specifically for drones. The FlytOS provides several key capabilities that seem to be ideally suited for our application. This OS provides onboard intelligence for collision avoidance, precision landing, and object tracking. It also provides for payload integration for sensors such as camera, thermal sensor, LiDAR, and multispectral camera. It has drone adapters for drone compatibility with all major drone/autopilot platforms and cloud connectivity using 3G and 4G/LTE for real-time access to a drone fleet [62].

LynxOS made by Lynx Software Technologies is a leading provider of real-time operating systems and secure hypervisors for avionics, aerospace and defense applications. LynxOS is a small-footprint kernel and support for multi-core/multi-threaded processors. It appears this has very large adoption in the large aerospace market as Lynx Software Technologies references clients, such as NASA, Boeing, Raytheon, General Dynamics and the US Armed Forces [63].

MAVSDK Software Development Kit made by Auterion is an open-source operating system for enterprise drones. MAVSDK Software Development Kit provides a set of libraries in different programming languages (C++, Python, Swift, Java) that provide high-level APIs to the MAVLink protocol for communication between a ground control

station and drones, or the drone and a payload sensor. The following is a list of capabilities for MAVSDK [64]:

- Provides a high-level, user-friendly API for developers (MAVLink is too basic for many use cases, even with language bindings).
- It allows the community to be cross-platform and supports multiple programming languages in a consistent manner.
- MAVSDK is scalable. Previous solutions, such as the defunct Drone Kit, were difficult to maintain because each language had its own implementation.
- The performance and scalability allow the use case for swarm scenarios, which requires a highly efficient backend.
- It empowers the SDK to be extensible for specific use cases and features.
- Most importantly, given the open nature of the MAVSDK, the goal is to enable a grassroots approach to innovation by sharing and collaborating with different “actors” (service providers, manufacturers) on a common API without hindering them from diversifying and improving.

FreeRTOS, though not specifically an operating system designed for drones, is an open-source real-time operating system (RTOS) that could be used in drone applications. In comparison to a general-purpose operating system (GPOS), such as Linux, real-time operating systems are smaller in code size, execute faster, and are deterministic. For example, FreeRTOS’s code size is 70,000 lines of code while a common Linux kernel is approximately 19 million lines of code [65]. This small footprint might be ideal for small hardware platforms used in small, lightweight drones. In order to keep the code size small, however, the operating system has scaled back the functionality provided.

Robot Operating System (ROS) is not actually an operating system but provides the middleware between an operating system and Application Programming Interfaces (APIs). ROS is a flexible framework for writing robot software. It is a collection of tools, libraries, and conventions that aim to simplify the task of creating complex and robust robot behavior across a wide variety of robotic platforms. Core elements of ROS are listed below:

- Communications Infrastructure
  - Message Passing
    - Recording and Playback of Messages
    - Remote Procedure Calls
    - Distributed Parameter System
  - Standard Message Definitions for Robots
  - Robot Geometry Library
  - Robot Description Language

- Preemptable Remote Procedure Calls
- Diagnostics
- Pose Estimation
- Localization
- Mapping
- Navigation

There are many criteria for the selection of the target environment operating system. One of the critical selection criteria for our Simulated Swarm operating system is cost. Since our budget for this project is three-hundred dollars, we are restricted to open source or free software applications because the three-hundred dollars could cover small costs such as specialized courses to learn a technology but not the larger costs such as the drone or renting an urban environment. Fortunately, for a general-purpose operating system, there are many viable sources available, but none that have the features and capabilities that some of the researched drone operating systems offer. However, under closer review of **FlytOS** from Flytbase, we discovered that FlytOS is built from Robot Operating System (ROS) and Linux. Both of these software applications are readily available as open source and can be combined to form the basis for our Simulated Swarm project.

The selection of **Robot Operating System (ROS)** with **Linux** works well with our development environment, which is also based on the Linux operating system. Any additional software packages required to implement the Simulated Swarm project will either be developed, or open-source libraries will be searched for relevant applications.

Virtual Box

To ensure that each team member is developing in the same development environment, a common virtual machine toolset will be established. **VirtualBox** by Oracle provides a robust x86 and AMD64/Intel64 open-source solution that runs on Linux, Macintosh, Solaris, and Windows hosts [68]. This will allow each team member to develop the project from any host machine that they are comfortable with yet provide the exact same project environment across the entire team. **VirtualBox** provides a development environment that has been abstracted away from the host machine's operating system.

A development environment will be established in the **VirtualBox** virtual machine and a clone of this VM environment will be uploaded into **GitHub** for retrieval by each team member. The establishment of this common development environment will eliminate the possibility that team member’s development environments are inconsistent, which could

ultimately cause delays and issues with incompatible builds or integration tasks. Figure 4 illustrates the virtual machine buildup of the development environment, including the target Linux OS and the Robot Operating System (ROS).

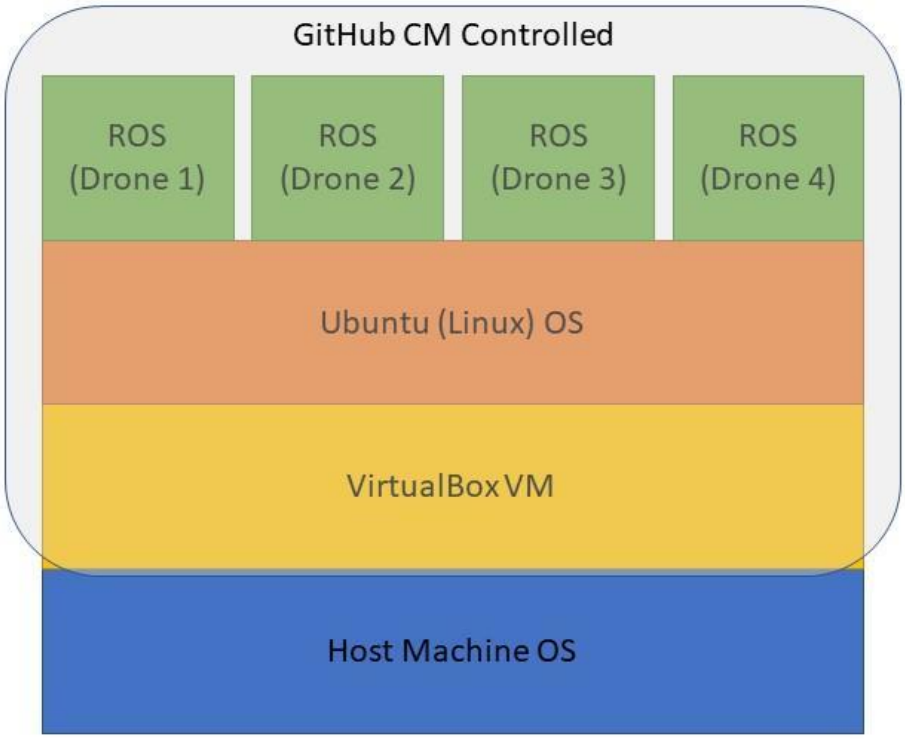


Image: Virtual Machine Layout Depiction[68]

VMware Virtual Machine Install

The decision to use a virtual machine in this project was made because of how easy the installation process for the software we use is in a Linux environment. Rather than dual boot Linux, the option of a virtual machine is easy and completely free to do.

Step 1: Download and run Vmware. Vmware has a free tier that can be downloaded. For this project Vmware Workstation 16 Player was used.

Step 2: Download Ubuntu Desktop OS. Ubuntu is free and easy to download. For this project Ubuntu 20.04.3 L was used.

Step 3: Run the Vmware software and create a new virtual machine, link the ISO file path of the Ubuntu download to the installer disc file location, and then proceed to making the environment. 25 GB disk space was allocated in this project for the virtual machine.

Once the machine is operational, you will know everything is functional when you have a fully operational ubuntu desktop available in Vmware Workstation 16 Player.

ROS (Robotic Operating System)

ROS Installation Guide for Windows

In order to ensure the entire team is operating in the same environment, the following setup guide was standardized for the team. This section also includes a list of packages and dependencies as well as their versions. Note: this installation guide is only for Windows 10 and utilizes ROS Foxy Fitzroy version of ROS.

All instructions for this installation can be found on ROS.org here.

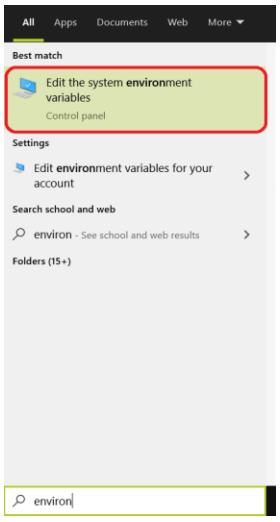
Install Dependencies

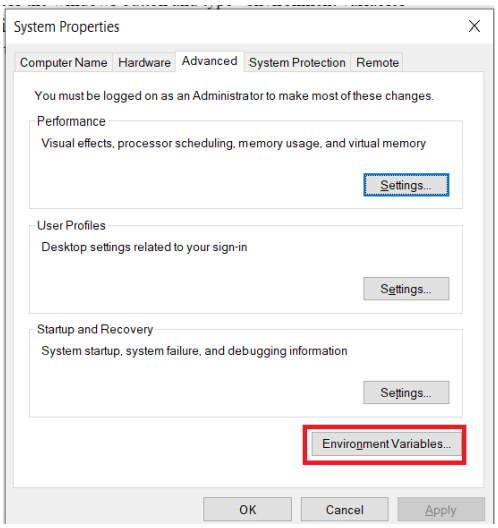
1. Install Chocolatey
    a. Open Windows PowerShell and make sure to run as administrator
    b. Run the following command:
    Set-ExecutionPolicy Bypass -Scope Process -Force; [System.Net.ServicePointManager]::SecurityProtocol = [System.Net.ServicePointManager]::SecurityProtocol -bor 3072; iex ((New-Object System.Net.WebClient).DownloadString('https://community.chocolatey.org/install.ps1'))

2. Install Python 3.8.3 through chocolatey
    a. Open a command prompt
    b. Run the following command
    choco install -y python --version 3.8.3
    c. Check that python was install to *C:\python38*

3. Install Visual C++ Redistributables through chocolatey
    a. Open a command prompt
    b. Run the following command in PowerShell
    choco install -y vcredist2013 vcredist140

4. Install OpenSSL
    a. Head to this link
    b. Scroll to bottom and download the Win64 OpenSSL v1.1.1L exe link
    Note: do NOT download the win32 or light versions of OpenSSL v1.1.1L and keep all settings default
    c. Once installed add the OpenSSL-Win64 bin folder to your PATH [3]
        i. Press the windows button and type “environment variables”
        ii. Click “Edit the system environment variables”
        iii. In the dialog box that opens:
            1. Click ‘Environment Variables…’
            2. Click ‘Path’ under System variables
            3. Click Edit
            4. Click New
            5. Type in ‘C:\Program Files\OpenSSL-Win64\bin\’
            6. Click OK





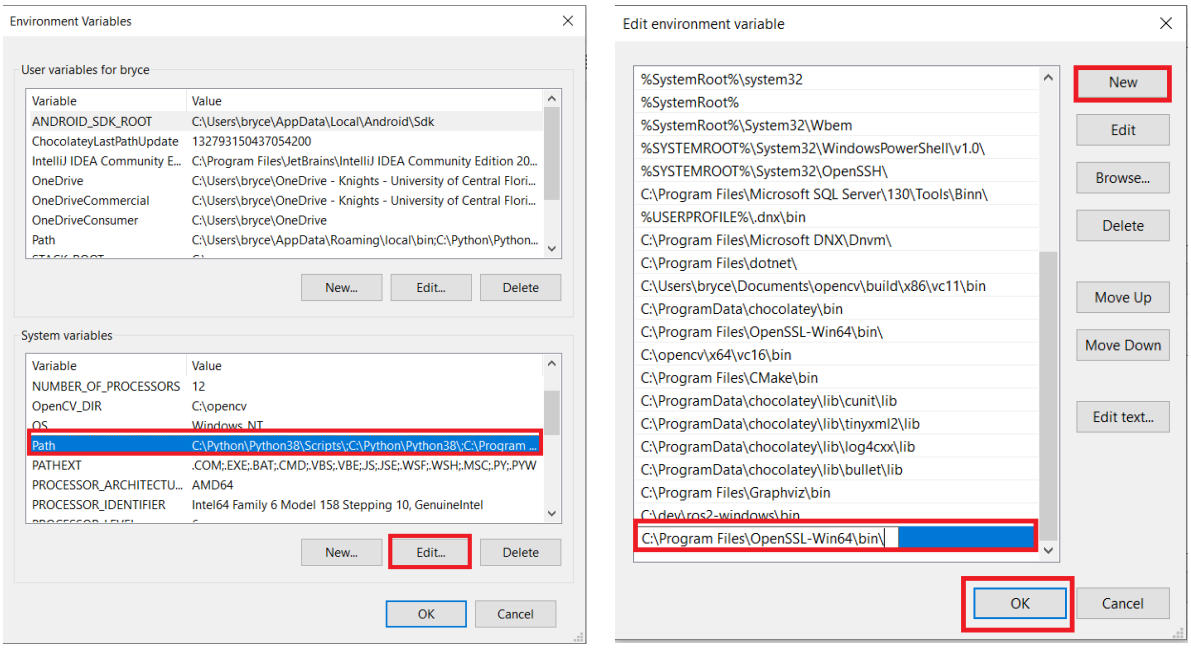
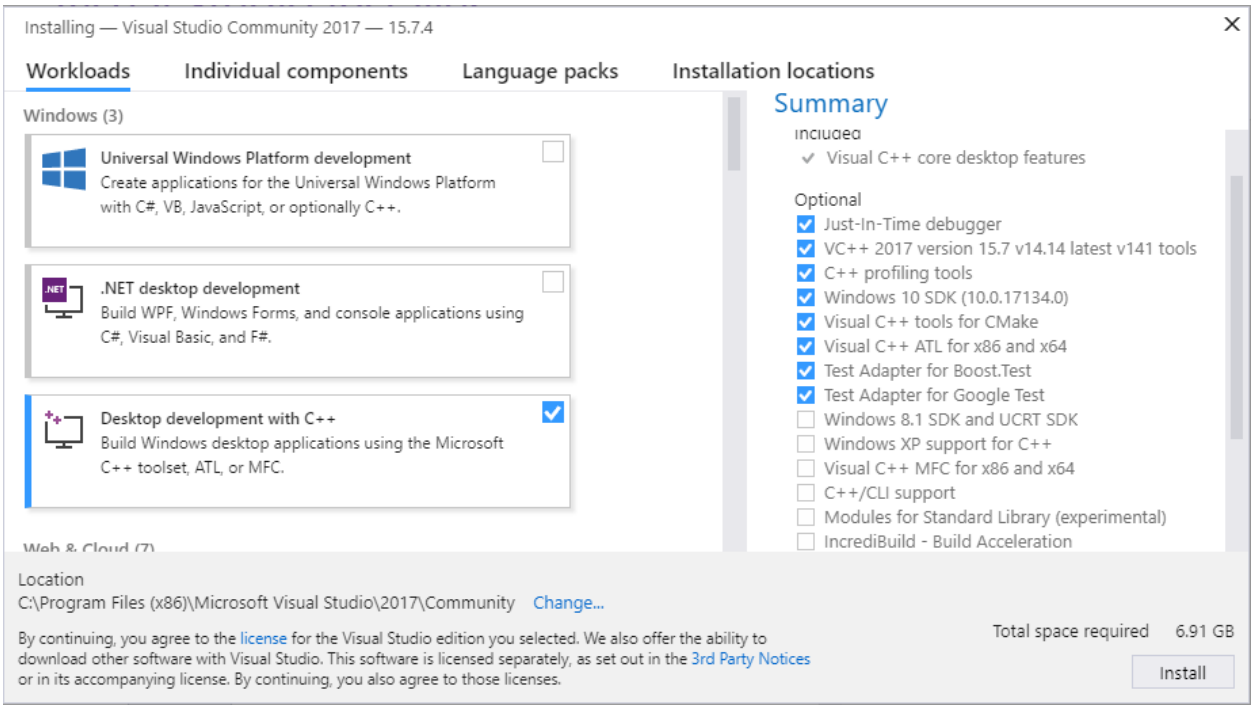


Figure 5: How to add to PATH Environment Variable[3]

5. Install Visual Studio 2019
    a. Download the Visual Studio 2019 Community Version from here
    Note: When installing be sure to select Desktop development with C++ and deselect any C++ CMake tools from the side menu



Visual Studio Installation Settings[3]

6. Install OpenCV
    a. Download OpenCV from here
    b. Unpack to '*C:\opencv*'
    c. Run the command *'setx -m OpenCV_DIR C:\opencv*' in command prompt
    d. Once installed add the OpenCV bin folder to your PATH
       Note: consult figure 5 for these steps
        i. Press the windows button and type "environment variables"
        ii. Click "Edit the system environment variables"
        iii. In the dialog box that opens
            1. Click 'Environment Variables…'
            2. Click 'Path' under System variables
            3. Click Edit
            4. Click New
            5. Type in '*C:\opencv\x64\vc16\bin*'
            6. Click OK

7. Install Extra Dependencies
    a. Install CMake
        i. Run the command in command prompt
           'choco install -y cmake'
        ii. Once installed add the CMakebin folder to your PATH
           Note: consult figure 5 for these steps
            1. Press the windows button and type "environment variables"
            2. Click "Edit the system environment variables"
            3. In the dialog box that opens
                a. Click 'Environment Variables…'
                b. Click 'Path' under System variables
                c. Click Edit
                d. Click New
                e. Type in 'C:\Program Files\CMake\bin'
                f. Click OK

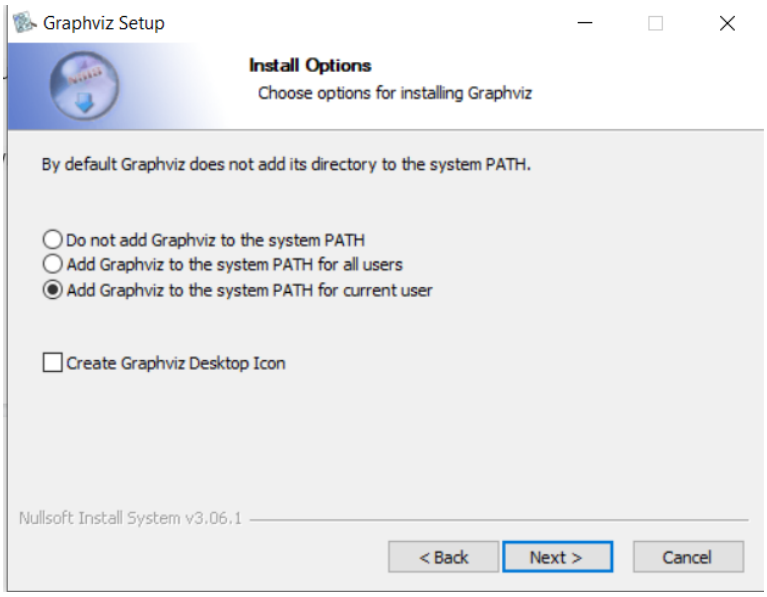
    b. Download the following packages from this Github repository
        i. asio.1.12.1.nupkg
        ii. bullet.2.89.0.nupkg
        iii. cunit.2.1.3.nupkg
        iv. eigen-3.3.4.nupkg
        v. tinyxml-usestl.2.6.2.nupkg
        vi. tinyxml2.6.0.0.nupkg
        vii. Log4cxx.0.10.0.nupkg

viii. Once these packages are installed open PowerShell as an administrator
ix. Run the following command in PowerShell:
choco install -y -s <PATH\TO\DOWNLOADS\> asio cunit eigen tinyxml-usestl tinyxml2 log4cxx bullet
Note: replace <PATH\TO\DOWNLOADS> with the path of the directory you downloaded all the packages to

c. Install some extra python dependencies
   i. Open command prompt
   ii. Run the following command:
   python -m pip install -U catkin_pkg cryptography empy ifcfg importlib-metadata lark-parser lxml matplotlib netifaces numpy opencv-python PyQt5 pip pillow psutil pycairo pydot pyparsing pyyaml rosdistro setuptools

8. Install Qt5
   a. Visit this link
   b. Select the Open-Source version and then the Qt Online Installer for Windows
   c. Use the online installer to install Qt5 and keep default settings
   d. Open PowerShell as an administrator
   e. Run the following command to set environment variables:
   setx -m Qt5_DIR C:\Qt\5.15.0\msvc2019_64
   QT_QPA_PLATFORM_PLUGIN_PATH C:\Qt\5.15.0\msvc2019_64\plugins\platforms

9. Install RQt dependencies
   a. Visit this link
   b. Install the graphviz-2.49.2 (64-bit) EXE installer for windows
   c. When the installer asks to add graphviz to PATH, select add to current user or all users



Graphvis installation settings[3]

10. Download ROS2
    a. Visit this link
    b. Download ros2-foxy-20211013-windows-release-amd64.zip
    c. Unzip the files to C:\dev\ros2_galactic

ROS 2 dependencies for windows

- graphviz-2.49.2
- Python 3.8.3
- Chocolatey v0.11.2
- ROS2
- OpenSSL v1.1.1L
- Visual Studio Community 2019
- OpenCV
- CMake
- asio.1.12.1.nupkg
- bullet.2.89.0.nupkg
- cunit.2.1.3.nupkg
- eigen-3.3.4.nupkg
- tinyxml-usestl.2.6.2.nupkg
- tinyxml2.6.0.0.nupkg
- Log4cxx.0.10.0.nupkg

3.4 Simulation and Physical Error

Our project will entirely be carried out in a simulation. The environment that we will be simulating is provided by the previous team, who worked on a similar project in the past. The simulation, as a whole, is an urban environment where our drones will be able to automatically search through the environment and locate our target.

For us as college students, the greatest limiting factor would be cost. The simulation greatly reduces the starting cost of the project; and some of the most significant costs that we avoided is a real-life urban environment, multiple drones, the sensors for the drones. The only funding, we received was three hundred dollars which is nowhere enough to cover any of those costs. We also avoided the large cost of failure such as when the drone crashes or a mechanical error occurs. The drone and the sensors attached would have to be replaced, and any additional affected part would need to be repaired.

The simulation will handle sensors almost optimally, assuming the software is well-made, while in contrast real-world sensors might be more troublesome. It is important to note the significant reduction of error within simulation. Errors can be simulated but there are a lot of factors to take into account. Even the physics engine could be tweaked to be made more or less like real-life. A real sensor could be dirty or malfunction while the ones in the simulation won't be affected by these random factors unless they are programmed in. A huge benefit for us is that any error that occurs for us can be quickly fixed by resetting the simulation while a physical will have to allocate time and resources into fixing that problem.

Some of the procedure that is absent when using a simulation is that we lack the physical processes and problems that occur in real life. We completely skip the process and problems of building the drone and the intricacy of making it functional. As stated before, the sensors could fail in real-life but there could also be mechanical, electrical, and many other problems that could occur with any part, but we avoid them when using a simulation. This may not be highlighted now but if this project were to become physical, the bulk of the work will be accounting for these issues.

Due to the scope of our project, we will mainly be working under ideal conditions. As long as our drones can fly and locate the target, we meet the goals and requirements for our project.

3.5 Supplemental Software and Tools Used

This section describes the software tools that are not related to the central project itself. Examples of some of the tools include project management, team communication, and code collaboration methods.

JIRA

Our team used Jira to implement an agile scrum framework for our project. We initially used the free version of Trello, but the free version of Jira has all the features of Trello and more such as a roadmap feature that allows us to visualize a timeline that illustrates our project schedule, in other words, a Gantt chart. Jira seems geared towards the scrum process with features such as sprints and backlog for our tasks. It also offers a method of assigning tasks to members of the team and board where we can add more tasks in.

Discord

Discord is our team's main method of communication. Most of our team meetings are held over discord unless they are held physically. We have channels, assigned chat rooms, that are meant for discussion on a specific task. Within the chatrooms, we have important messages and links pinned, so we can easily refer back to them quickly. Discord has both a mobile and computer version of its application so our team can be constantly informed of any new updates or issues that occur.

Version Control Software

Version Control is always critical to manage and track progress of any project. When working with a software development team that will be developing subcomponents of the final project, being able to keep all members of the team up to date on the current project configuration, tools, and project updates improves team performance and results. An additional complexity is having team members geographical disperse or working remotely further complicates the configuration management task. Ensuring version control, accessibility, and project updates are managed, controlled, and distributed requires a version control tool that can meet the team's needs and goals.

Fortunately, this is common practice in software development environments and many tools exist to ensure version control is an integral part of the software development toolbox. Our team will use **GitHub** is a cloud-based version control tool that has been widely used in the free and open-source software development ecosystem, as well as in many corporate software development environments. The **GitHub** collaborative development environment provides the following features and capabilities [9]:

- Code spaces - Visual Studio Code backed by high performance VMs that start in seconds.
- Pull Requests - Allow contributors to easily notify you of changes they've pushed to a repository – with access limited to the contributors you specify. Easily merge changes you accept.
- Notifications - Get updates on the GitHub activity you've subscribed to. Use the notifications inbox to customize, triage, and manage your updates.
- Code Reviews - Review new code, see visual code changes, and confidently merge code changes with automated status checks.
- Code Review Assignments - Assign code reviews to make it clear which team members should submit their review for a pull request.
- Code Owners - Automatically request reviews–or require approval—by selected contributors when changes are made to sections of code that they own.
- Draft Pull Requests - Use a pull request as a way to discuss and collaborate, without submitting to formal review or risking an unwanted merge.

- Protected Branches - Enforce restrictions on how code branches are merged, including requiring reviews, or allowing only specific contributors to work on a particular branch.
- Team Discussions - Post and discuss updates within your entire GitHub organization, or just your team. Notify participants with updates, and link from anywhere.

- Team Reviewers - Request a team on GitHub to review your pull request. Members of the team will get a notification indicating that you've asked for their review.
- Multiple Assignees - Assign up to 10 people to work on a given issue or pull request, letting you more easily track who's working on what.
- Multiple Reviewers - Request review from multiple contributors. Requested reviewers will be notified that you've asked for their review.
- Multi-line Comments - Clarify code reviews by referencing or commenting on multiple lines at once in a pull request diff view.
- Public Repositories - Work with any GitHub member on code in a public repository you control. Make changes, open a pull request, create an issue, and more.

4 Division of Labor

4.1 Huy Pham

- Project Manager
    - Coordinates team meeting dates and agendas
    - Responsible for meetings with the sponsor
    - Track all tasks in all aspects of the project and make sure that each member is doing what they say they will do
    - Assist in any component that needs help

4.2 Raymond Price

- Machine Learning Specialist
    - Will work to get the photographs and images needed to create a data set that will be used to train the drones computer vision
    - Using image machine learning technology, will train the dataset and return a bounding box around the target with a specified accuracy

4.3 Bryce Hitchcock

- Robotics (ROS) Specialist
    - Handle the implementation of the computer vision developed from machine learning and the pathfinding established to construct all of the drone's mechanics and how they will conduct search
    - Will also need to consult with pathfinding specialist to research swarm techniques and how to best synchronize the use of multiple drones

4.4 Daniel Cisneros

- Simulation (Gazebo) Specialist
    - Populate the urban environment and introduce obstacles that the drones might have to face, such as shadows, cars, odd colors, etc.
    - Help produce photos and situations for the machine learning specialist so that the drones are trained for multiple different situations
    - Will work with the pathfinding specialist to help map the area and give insight on different aspect of the simulated world

4.5 Keifer Wheatley

- Pathfinding (SLAM/YOLO) Specialist
    - Handle the pathfinding algorithms implemented with assistance from the Gazebo specialist
    - Will work closely with the ROS implementer in order to correctly program the drone to be able to traverse the environment

5 Research

5.1 Robot Operating System (ROS)

What is ROS

The robotic operating system (ROS) is a framework to write robot software. Since this project uses several robots, more specifically a float of five to six drones, ROS can become handy. With ROS, we can write the software for our drones' behavior and push it towards our desired goals. ROS has a collection of tools, libraries, and more conventions to simplify certain tasks to make complex robot behavior across robotic platforms. ROS has a large community that also makes it easier to share your work with others in the community and gives you the ability to test other people's work by obtaining their packages and deploying/implementing their work. This is also because ROS is an open-sourced software making it very accessible for people with a limited budget.

Nodes

A node in ROS is a process that carries out computation. Nodes are able to be combined into what is called a graph, where they can communicate with each other by sending messages through topics, which will be further explained in the next section. A robot system could be made up of many different nodes. For example, there may be a node that controls a camera, a node that controls a drone's wing motors, and a node that handles routing the drones.

Using these nodes in ROS helps developers in several ways. One of which being that individual nodes may fail or crash, but the system as a whole can have the other nodes still perform. By isolating out the nodes that crashed, the user can narrow down what nodes failed and begin to assess the problematic nodes. This can be a benefit as you are able to see exactly which part of the project is not performing as intended; greatly speeding up the process of troubleshooting. The code complexity of the code is also stated to be reduced, compared to monolithic systems.

Topics

ROS topics are mainly used for sending data streams between nodes. For example, you are monitoring the weight placed on a robot's wheels. The node that monitors the wheels will send a data stream with the current weight. Now any other node can subscribe to this topic and receive the current weight on the wheels. Topics have a publish/subscribe methodology that makes them anonymous. Nodes are not aware of where the data is coming from instead, they are just focused on the data being published to the topic.

Messages

ROS nodes need a way to communicate with one another and they can achieve this by publishing messages to topics. A message in ROS is a data structure that is made up of different typed fields. Data such as integers, floating points, Booleans and arrays of all these types and more are supported.

Messages can also contain a special type called “Headers”. Headers contain common metadata fields such as timestamp and frame ID. Here is an example of a header:

```
# Standard metadata for higher-level stamped data types.
# This is generally used to communicate timestamped data
# In a particular coordinate frame.
#
# Sequence ID: consecutively increasing ID
uint32 seq
#Two-integer timestamp that is expressed as:
# * stamp.sec: seconds (stamp_secs) since epoch (in Python the variable is called 'secs')
# * stamp.nsec: nanoseconds since stamp_secs (in Python the variable is called 'nsecs')
# Time-handling sugar is provided by the client library
time stamp
#Frame this data is associated with
# 0: no frame
# 1: global frame
string frame_id
```

There are three fields in the example header, seq which is id for the header. The stamp field will store information that deals with time related to the data that was collected (timestamp). The frame_id field stores all the frame information related to the data that was collected.

Namespaces

Namespaces in ROS are the best option for users when it comes to dealing with a name collision which is not a rare issue in robotics. This is even more true when a large and complex system is brought into question. For example, there could be a robot with two sensors streaming data for distance. One could be in the back, one in the front, from here we would have two topics streaming this data with identical values: **distance**

= 20 and **distance = 20**. In order for another node to distinguish between these two we could use the namespaces and give these nodes the names front/distance and back/distance. Now we are able to accurately read our data streams without any conflict.

Workspace

Catkin is the official build system of ROS and was designed to be more conventional for users. Catkin allows for better distribution of packages, better cross-compiling support and better portability. Build systems are responsible for generating targets from raw source code that users will be able to manipulate. Packages built in ROS are all built in a catkin workspace. There is a typical and recommended catkin workspace layout is available on the ROS wiki

```
workspace_folder/        -- WORKSPACE
  src/                   -- SOURCE SPACE
    CMakeLists.txt       -- The 'toplevel' CMake file
    package_1/
      CMakeLists.txt
      package.xml
      ...
    package_n/
      CATKIN_IGNORE      -- Optional empty file to exclude package_n from being processed
      CMakeLists.txt
      package.xml
      ...
  build/                 -- BUILD SPACE
    CATKIN_IGNORE        -- Keeps catkin from walking this directory
  devel/                 -- DEVELOPMENT SPACE (set by CATKIN_DEVEL_PREFIX)
    bin/
    etc/
    include/
    lib/
    share/
    .catkin
    env.bash
    setup.bash
    setup.sh
    ...
  install/               -- INSTALL SPACE (set by CMAKE_INSTALL_PREFIX)
    bin/
```

```
  etc/
  include/
  lib/
  share/
  .catkin
  env.bash
  setup.bash
  setup.sh
  ...
```

As seen in the example above, there are four different sections each having a different purpose. The source space contains all source code of the catkin packages.
The build space is where CMake is invoked in order to build the catkin packages that are present in the source space. The development space is where built targets are placed before installation. Lastly, within the install space, where the targets are built, they can be placed into the install space by invoking the ‘make install’ command.

Launch Files

Launch files are XML formatted files that end with the. launch extension that specify parameters to be set and nodes to be launched, as well as where they should be run. Using roslaunch one can easily launch multiple ROS nodes at once. The roslaunch package contains the roslaunch tools, which reads the roslaunch .launch/XML format. It also contains a variety of other support tools to help you use these files.

Many ROS packages come with "launch files", which you can run with:

```
$ roslaunch package_name file.launch
```

These launch files usually bring up a set of nodes for the package that provide some functionality.

ROS Packages

As the main goal of this project is to establish novel and efficient swarm navigation, many of the lower level tasks can be successfully solved with the use of packages so that more time can be spent on the actual goal of this project. This section contains all the ROS packages that were researched, their functions, and their application to the project.

Mavros

Mavros is the, “MAVLink extendable communication node for ROS with proxy for Ground Control Station” [3]. This is essentially a ROS package that handles all MAVLink functionalities. MAVLink is a communication protocol that is generally used to communicate between drones. It follows a modern publish-subscribe and point-to-point design pattern [4]. This package shows the potential to be useful in inter-drone communication, a topic that is at the core of developing swarm algorithms. In addition to this fact, many of the other projects that have been done in the area of swarm robotics feature this as a dependency, so if they are to be utilized, this package must be considered. In order to install Mavros run the following command [17]:

```
sudo apt-get install ros-kinetic-mavros ros-kinetic-mavros-extras
```

Micros Swarm Framework

Micros Swarm Framework is a full framework containing multiple drone swarm organization patterns with the goal of abstracting much of this work so more complicated patterns can be developed. From the developer, “This is a programming framework to facilitate application development involving robot swarms. It makes coding for swarms much easier by providing an adequate swarm-level abstraction, as well as tools for swarm management, various communication mechanisms and so on” [5]. This package shows potential in organizing our drones into patterns that would facilitate searching for and finding a target in an urban environment. Its use will be further looked into during implementation. In order to set up Micros Swarm Framework run the following commands [5]:

```
mkdir -p catkin_ws/src
cd catkin_ws/src
catkin_init_workspace
git clone https://github.com/xuefengchang/micros_swarm_framework.git
cd ..
catkin_make -j1
source devel/setup.bash
```

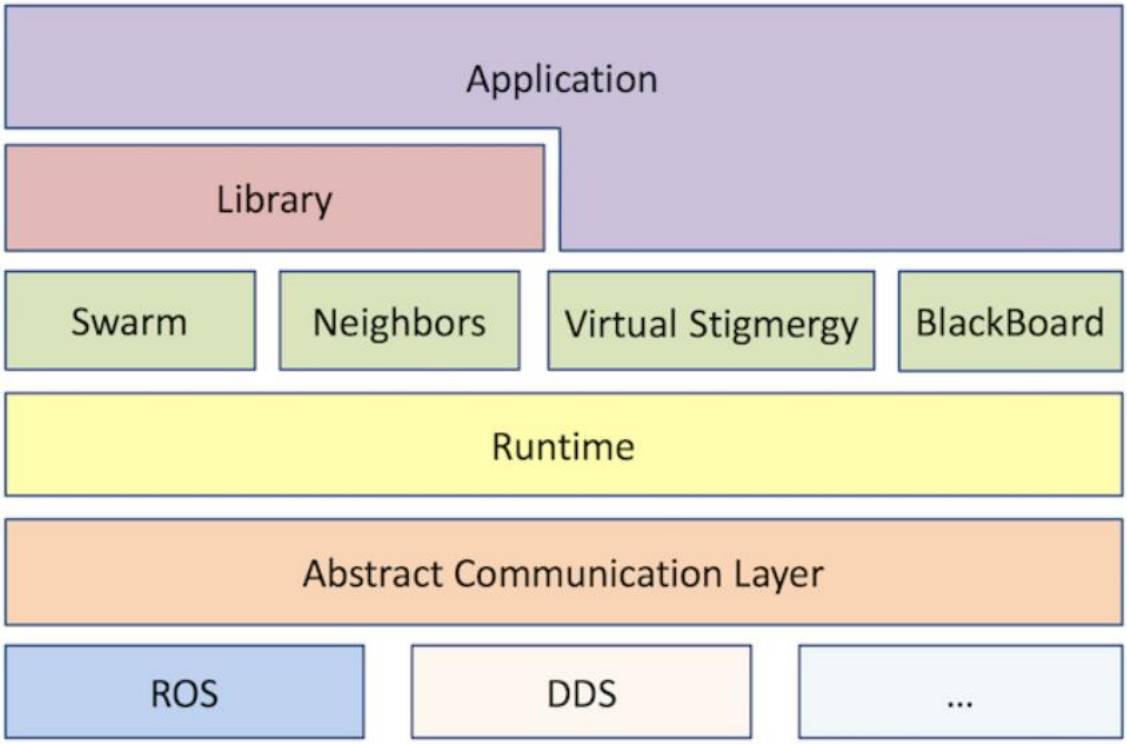


Figure depicting the architecture of Micros Swarm Framework [5]

Hector Quadrotor

This package includes an already model aerial drone model that comes with most features that we need for this project. It contains the full stack for all tasks needed for the drone including, “modeling, control and simulation of quadrotor UAV systems” [7]. Inside the stack is also included additional dependencies/packages that can extend the functionality of the Hector Quadrotor. These additional packages are listed below [7]:

- hector_quadrotor_description provides a generic quadrotor URDF model as well as variants with various sensors.

- hector_quadrotor_gazebo contains the necessary launch files and dependency information for simulation of the quadrotor model in gazebo.
- hector_quadrotor_teleop contains a node that permits control of the quadrotor using a gamepad.
- hector_quadrotor_gazebo_plugins provides plugins that are specific to the simulation of quadrotor UAVs in gazebo simulation.

This package would save us a lot of time by skipping over having to create our own drone and worrying about how to create the model, add the sensors, and launch it into Gazebo.

Not only does the stack provide all these packages for easily loading and controlling our model in Gazebo. There is also a hector_slam package that can run a SLAM algorithm and build a map for us using RViz. Using the tutorial from the ROS WIKI we can easily setup and run the SLAM algorithm.

Setup

Step 1: Install binary packages
The hector_quadrotor, hector_slam, hector_localization, hector_gazebo and hector_models packages are needed to run this demo.

The simplest option is to install the Ubuntu binary packages of hector_quadrotor available in the ROS package repository. To install all required packages for this tutorial run:

```
sudo apt-get install ros-hydro-hector-quadrotor-demo
```

Step 2: Launch the indoor SLAM demo:

```
roslaunch hector_quadrotor_demo indoor_slam_gazebo.launch
```

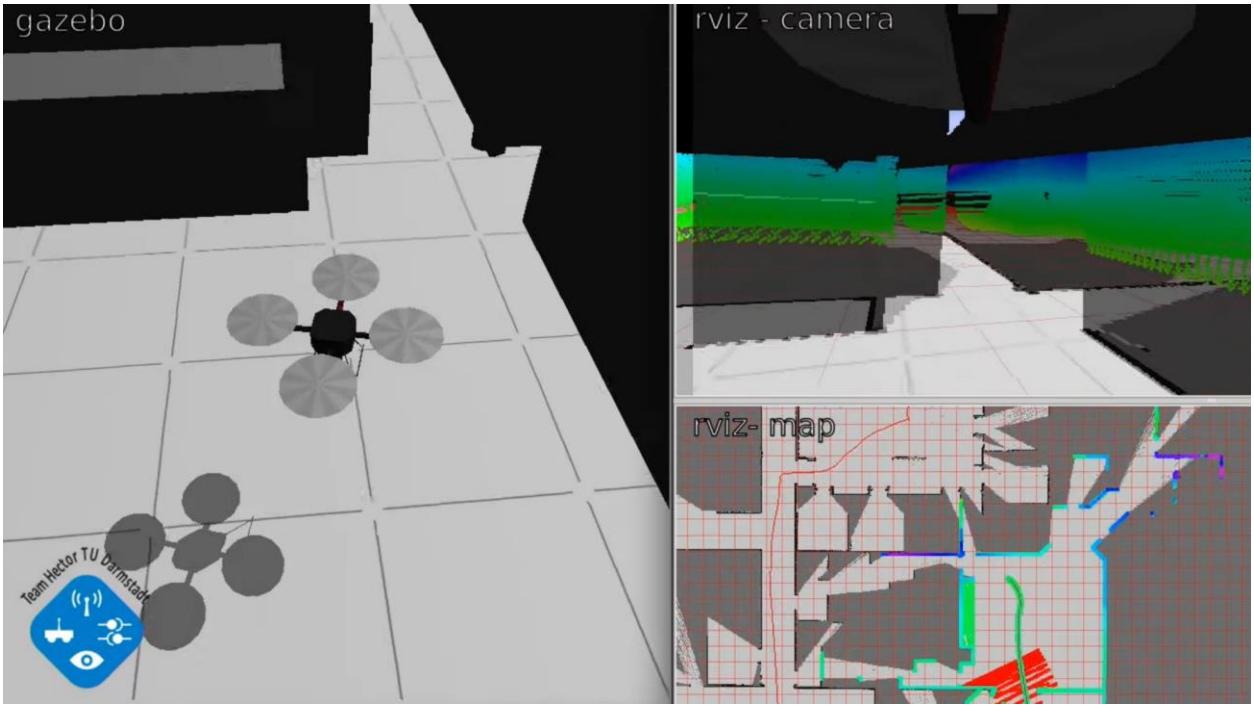
gazebo simulation as well as rviz visualization should start up now, and the quadrotor UAV should be on the ground.

Step 3: Control
Depending on the version of the package you use, you might have to enable motors via a service first:

```
rosservice call /enable_motors "enable: true"
```

The quadrotor accepts geometry_msgs/Twist messages on the 'cmd_vel' topic, as many other robots do, too. This of course also includes controllers for autonomous flight. If you have installed the teleop_twist_keyboard package, you can for example also use this:

```
rosrun teleop_twist_keyboard teleop_twist_keyboard.py
```



SLAM demo with the hector_quadrotor[43]

Hector Quadrotor Description

This package includes the model drone itself, which can be configured in 3 configurations. One configuration is the base Hector Quadrotor, one configuration is the Hector Quadcopter without any sensors, and one is the Hector Quadcopter with extra sensors mounted. From the documentation itself, “This package provides a URDF model of a generic quadrotor UAV. The visual geometry is provided as a COLLADA model, and the collision geometry is provided as a STL mesh” [8]. This package has potential to be the drone that we use for the simulation as everything seems to be here that we’re looking for.

Hector Quadrotor Gazebo

This package provides the gazebo component of the Hector Quadrotor. According to the documentation it provides these functionalities [9]:

- Colored COLLADA Quadrotor mesh model (.dae)
- URDF description
- Publishing of ground truth pose and simulated imu data
- Spawnable in gazebo
- Dedicated controller for use in gazebo simulation. The UAV can be controlled in gazebo using geometry_msgs/Twist messages on the 'cmd_vel' topic (as used on many other robots, but with added climb rate)

Running the drone in Gazebo is crucial for the performance of our drone swarm, so having the already completed Gazebo infrastructure for our drone is very useful. With that being said this package might be very important for the success of our project.

Hector Quadrotor Teleop

The Hector Quadrotor Teleop package provides support for controlling the Hector Quadcopter drone via a remote joystick allowing the drone to be manually controlled by a person. Since our project will be dealing with drone swarm technology via autonomous control of the drones, this package does not provide any use for the completion of our project.

Hector Quadrotor Gazebo Plugins

The Hector Quadrotor Gazebo Plugin package provides extra capabilities and sensors hector_gazebo_ros_baro sensor plugin to the Hector Quadrotor. According to the documentation these new features include a barometric sensor that can calculate an altimeter based on barometric pressure and," a simple controller allowing to command the quadrotor's velocity using a geometry_msgs/Twist message for teleoperation just by means of applying forces and torques to the model" [10]. Neither of these plugins seem important to the completion of the drone swarm, but they are included in the Hector Quadrotor stack.

RotorS Simulator

RotorS is another drone model package that can interface with Gazebo. It comes with three drones: AscTec Hummingbird, Pelican, and Firefly as well as multiple sensors that can be equipped on these drones. These sensors include an IMU, a generic odometry sensor, and the VI-Sensor. This package would be similar to the Hector Quadrotor in that it provides a drone model that can be loaded with sensors and imported into Gazebo.

Roslaunch

Roslaunch is a utility package that enhances the functionality of ROS. It accomplishes this by allowing the user to launch multiple ROS nodes locally and remotely. It requires one or more XML configuration files with the .launch extension that, “specify the parameters to set and nodes to launch, as well as the machines that they should be run on” [11]. The package contains a variety of tools to help support and use the .launch file formats. In order to run these launch files, the following command can be run [11]:

```
$ roslaunch package_name file.launch
```

The XML files in the launch files specify information needed to launch the ROS application. These include nodes that should be run, parameters that should be set, and other characteristics for ROS nodes. An example of a properly formatted .launch file see below:

```
<launch>
  <node name="talker" pkg="rospy_tutorials" type="talker" />
</launch>
```

Simple .launch XML file example [11]

This snippet shows a simple example of launching a single “talker” ROS node. As for this package’s application for this project, almost all the other packages listed under this section use Roslaunch in order to run their applications, so this package is critical to the completion of this project.

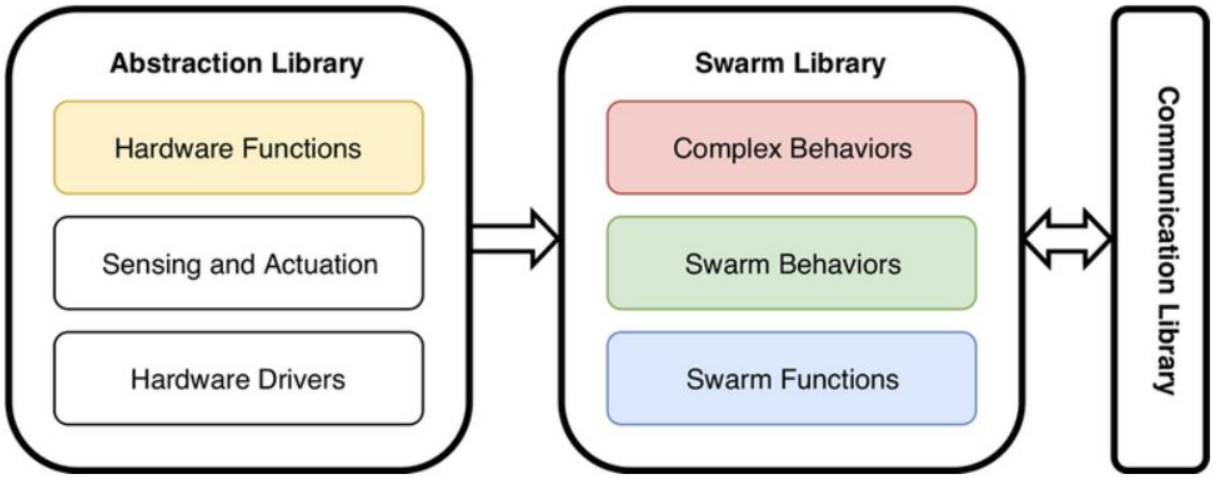
Catkin

Catkin is the build system that most ROS packages use and seems to be the go to build system for the ROS community. It is the successor to the old build system for ROS called rosbuild. From the documentation, “catkin combines CMake macros and Python scripts to provide some functionality on top of CMake's normal workflow” [15]. Catkin is an improvement over the previous rosbuild and allows for better package management, cross-compiling support, and better portability [16]. Due to its use as ROS’s main build system, this package is highly important for the success of this project. In order to install catkin run this command:

```
sudo apt-get install ros-noetic-catkin
```

Swarm Library

Similar to the Micro Swarm Framework, Swarm Functions is a ROS package that includes swarm functionality by abstracting away the low-level controls and communications between agents. This package contains three other packages: Complex Behaviors, Swarm Behaviors, and Swarm Functions. The documentation states, “Complex Behaviors contains complex behaviors for swarms of cyber-physical systems (CPSs)” [20]. Swarm Behaviors deals with the implementation of swarm algorithms. Swarm Functions works with swarm behaviors to enable swarm algorithms to work.



Architecture of the Swarm Library [21]

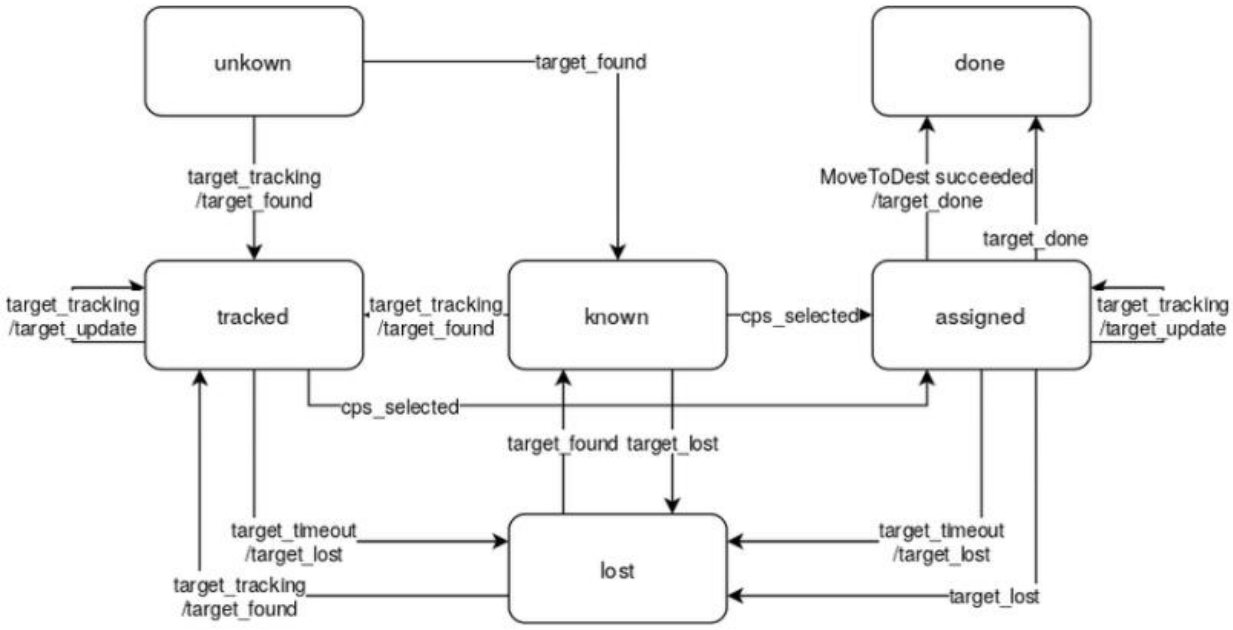
Area Division

This package is included in the swarm functions repository. Its functionality enables the swarm to divide the simulation area into multiple sections that will then be sent to each drone. Specifically, from the documentation we learn that this package, “divides the available environment area among multiple cyber physical systems (CPSs) in a swarm” [22]. The functionality of this project could be applied to this project in order to break up the simulation environment for faster and easier searching of the whole target area. This can be accomplished by assigning a drone to each new subarea. The area division package can be used by entering the following command:

```
roslaunch area_division area_division.launch
```

Target Monitor

The target monitor keeps track of information regarding the targets in a swarm. This is more useful for situations where multiple targets are in the area of operation, or the targets move from one drone’s sensors to another. This package allows for the swarm to communicate to each other where the target is and where it’s moving. Aside from this, the target monitor package might still be useful in this project to allow the swarm to signal each other once the target is found.



Flowchart depicting the functionality of the Target Monitor [23]

Slam Toolbox

The slam toolbox is an all-encompassing package dealing with everything 2D slam related. It includes the basic 2D slam functionalities such as start, map, and save pgm as well as additional functionalities. These additional functionalities include allowing for continuous refinement, remapping, or mapping a pgm and life-long mapping which allows for a pgm file to be loaded and continue to be mapped in a space while removing any detections that were overridden with the new scan. This package provides a lot of usefulness to our project. Firstly, it abstracts out the need to create our own slam algorithm. Additionally, it provides a lot of extra functionality that could become useful when we start to work on swarm algorithms and typing multiple slam mappings together that we receive from the multiple drones.

Darknet ROS

This package provides support for Darknet, which is the neural network framework behind the YOLO (you only look once) algorithm. YOLO provides fast and efficient image classification that we plan to use on our drone agents for object detection and target identification. This package allows for YOLO to be loaded onto the drone more easily and saves us from the trouble of having to download or recreate the YOLO algorithm from other sources, so this package should provide high functionality for the successfulness of our project.

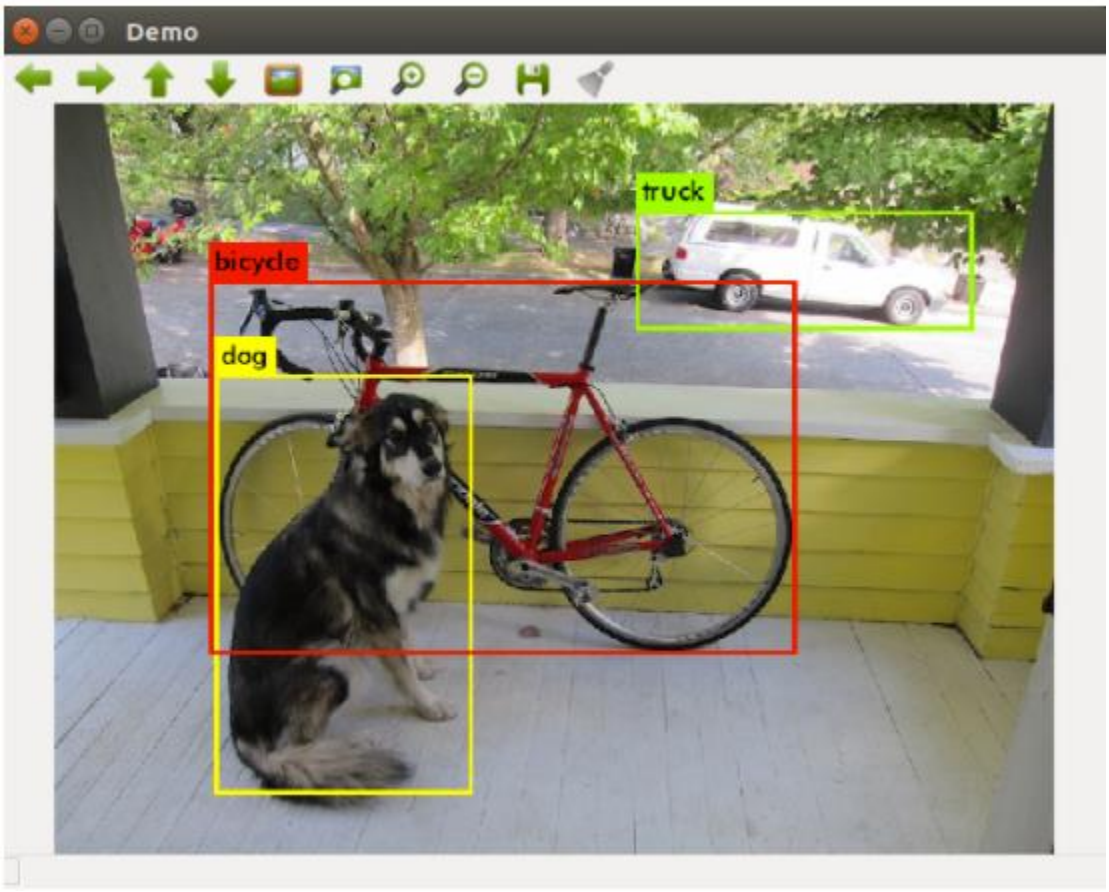


Image showing how Darknet ROS works[30]

UAV Optimal Coverage

This package provides functionality to divide the search area into sections and assign a drone to each section. The package will handle the drone communication and determine how the sections will be split up amongst the different drones.

MRS UAV System

This package provides us with five large drones that can be launched into Gazebo. As we need a drone that is large enough to fit a Lidar sensor as well as our communication gear and a camera, it's important that the drone we select has enough space to be able to fit all these sensors. All MRS UAV System drones are large enough to be fitted out with the appropriate hardware to be able to accomplish their respective roles, so this package could provide use for us in selecting the drone model we will use to run the simulation. An example of one of the drones included in this package is listed below:

Image of the DJI f330 [79]

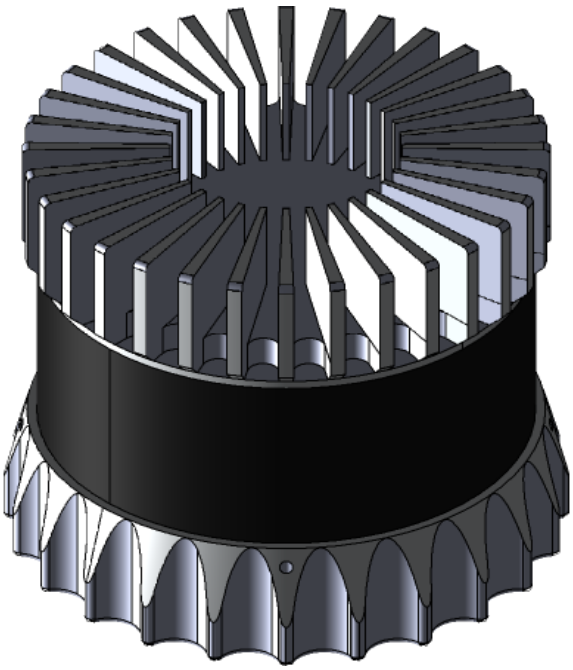
Drone Sensors in ROS

Being able to simulate different types of sensors in Gazebo is going to be a huge benefit as we are not required to go and purchase these expensive devices. These sensors can range in price anywhere from a couple hundred dollars to thousands of dollars. With gazebo we can test a range of different sensors and determine things such as the ideal placement or orientation of the sensor for its specific job or task. We will be able to switch different models in our software stack to make sure we find the best possible performing sensors.

Ouster OS-1

The Ouster OS-1 is one of the many different sensors available for simulation in Ros and Gazebo. It is a multi-beam flash lidar sensor manufactured by the company Ouster. The sensor comes in two different versions one with 16 lasers and the other with 64 lasers. It has capabilities of performing at 10 or 20hz and covers a full 360 degrees for each scan processed. The sensor has a wide range of 1M minimum and 150M maximum. Last but not least there is an Inertial Measurement Unit (IMU) on the sensor. This will be able to measure the drone's force, angular rate and orientation.
Ouster provides sample code for configuring the OS-1, reading data, and interacting with ROS. Specifically, the ouster_ros package contains sample code for publishing OS-1 data as ROS topics. The package contains the following ROS nodes:

- os1_node: The primary sensor client that handles the sensor initialization and configuration as well as publishing the raw IMU and lidar packets
- os1_cloud_node: Converts the raw IMU and lidar packets to ROS IMU and PointCloud2 messages
- viz_node: Runs the provided visualizer displaying the pointcloud and the intensity/noise/range images
- img_node: Publishes the intensity/noise/range images as ROS image messages



Ouster OS-1 Sensor [33]

Data is sent between the lidar and IMU coordinate frames which can be fed to the ROS driver to produce the ROS topics for data exchange. The data can be viewed with RViz, a 3D visualization tool for ROS. In order to use the sensor for simulation we will have to accurately describe the physical properties of it. This is where an SDF model comes in.

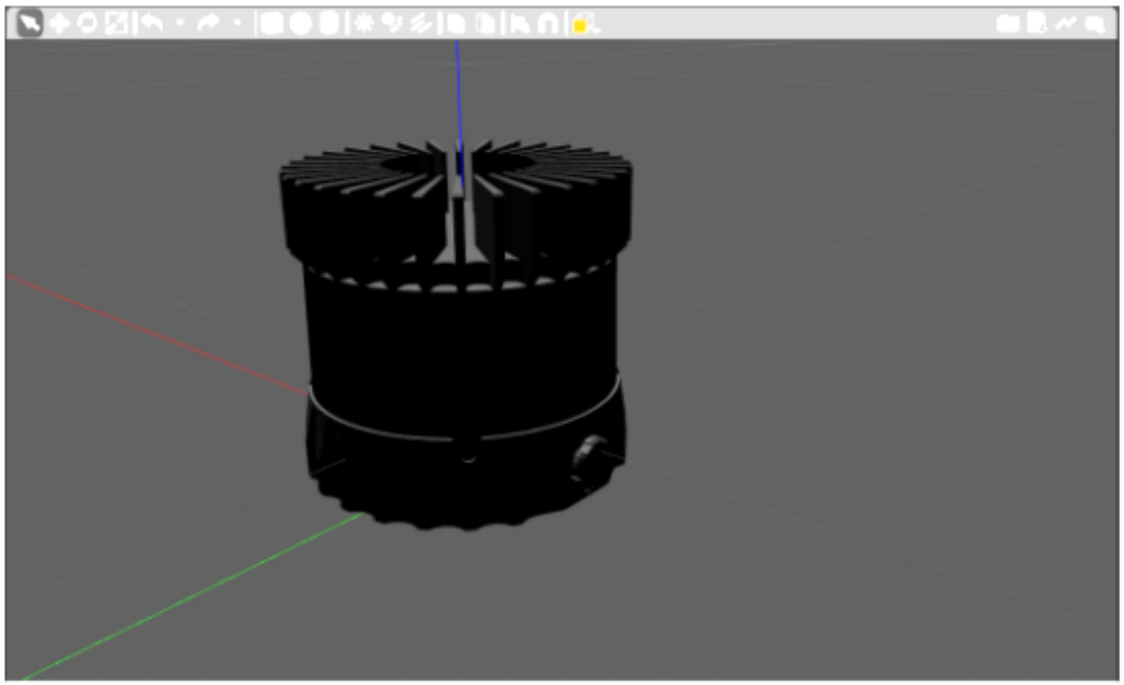
Creating SDF Models

SDF, or simulation description format models, are XML format files that describe objects and environments for simulators such as Gazebo. In order to use the sensor, the SDF model will be created in a .world file that will be run on Gazebo. First you would define the link properties of which include the inertial and collision fields. These values are going to depend on the physical properties of the sensor itself. This can be found in the manufacturing guide, and we input these parameters:

- Mass: 380g
- Height: 73mm
- Diameter: 84mm

The moment of inertia matrix can be computed using equations and are input into the correct field in the OS1-64.urdf.xacro file. After this the model should be able to be seen in Gazebo for additional debugging.

The next step would be to make the sensor look more realistic and not just a cylinder. This can be done using STL,OBJ or Collada files, the mesh file provided by Ouster can be used in Gazebo. To use this mesh file, update the <visual> element and replace the <cylinder> element within with a <mesh> element. The <mesh> element should have a child <uri> that points to the top collada visual.



Ouster OS-1 Sensor Loaded in Gazebo[33]

Creating URFD Models

After basic dimensions and visual appearance are now correct, we add the simulated IMU and lidar sensors. Take the core components from the SDF model and convert them to the OS1-64.urdf.xacro model. The universal robotic description format (URFD) models are XML format files used in ROS to describe all elements of a robot. To use a URDF file in Gazebo links are added for the IMU and lidar sensor. These frames contain the transformation data between the IMU sensor and the lidar sensor. The last joint element to be added is a joint between the sensor and the object it is attached to. After this all joints can be viewed in Gazebo easily by right clicking the model and choosing View->Joints.

Add the Sensor to the Model

A sensor’s purpose is to gather data from the environment. We will be using the gazebo_plugins/gazebo_ros_block_laser plugin. We will again reference the manufacturing guide to fill in the following data fields:

- Lasers
- Horizontal Resolution
- Field of View
- Minimum and Maximum Range
- Range Resolution

Given this information, the plugin element is added. This references the libgazebo_ros_ouster_laser.so plugin generated by the files in the ouster_gazebo_plugins package. This plugin publishes lidar data using the PointCloud2 message format on the /os1_cloud_node/points topic.

The OS-1 also contains an IMU. Using the hector_gazebo_plugins package to simulate the IMU sensor, a plugin element referencing the libhector_gazebo_ros_imu.so plugin can be added. This plugin publishes IMU data on the /os1_cloud_node/imu topic.

After the sensor is configured and launched you can see the output via Rviz. Since the sensor is configured using an URDF it can easily be attached to other robots for simulations.

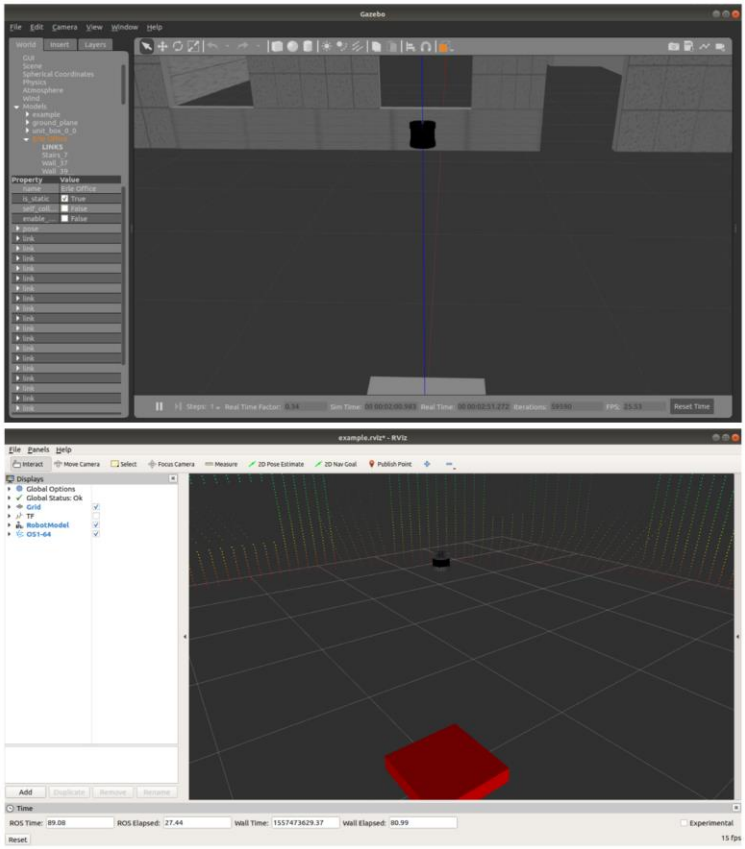


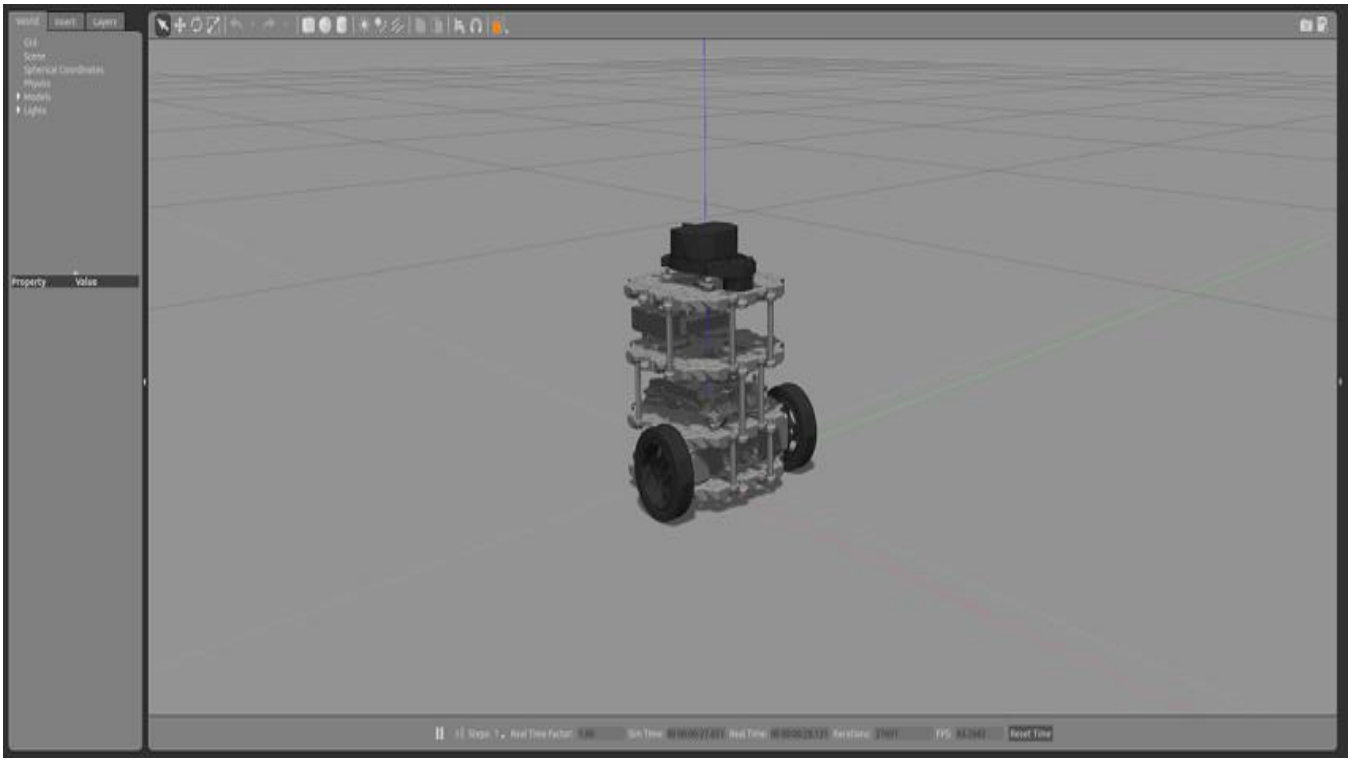
Image: Gazebo simulating the OS-1 (top)
The RViz visualization of the pointcloud data from the sensor(bottom) [33]

Drone Models

There are a number of robots within ROS with their own purpose and use case. The methodology of our search can be a large factor in what drone we should use. An example of this is one of the methods we considered was suggested by Dr. Leinecker. It was to have a fast drone that quickly scans through the environment and finds any object that looks at anything that moderately looks like the target and marks it. The second slower drone will double-check the fast drones' potential targets and confirm whether the target is the correct. For this method, the fast drone should probably be a light-weight drone holding a lower-resolution camera and a LIDAR sensor for mapping. The slow drone should be a larger drone that is generally stable and should have the ability to hold a high-resolution camera.

Ground Based

Ground based agents could be used in unison with our aerial agents to achieve our goal of finding the target more efficiently. One model that is compatible with our software would be the TurtleBot3. This robot was designed to easily teach those who are new to ROS and has become one of the most popular among developers. “TurtleBot3 is a small, affordable, programmable, ROS-based mobile robot for use in education, research, hobby, and product prototyping. The goal of TurtleBot3 is to dramatically reduce the size of the platform and lower the price without having to sacrifice its functionality and quality, while at the same time offering expandability. The TurtleBot3 can be customized into various ways depending on how you reconstruct the mechanical parts and use optional parts such as the computer and sensor. In addition, TurtleBot3 is evolved with cost-effective and small-sized SBC that is suitable for robust embedded system, 360-degree distance sensor and 3D printing technology.” [35] The TurtleBot3 is capable of running SLAM algorithms to build maps of its environment. We will have to run tests to see if using the TurtleBot in unison with our aerial drones proves to be beneficial or not. But for now, it has been good for learning ROS and our environment.



TurtleBot3 rendered in Gazebo [36]

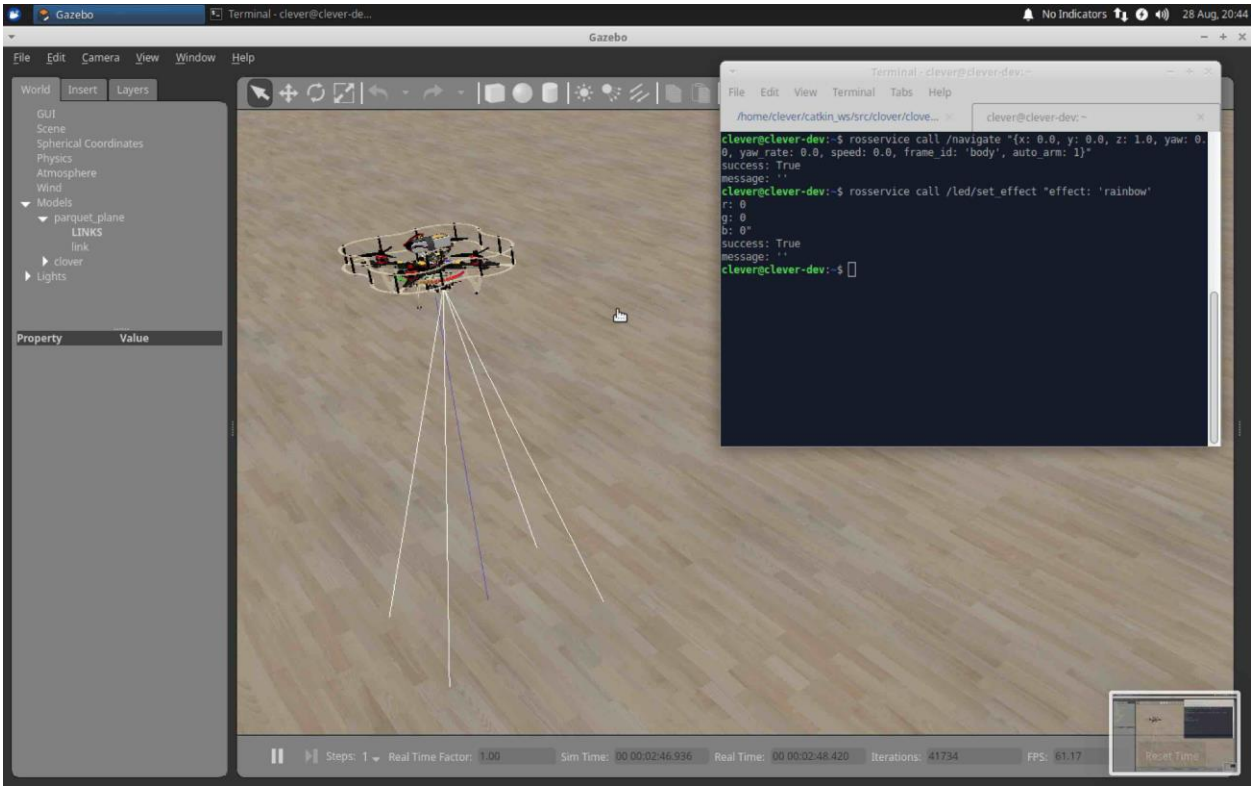
Aerial

Our main objective is to assemble a swarm of autonomous drones with Aerial agents. From the available documentation on the ROS robots wiki we had a few choices of drones that supported ROS and gazebo. As we need to mount a lidar sensor and a camera drone, we have to consider drones that are larger and have enough space to mount the needed hardware. The first option was the Gapter manufactured by Gaitech EDU. Gapter fully supports the MAVLink protocol which allows for its control and monitoring through MAVLink ground stations such as QGroundControl. Using this ground station, controlling autonomous missions of the drone would be easily done. Gapter EDU was designed for research and education in mind. Gapter comes with robust documentation and software packages that allow users to easily begin a project with a Gapter drone.

Gapter drone [76]

Another model that was on the ROS robot wiki was the COEX Clover. The Clover has a large library of open-source software and documentation. The Clover runs off a PX4 based firmware meaning it is MAVLink compatible with QGroundControl just like the Gapter. There is an already installed camera for navigation as well as additional

sensors and peripheral devices. The Clover has ROS based software packages for autonomous flights. As well as a full tutorial in documentation for simulating in Gazebo. Both seem to be good options. It's going to have to come down to speed for our final choice of drone when our final presentation is due.



A simulation of the COEX Clover in Gazebo [80]

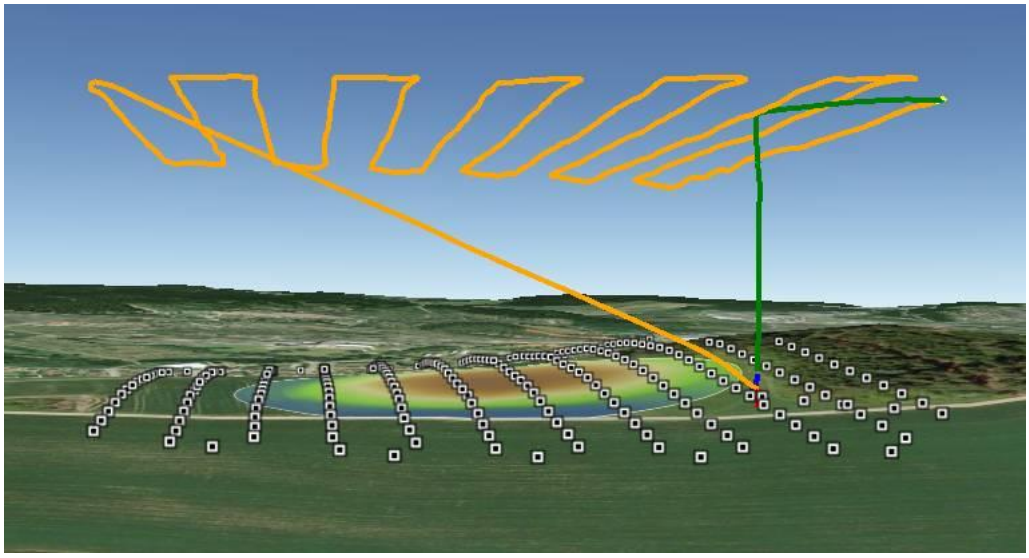
5.2 Ardupilot

ArduPilot is an open-source software that aids the creation of an autonomous piloting system for all different kinds of drones. The different drones that this software provides frameworks for are ground drones, ArduPlanes, ArduHelicopters, and multiple different drone types. The software has support for simulation tools and analysis with autonomous drone training. The workspace for our project is ROS and Gazebo, and this works perfectly with ArduPilot.

3D Mapping using ArduPilot

The mapping algorithm for ArduPilot works for both the ArduPlane drone and any of the ArduCopter drones that can have a still camera sensor installed. The altitude does not matter as the algorithm can work from any height; however, the suggested range is 25 meters to 100 meters. By aiming for overlap in the pictures taken the accuracy of the 3D mapping is increased dramatically. The suggested overlap amount is about 80% between pictures, and this can be achieved by maintaining a uniform spacing between each of the different parallel tracks. The cameras that are installed on the drone will need to be taking photos every few seconds to ensure the most efficient mapping is made. There are different commands that an ArduPilot mission will have that need to be implemented.

- TakeOff - The user will input a number for the altitude that they want the drone to fly or hover at and then the drone will climb to that given altitude and begin its mission.
- Waypoint - The waypoint command will be entered to instruct the drone where to take photos at. The best method to map an entire area is by using a grid structure. By changing the value of the “decay” column to a 1 the drone will stop over each of the waypoints during the picture taking process.
- Do_Digicam_Control - Used to manually cause the camera to take a photo regardless of whether it is at a waypoint or not.
- Return_To_Launch - The last command given to the drone. This will cause the drone to return to the launch position.



[12] Mapping using the ArduPilot Software Example

In our project utilizing this software can help with the swarm navigation aspect as the drones will need a way to map the environment that they have already been in. By simply running this software in parallel with the machine learning algorithm that will be performing object detection, an effective solution may be found.

Multiple Vehicle Flying

The ArduPilot software has packages that make multi-vehicle flying a lot simpler. The material requirements for the drones are not too extensive either, all of the items are within scope of this project. The primary equipment that each of the drones that will be flying together must have been some form of antenna that can receive an RC signal from another drone, RC transmitters to transmit to the other drones, and some interface or tool used to display and or control each of the vehicles. Controlling the vehicles is done after the mission planner has been made. This mission plan includes loading all of the extra algorithms and or systems that the drones may be using. During the actual mission the drones will be manually controllable, and their sensors will be visible to the user in order to make sure that everything is working properly. The main sensors that the ArduPilot software will be using are the antenna sensors. Each drone will use an RC transmitter to emit a signal from their current position. The other drones flying will then intercept this signal with their antenna sensors and be able to track the positions of the other drones. This method is very effective and will be very useful in this project, as

communication between each of the drones is one of the main objectives we are trying to accomplish.

Mission Planner

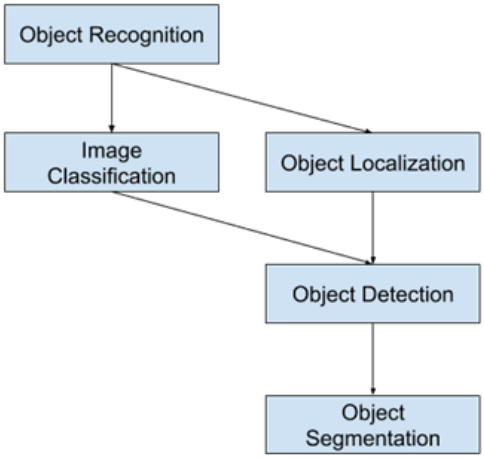
The mission planner is the user interface that is included with the ArduPilot software and is how all of the missions are carried out. The interface includes multiple things that are helpful for coordinating projects. Whether it be a plane mapping out the flat area of a farm to a drone mapping the layout of a city, the ArduPilot planner is very useful. In our project, we are using Gazebo to simulate an urban world. The planner will use the info from the world to create a mapping of the area from scratch, just as the drones would do in an actual real-world environment. Once the drones' positions and everything is loaded the other algorithms are loaded. By using the terminal that is included with the software a ROS environment is created and the machine learning algorithms are loaded onto the drones. This allows them to simultaneously map out the environment while also being able to identify the target should they come across it. The user interface also shows the camera view of each of the drones as well as their positions on the mapped environment. An example of how the user interface looks is seen below.

[14] Mission planner UI example

As you can see, there are many different options that the software provides. Almost everything in the mission can be done solely on this interface, such as any configurations that need to be changed and if any of the gauges need to be checked and calibrated.

5.3 Machine Learning Algorithms

The mission of our drone is to find a target robot placed randomly somewhere in our environment. But how will it know what exactly he is looking for? We will have to train our drone with deep learning techniques and use computer vision to be able to identify the objects in its view, more specifically the target robot. Image classification involves predicting the class of one object in an image. Object localization refers to locating an object in an image and drawing a bounding box around it. Object detection is a combination of both these tasks it locates as well as classifies objects in an image. I am going to go over someone at the top performing deep learning models.



Overview of Object Detection [29]

Machine Learning Overview

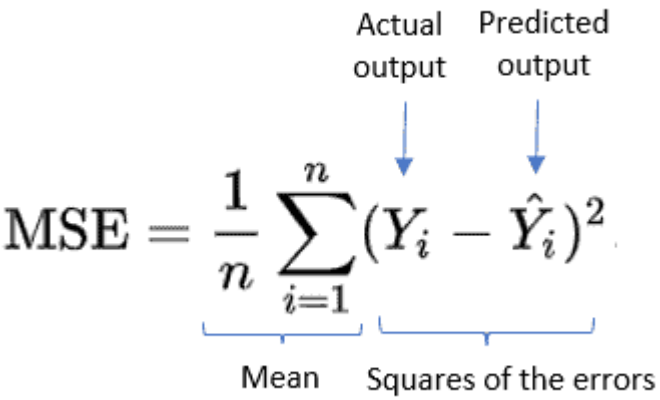
Machine learning is a computer science technique that is used for a countless number of applications and programs today. A large number of things can be broken down into quantifiable values. An example I am going to use to explain it is a hurricane. A hurricane has certain attributes or features that can be used to describe it, such as wind speed, position, and other variables that all are attributed to it. As humans we can use

the data from past hurricanes to predict where the new ones will travel and whether or not they will increase or decrease in strength. Predicting something based on a given set of input values is exactly what machine learning is. Utilizing the computing power of computers, mathematical algorithms and calculations can be made to “train” a machine learning model to be able to predict values. By initializing a given number of starting variables, also called weights, we use them alongside the actual values of the training dataset. The model will use the weights to make a prediction, and then based on the prediction that is made the weights are adjusted to make the next prediction more accurate. The model continues this process of updating the values used in the equation to predict more accurately and this is the foundation of how machine learning works. There are many different ways to change the equation used, how the weights are updated, and other factors that impact how well the model performs the task at hand.

Regression Vs Classification

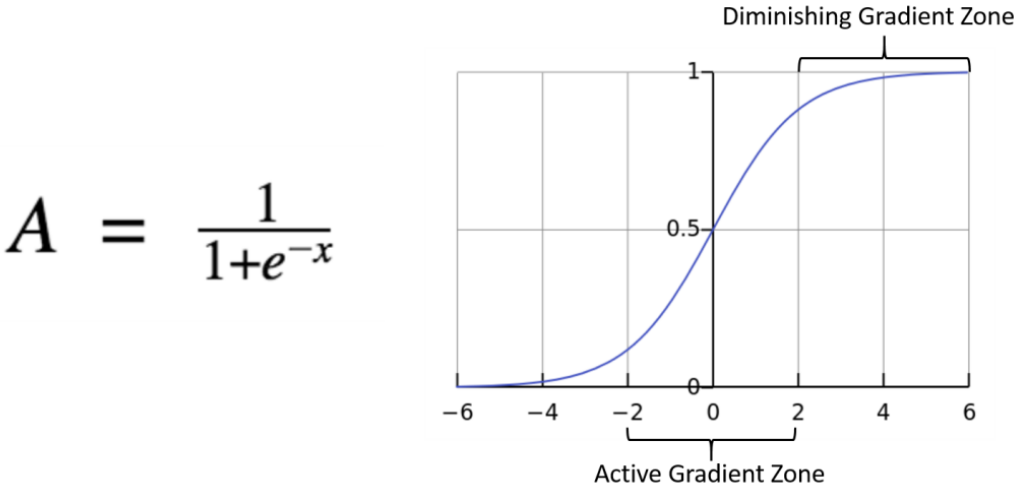
For machine learning models that predict, there are two major techniques that are widely used for two different reasons.

Regression - Used to predict a singular value that is either continuous or is the output based on a given set of inputs. For example, let's say you have a lot of data on the houses in some area. The data may include important factors about each house such as the size, location to schools, and other important features. You could use a regression model that will train an equation based on these values and produce a price value prediction based on them. A common equation that is used to perform regression problems is a linear one which produces only one output based on the input set. The predicted values are then used in a loss function that determines how well the model did. The most common loss function for regression would be Mean Squared Error (MSE) that takes each predicted value, subtracts it from the true value, and then squares the result and adds it to the running total for all of the predictions. How much the weights are changed depends on the gradient that is calculated based on this loss function.



[18] Mean Squared Error Equation Used in Regression

Classification - Used to classify a single object into one class based on a list of classes. An example of classification with a dataset of images that contain pictures of different articles of clothing, the model is trained to predict and classify the article of clothing in the picture. A common algorithm that is used for classification is actually a form of regression. Logistic regression is used to create a bounding line that can be used to classify objects based on where they fall in relation to the line. A very popular line that is used for logistic regression is the sigmoid line.



[19] Sigmoid Function Equation and graph

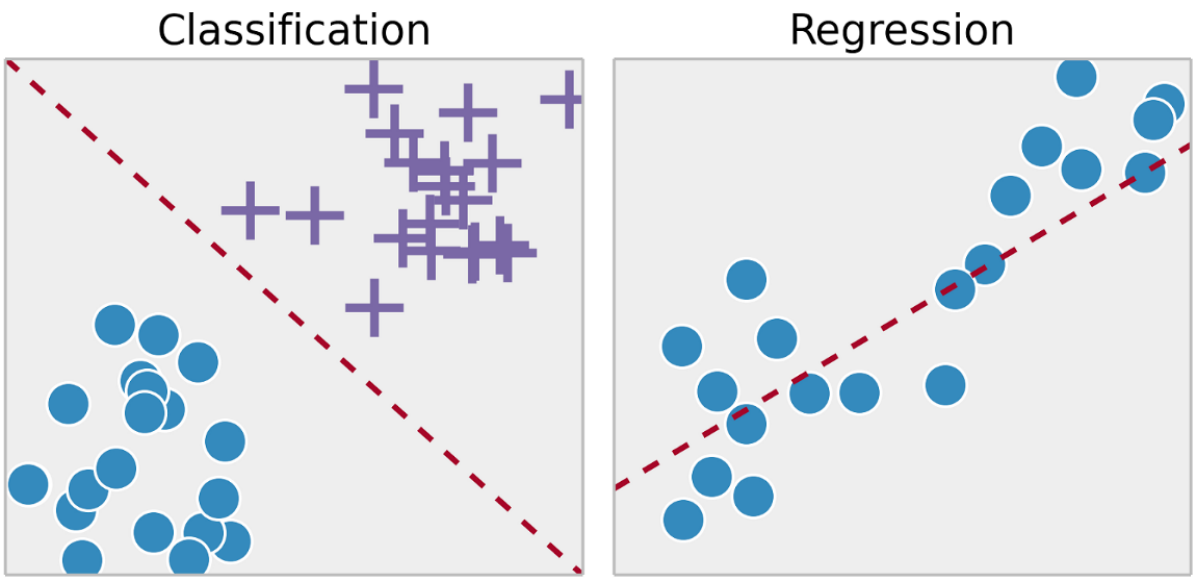
As a model is trained and the weights of the equation are updated. The number of classes is split into different positions from the line. A cost function is used in

coordination with this line in order to make predictions. This loss that this function produces is then minimized the same way that a regression equation is minimized, with a gradient. The cost function used in logistic regression is the binary-cross entropy, also known as log loss, function that does not produce the linear line that regression does, which allows the line to be more flexible based on how much the features impact the classification process.

$$H_p(q) = -\frac{1}{N}\sum_{i=1}^{N} y_i \cdot log(p(y_i)) + (1 - y_i) \cdot log(1 - p(y_i))$$

Binary Cross-Entropy / Log Loss

[19] Binary Cross-Entropy Function

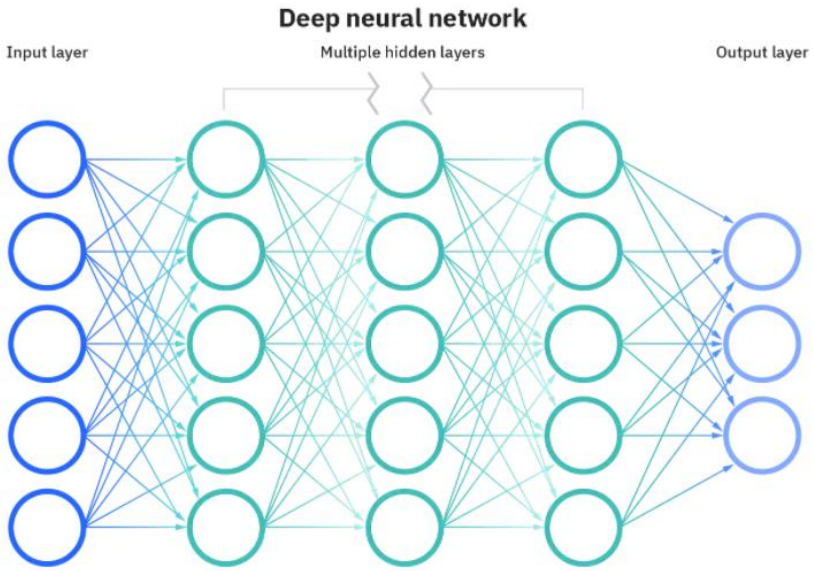


[70] A Depiction of Regression Vs Classification

Neural Networks

A relatively new technique that has resulted in the best performing machine learning and artificial intelligence systems are neural networks. These neural networks arose out

of the need for more complicated models to solve much more complicated machine learning problems, especially in the area of image classification, object detection, and speech recognition. Much like traditional machine learning, these models rely on training data in order to learn how to improve their accuracy over time. Generally neural networks are composed of input and output layers with hidden layers between the two. Each node is connected with a weight and a threshold that are changed during training to try and get the best performance out of the model.



A sample Neural Network [25]

The weights of the different nodes assign the importance of that node. The larger the weight the more impact that node has on the predicted value. These weights are then multiplied by the input and whatever the output is will be passed through an activation function that will determine if this node contributes to the output or not.

$$\text{output} = f(x) = \begin{cases} 1 \text{ if } \sum w_1 x_1 + b \geq 0 \\ 0 \text{ if } \sum w_1 x_1 + b < 0 \end{cases}$$

Example of an activation function [25]

In order for the neural network to learn, it must have a way to measure its accuracy, or in actuality its error. This is where a loss function comes. The loss function calculates how much the model's predictions were off versus the ground truth values and changes the weights in such a way that the loss function is minimized. The way this is done is

through changing the weight in such a way that they will converge to the minimum on the loss function's graph. See the visual below for a visual cue of how this works.

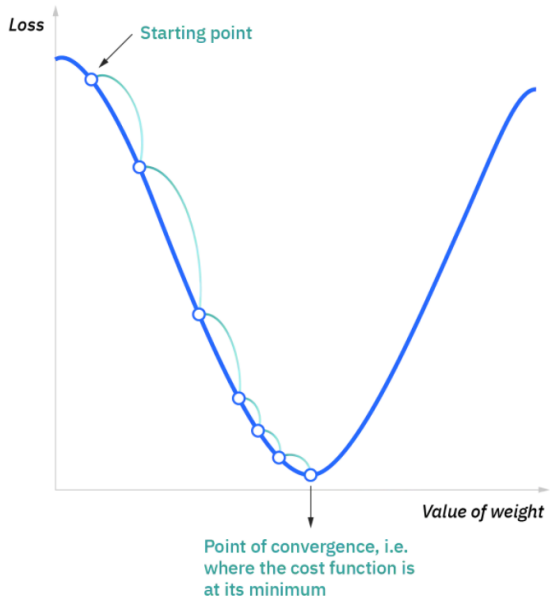


Figure showing the graph of the los function [25]

There are also many variations of neural networks with differing use cases. A few different types are listed below:

- Multi-layer perceptrons (MLPs)
- Convolutional Neural Networks (CNNs)
- Recurrent Neural Networks (RNNs)

Due to the complexity of our project, neural networks will be utilized extensively in object detection to allow our drones to detect the target. Specifically, we will use YOLO which is an algorithm based on a CNN to detect different objects around the world and detect the target.

Bounding Box Regression

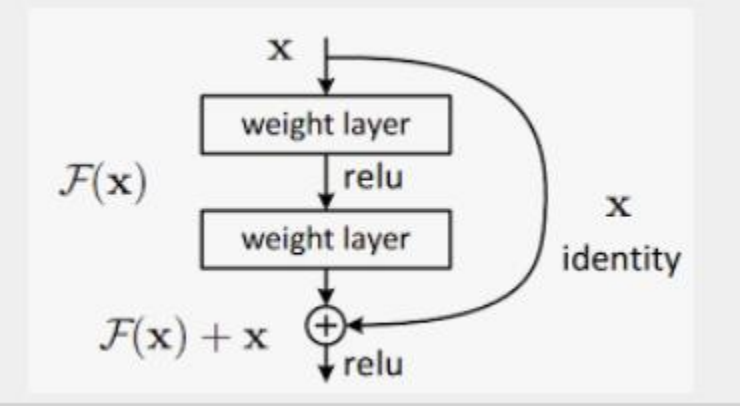
An algorithm that can predict and break down a picture into regions and objects is bounding box regression. Using labeled images that indicate a given area where an object is, machine learning models, such as the R-CNN network can train itself on the images. Passing the images through the filters and layers allow gradients attached to a given function to be constantly calculated and used to update the weights of a model. The pixels of an image are processed through each layer which allows for a process to be made for when there are similar pixels between images. Using images that already have a bounding box labeled for the machine is how the algorithm calculates the weights and is how the function that the model uses is formed. By performing this process of training a model and making the function as accurate as possible for detected objects in images using pixels, new images can be predicted upon, and this is how object detection using machine learning works.

Residual Blocks

Residual blocks are a twist on how traditional neural networks work. The way that a normal neural network works is by feeding one layer into the next. The difference with residual blocks is that each layer contained in the model is connected and links to a layer that is more than one layer away. In an average neural network, there are multiple problems that may arise while approximating the given target value for some problems, meaning that they cannot work for any problem. One of the problems that arise would be the vanishing gradient problem. This occurs when a gradient is calculated and is so small that the updated change to the weights is almost non-existent. This problem can even lead to the neural network to stop training all together and completely ruin a model.

Another problem that can occur in a normal model would be the curse of dimensionality where the sample size needed for accurate results grows exponentially. Increasing the number of layers in these models would seem like the logical next step, yet it has been proven that as more layers are added the accuracy that the model produces will improve slower and slower up until the point where it may actually get worse. In order to solve the problem that is faced, a system of skipping layers was formed. In order to achieve the same level of accuracy as the smaller networks while also allowing for the learning of unique and different functions for the model, these residual blocks have proved to be useful. There are many different ways to perform this type of training, whether that is done by changing the number of layers being skipped at a time or

finding which layers are skippable, the combinations are endless and can only lead to making a better model. In object detection residual blocks help to break down a picture into a grid, allowing for the individual detection of different objects inside of a single picture. Residual blocks play a key factor in R-CNN networks.



Example of residual block [75]

Convolutional Neural Network (CNN)

Convolutional neural networks form part of a deep learning algorithm which takes an image as an input and applies weights and biases to differentiate one from the other. In other words, convolutions are a way to capture information about the ordering of pixels. There are different parts that form a convolution like the kernel and pooling. CNN will be important to our project since most of the image classification or recognition uses it.

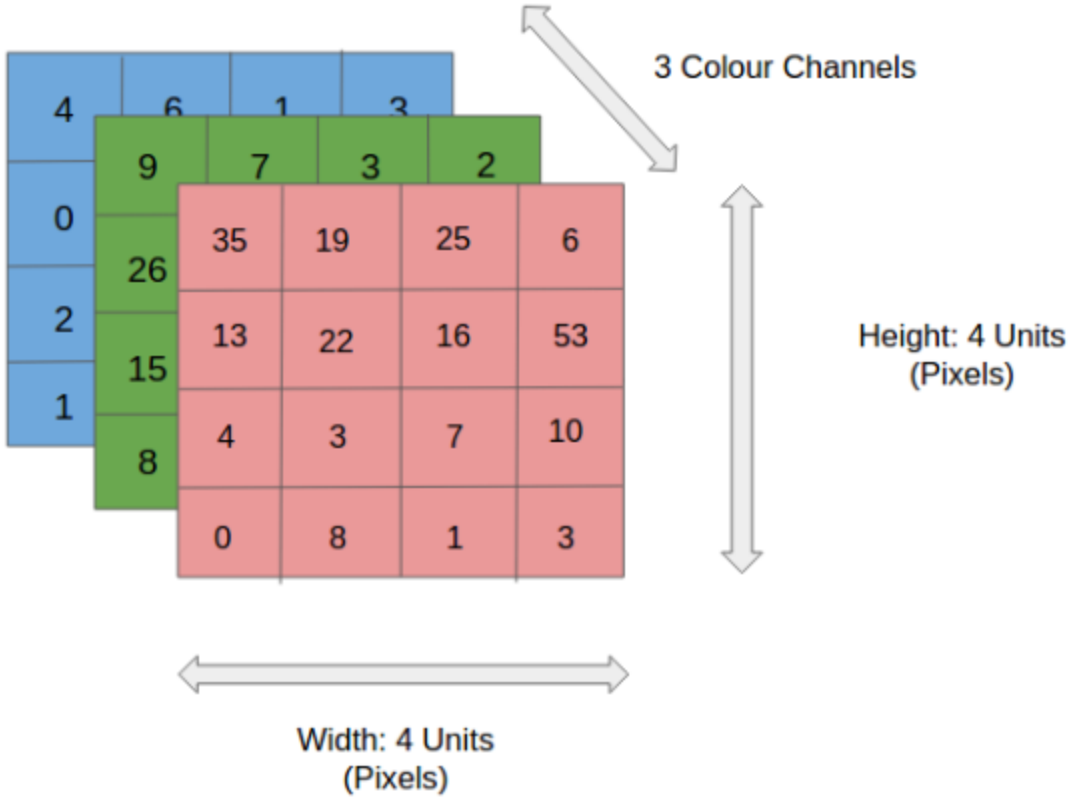


Figure: RGB Image Pixels[55]

The convolutional layer is the core building of a convolutional neural network and is where most of the computation operation occurs. The layers are composed of the input image, a filter/kernel, and the output array.

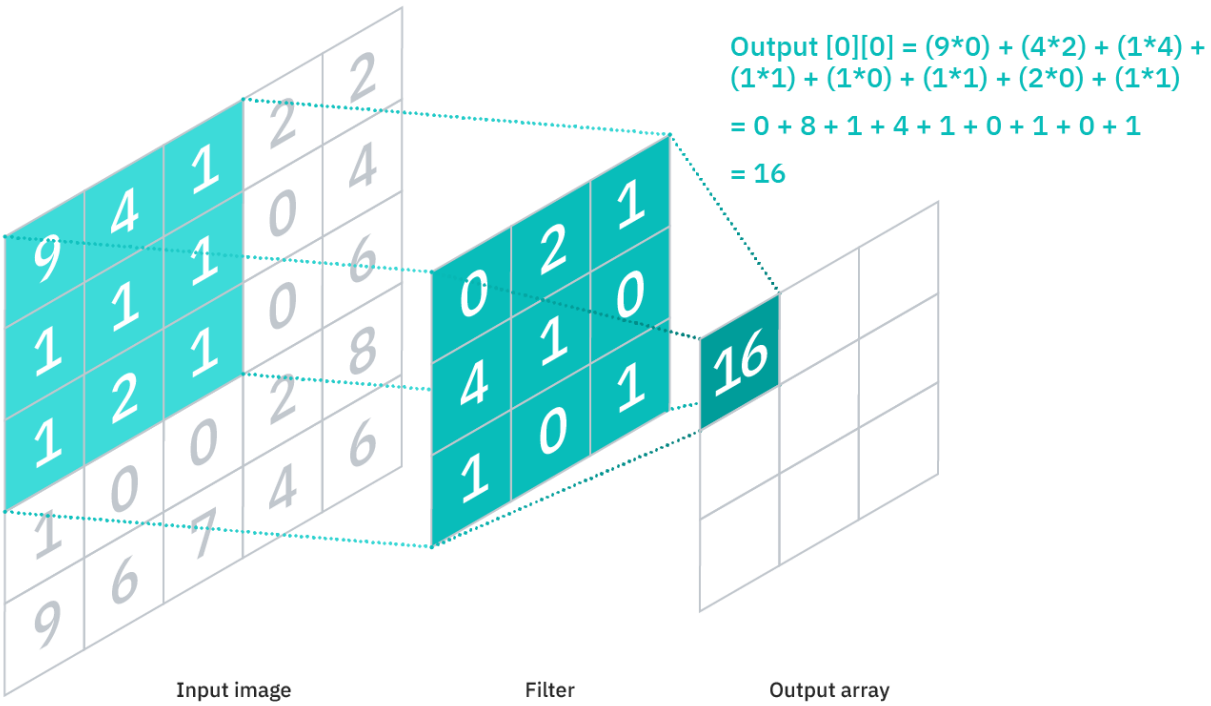


Figure: Convolution Layer[55]

There are three hyperparameters which might affect the output array in terms of size and that needs to be set before the neural network training actually begins. The parameters that can affect this out array are the number of filters which is also known as the kernel, the stride which can be seen as a distance or number of pixels, and also there is the padding which is usually used when the filters do not fit the input.

After finishing each convolution operation, one of the activation functions is applied, mostly known as ReLu or Rectified Linear Unit.

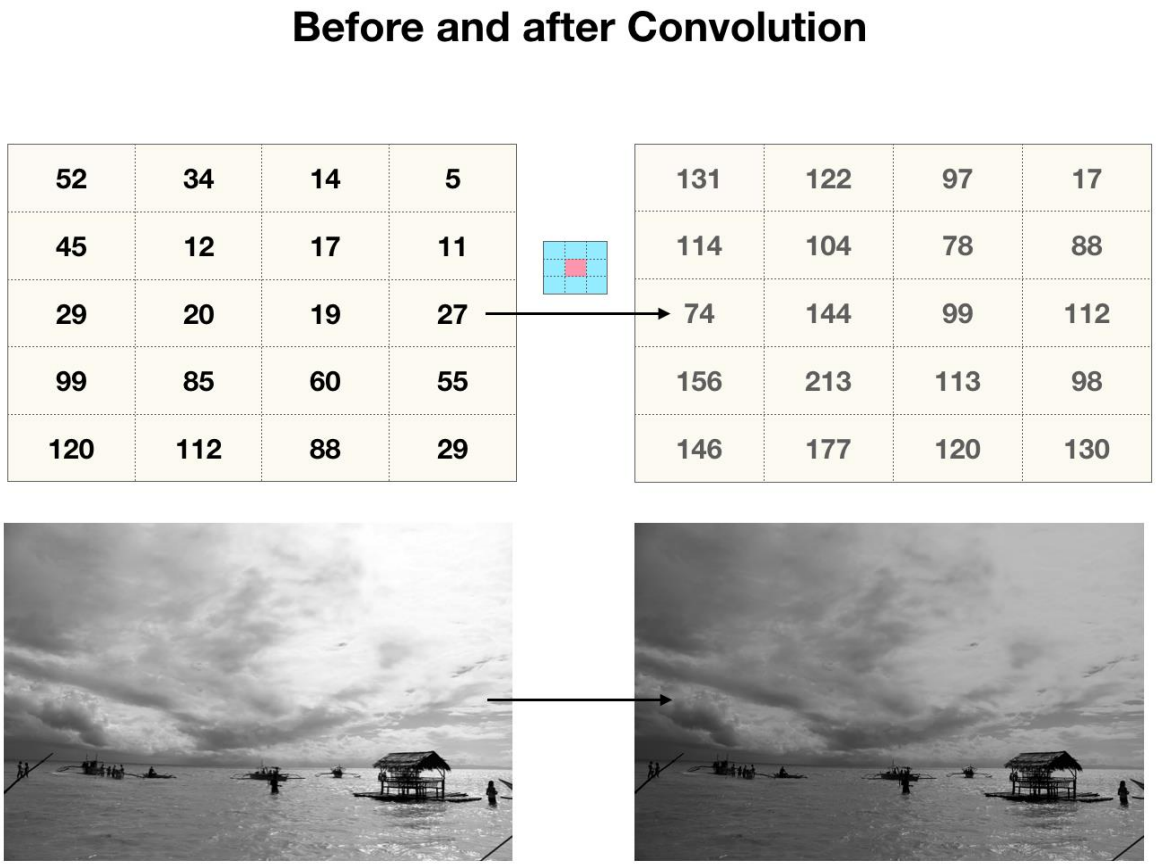


Figure: Convolution Operation[54]

We also have the pooling layer, which is similar to the convolutional layer, but pooling is mostly responsible for reducing spatial size. By reducing the spatial size, the computational effort that requires processing the input data will be reduced dimensionality.

There exist two types of pooling which returns different values. We have Max Pooling and Average Pooling. Average Pooling returns the average values of the input image covered by the filter/kernel. Max Pooling on the other hand returns the maximum values of the input filter, it can also use noise suppressant. Usually, Max Pooling in terms of performance does a better job than the Average Pooling.

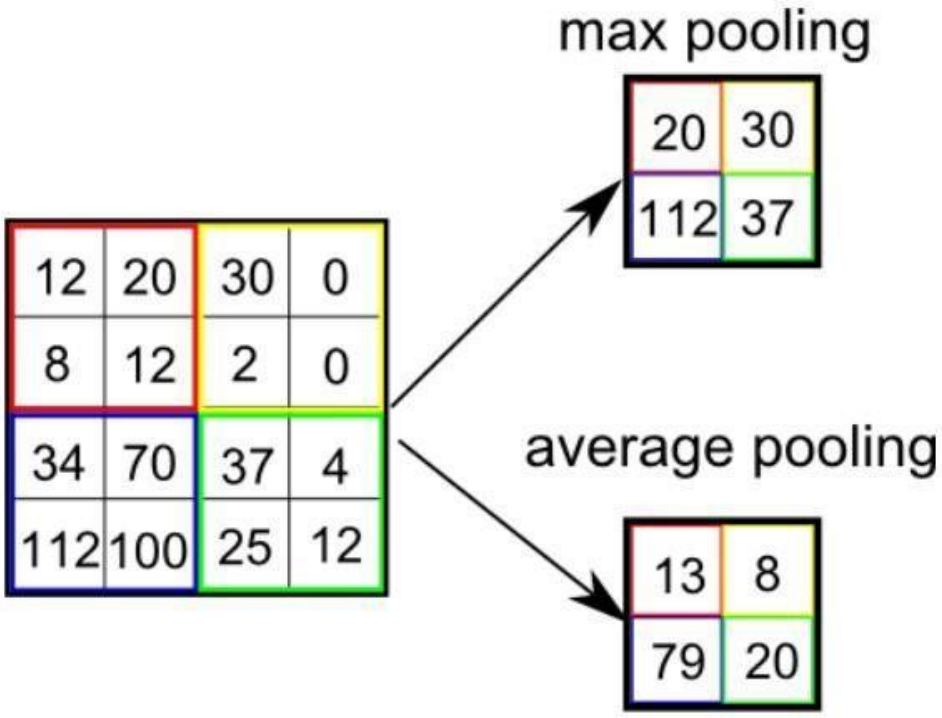
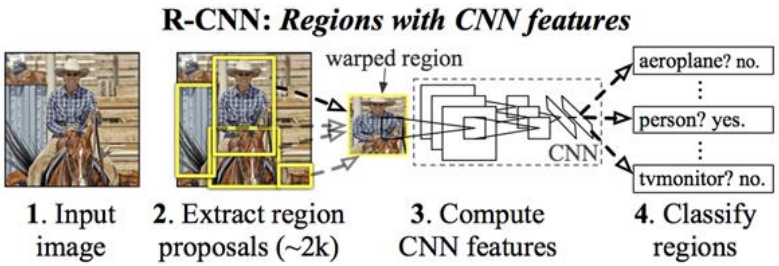


Figure: Types of Pooling[49]

R-CNN

RCNN is an acronym for region-based Convolutional Neural Network. It was first described in 2014 by Ross Girshick from UC Berkely in his titled paper “Rich feature hierarchies for accurate object detection and semantic segmentation.” The RCNN model was composed of three different modules they are:

- Region proposal. Generate class independent region proposals, bounding boxes
- Feature extractor. Extracts features from each region proposal using a convolutional neural network
- Classifier. Classifies features as one of the known classes



Example of R-CNN Classifying Objects [29]

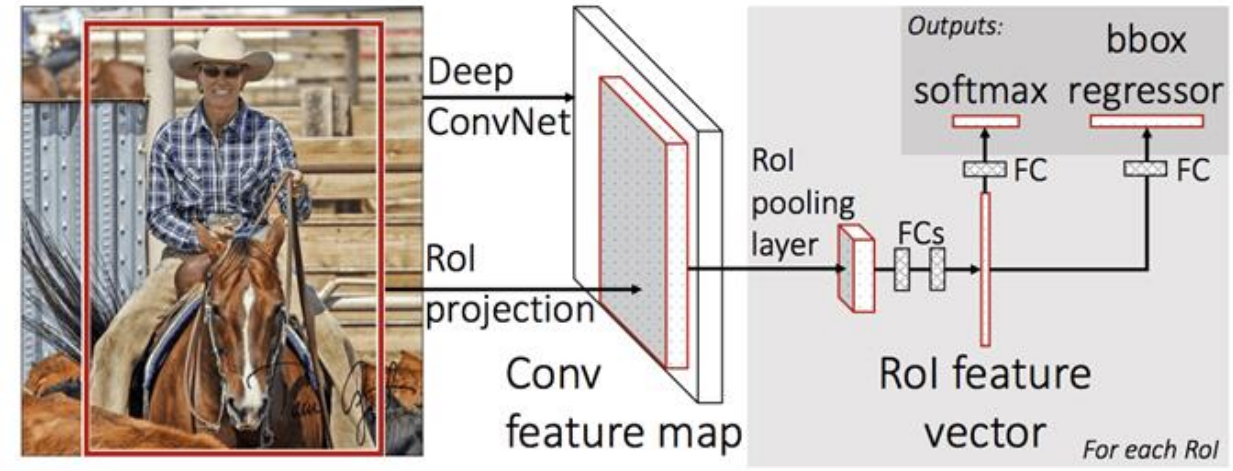
Summary of the R-CNN Model Architecture Taken from Rich feature hierarchies for accurate object detection and semantic segmentation.

This R-CNN model uses concepts such as bounding box regression and residual blocks in order to perform this object detection. It involves breaking down a picture and training a model in order to be able to make predictions on new images. However, it was soon discovered that there were issues with the model that was constructed. The problem with R-CNN was that it was slow, it required a CNN-based feature extraction pass on each of the proposed regions made by the region proposal algorithm. This is a problem because images could sometimes create upwards of 2000 proposed regions at test time. All this leads to a slow process for making predictions. The speed however was addressed in the next iteration of RCNN, Fast RCNN.

Fast RCNN

With the great success of RCNN Ross Girshick added an extension to fix speed issues with RCNN and gave life to Fast RCNN in 2015. The main issues looking to be solved here were, training involved the preparation and operation of three separate models, training a CNN on so many region proposals per image was slow and making predictions on so many region proposals was slow.

In Fast RCNN, the input image is given to the CNN, which will generate a feature map. The regions of interest (RoI) are then taken from this feature map and the output of the CNN is then interpreted by a fully connected layer. The fully connected layer classifies objects within the ROI's using a SoftMax layer and generates a bounding box. This process is then repeated for each region of interest in each image.



Summary of Fast R-CNN model Architecture [29]

The model was much faster to train and make predictions, but still requires a set of candidate regions to be proposed along with each image given and can slow down with a large data set.

Faster R-CNN

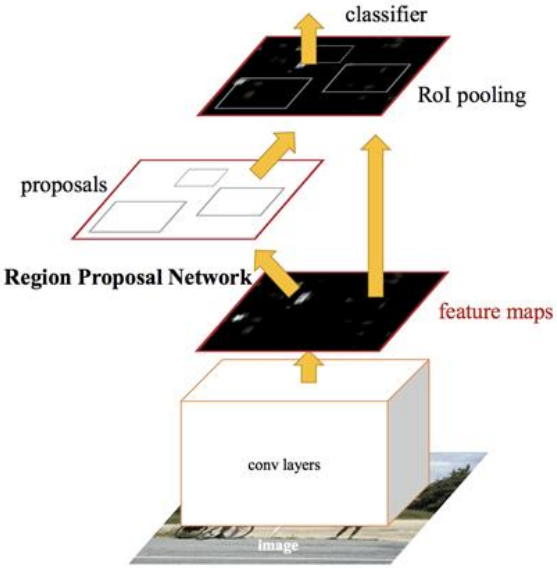
The model architecture was improved even more by speeding up training and detection by Shaoqin Ren in 2016. The new architecture was designed to both propose and refine region of interest proposals as part of the training process. This is called a region proposal network and these regions would then be used with a Fast RCNN model design to get the desired output. These improvements reduced the number of regions of interest and sped up test time to almost real-time object detection.

Although Faster RCNN is a single unified model the architecture is a combination of two different modules:

- The Region Proposal Network (RPN): a CNN for proposing regions of interest and the type of object in the region
- Fast RCNN: a CNN for extracting features and producing bounding boxes and labels

The RPN works by taking the output of a pretrained CNN and passing a small network over the feature map to generate anchor boxes designed to accelerate and improve the proposals of regions. Each anchor box will predict if there is an object within it and

adjust the box to fit the object. Faster RCNN is very similar to Fast RCNN but the use of the RPN makes selecting regions of interest much faster and the overall process much quicker. This is why these methods of RCNN are used in object detection models that detect objects at actual camera speeds and are used on autonomous cars and drones.



Summary of Faster R-CNN Model Architecture [29]

RetinaNet

RetinaNet is one of the best one-stage object detection models when it comes to working with dense and small-scale objects. It utilizes a focal loss function to address class imbalance during training. Focal Loss applies a modulating term to the cross-entropy loss in order to focus learning on hard negative examples. It has become a popular object detection model to be used in aerial and satellite imagery. Since our drone will be used in aerial mode, RetinaNet is highly considered and also RetinaNet is compatible with Robotic Operating System (ROS).

RetinaNet Architecture

The architecture that forms RetinaNet is based on four major components:

1) Bottom-up Pathway - The backbone network using ResNet which calculates the features of a map at different scales, not taking into account the input size of the image or the backbone.

2) Top-down Pathway and Lateral Connections - The top-down up-samples feature maps from pyramid levels, and the lateral connections merge the top-down and bottom-up with the same size.

3) Classification subnetwork - Predicts the probability of a certain object being present for each anchor box and object class.

4) Regression subnetwork - Regresses the offset for the bounding boxes for each ground truth object.

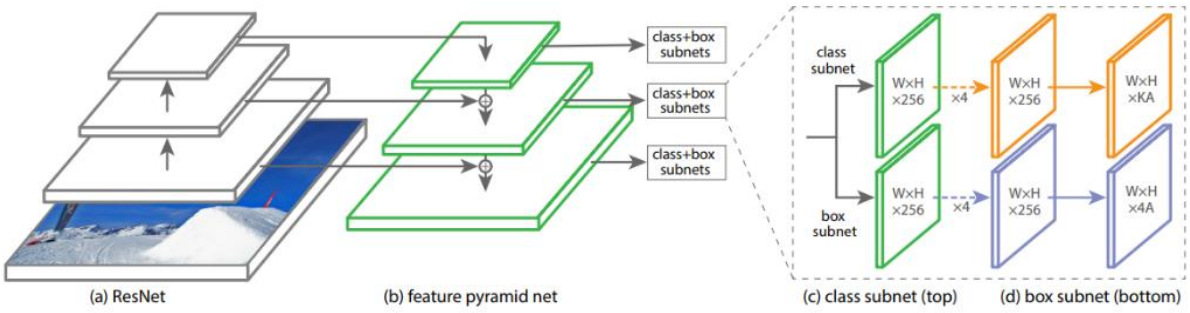


Figure: RetinaNet Architecture [74]

RetinaNet Focal Loss

RetinaNet uses Focal Loss (FL) as an enhancement over Cross-Entropy Loss (CE). CE is used to handle the class imbalance problem. In Two-Stage Detectors such as Faster R-CNN, the first stage, region proposal network makes the number of candidates object

reduce to a small number of 1-2k, obtaining or filtering the background samples. Classification is performed for each candidate location with sampling heuristics using an already fixed foreground-to-background ratio of (1:3) or instead it can use an online hard example mining (OHEM) to select a small set of anchors (,256) for each minibatch.

When it comes to One-Stage Detectors, there exists a larger set of object locations across an image which can range from ~100k locations, this covers spatial positions, scales and aspect ratios. The training procedure for One-Stage Detectors is still dominated by easily classified background examples which can be done with hard example mining.

In comparison of number of boxes, YOLOv1 can handle 98 boxes, YOLOv2 around 1k, OverFeat around 1-2k boxes, OverFeat can range from 8-26k, and finally RetinaNet can handle 100k boxes with the resolve of class imbalance using Focal Loss.

To understand the Focal Loss equation, we first need to take a look at the Cross Entropy loss for binary classification equation and rewrite it in terms of model probability *p*

$$\mathrm{CE}(p, y) = \begin{cases} -\log(p) & \text{if } y = 1 \\ -\log(1-p) & \text{otherwise.} \end{cases}$$

Figure: CE loss binary classification[57]

$$p_t = \begin{cases} p & \text{if } y = 1 \\ 1-p & \text{otherwise,} \end{cases}$$

$$\mathrm{CE}(p, y) = \mathrm{CE}(p_t) = -\log(p_t).$$

Figure: CE loss binary classification rewritten[57]

But if we make a comparison of using RetinaNet for one hundred thousand (100k) of easy examples and just 100 hard examples which contain 2.3 each, we can calculate the CE loss.

- Loss from hard examples = 100 x 2.3 = 230
- Loss from easy examples = 100000 x 0.1 = 10000

If we divide the easy examples over hard examples (10000 / 230) we obtain 43. Which is about 40 times bigger loss from easy examples. Therefore, CE loss is not good when it comes to an extreme class imbalance situation. This means that we need to address this problem with a balance CE loss function.

Here is the a −Balance CE Loss equation in order to address the class imbalance

$$\mathrm{CE}(p_t) = -\alpha_t \log(p_t).$$

Figure: alpha - Balance CE Loss

a (alpha) is implemented by selecting the foreground-to-background ratio of (1:3) and add a weighting factor for class 1 and 1 - a for class -1. a can be set by the inverse class frequency or treated as a hyperparameter to set by cross validation. We finally obtain the Focal Loss function

$$\mathrm{FL}(p_t) = -(1 - p_t)^{\gamma} \log(p_t).$$

Figure: Focal Loss function

There exist two properties for the FL:

1. If an example is not well classified and the probability of (p_t) is small, the modulating factor is close to 1 and the loss is unaffected, but as the probability goes to or get close to 1 ($p_t \rightarrow 1$), the factor goes to - and the loss for the examples that are well-classified is eventually down-weighted.

2. The focusing parameter γ adjusts the rate at which easy examples are downweighted. When gamma equals zero ($\gamma = 0$), the Focal Loss is equivalent to the Cross Entropy. But when this is increased, the effect of the factor is also increased.

There also exist an alpha-Balance variant for Focal Loss

$$\mathrm{FL}(p_t) = -\alpha_t(1 - p_t)^{\gamma} \log(p_t).$$

Figure: a-Balance Variant of FL

This variant for Focal Loss is used to improve accuracy including a over the one equation that does not use it. It also uses a sigmoid activation function in order to compute probability which results in a greater numerical stability. Therefore, γ focuses more on hard examples and a on offset class imbalance of number of examples.

Here is a small study of dataset named COCO which compares the variant of Cross entropy and the variant of Focal Loss using a dataset with thirty-five thousand (tranval35k) for training and minimal (5k) for validation

α	AP	AP_{50}	AP_{75}
.10	0.0	0.0	0.0
.25	10.8	16.0	11.7
.50	30.2	46.7	32.8
.75	31.1	49.4	33.0
.90	30.8	49.7	32.3
.99	28.7	47.4	29.9
.999	25.1	41.7	26.1

(a) **Varying α for CE loss** ($\gamma = 0$)

γ	α	AP	AP_{50}	AP_{75}
0	.75	31.1	49.4	33.0
0.1	.75	31.4	49.9	33.1
0.2	.75	31.9	50.7	33.4
0.5	.50	32.9	51.7	35.2
1.0	.25	33.7	52.0	36.2
2.0	.25	**34.0**	**52.5**	**36.5**
5.0	.25	32.2	49.6	34.8

(b) **Varying γ for FL** (w. optimal α)

Figure: CE loss (left), FL (right)[57]

Intersection Over Union

If we have a picture that contains objects in it, and a model that is predicting the bounding box of where those objects are in a picture, there is a need for a scoring aspect that we can use to tell how well our model is actually performing. The evaluation method for this project and for the machine learning algorithm that we will be using is intersection over union. In order to calculate this metric, we first need to have a drawn or established “ground truth” object box that marks the correct position of where the

object is in the picture or image. We then can take the area of this ground truth box and use it in our intersection over union calculation. The actual formula itself is derived right from the name, as it is the intersection of the ground truth box and the predicted bounding box divided by the union of both the ground truth and predicted bounding boxes. The final number produced will be a number ranging from zero to one that is a percentage-based indication of how much the predicted box encompasses where the actual ground truth box is.

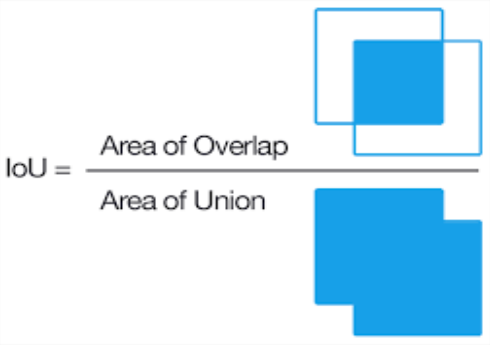


Image depicting how the formula for how IoU is derived based on the boundary boxes[57]

In our project, the intersection over union that is calculated based on how where the target object is in our simulated world must be greater than 0.7 and be able to maintain this efficiency throughout the trial.

TensorFlow

TensorFlow is a vast open-source machine learning API that has an immense amount of regression, classification, and neural network machine learning models. In this project, we will mainly be focusing on the different kinds of neural networks that can perform object classification on images and photos to be used in a real time object detection algorithm that our drone will use. The most popular classifier is the buildable Keras sequential model that is embedded in TensorFlow. Inside of this sequential model the layers that are added are customizable and all depend on the programmer. The layers all have their different uses for the different machine learning models.

Keras

Keras is the API that is a part of TensorFlow that contains sequential models that can be customized and built to meet the specifications of whatever model suits a given problem's needs. The model is composed of layers and each of these layers have different properties and uses. Each layer has hyperparameters that can be changed so that the best use of a layer is being had. These are the most popular layers and hyper parameters for different layers from Keras:

- Conv2D Layer - Most useful layer in a Convolutional Neural Network (CNN). This layer creates and uses a convolutional kernel that works on an input to produce an output.
- MaxPooling2D Layer - Also useful for CNNs, Max Pooling layers use a kernel to downsize a given sample by taking the largest value inside the kernel as it moves along from region to region.
- Dense Layer - Useful for many different machine learning models, this layer acts as the output layer and will use an activation function to determine either the class or regression number that a model produces based on a sample input.
- Relu Activation - Activation used in regression problems where if the output number is positive, it will output that number, otherwise if the number is zero or negative the activation will cause zero to be the output.
- Sigmoid Activation - Activation used for classification. This function will output based on the weights of the model and classify the object according to what the input values are.

These are just some of the many activations and layers that koras provide for their models, and they are the most relevant to the object detection and R-CNN that the algorithm we use will be utilizing.

YOLO

You Only Look Once (YOLO) is an open source, state of the art object detection algorithm, developed by a team from the University of Washington. Used because of its very high speeds while maintaining accuracy and ability to track objects in real time makes it very useful for autonomous robot computer vision.

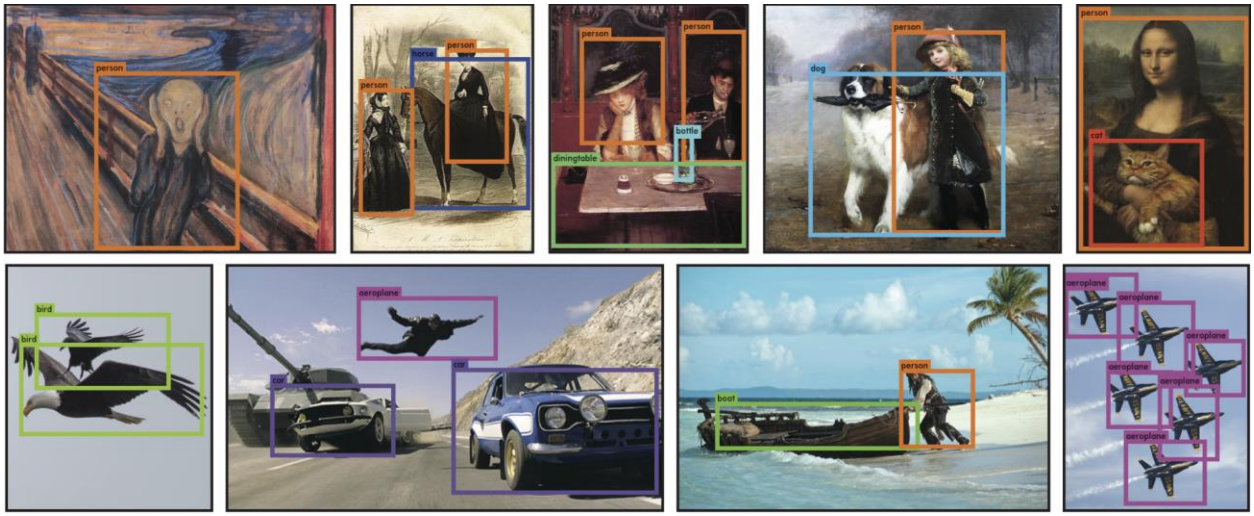


Figure: YOLO running on sample artwork and images [6]

Within one evaluation of full images, a single neural network can predict the bounding boxes and class probabilities. This algorithm utilizes three main techniques for its calculation. Bounding box regression is used to train the model and adjust the weights to allow for new images to be broken down and classified themselves. The model uses a residual block layout where a certain number of layers are skipped in order to obtain a certain accuracy result while also maintaining a fast speed. For an accuracy metric, intersection of union is used to determine how well the model is performing and indicate whether adjustments need to be made.

Here is a figure and quote of how YOLO works from the creator's paper and that highlights how simple YOLO is:

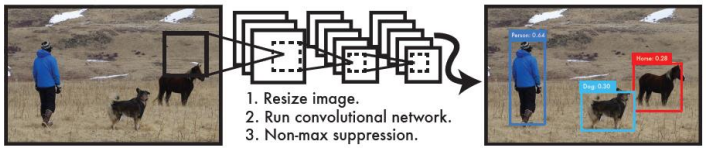


Figure: Yolo Detection System [6]

"Our system (1) resizes the input image to 448 × 448, (2) runs a single convolutional network on the image, and (3) thresholds the resulting detections by the model's confidence".

There are different variants of YOLO object detection which will be compare for the purpose of this project, but our main focus will be YOLOv3 and YOLOv4

YOLO Architecture

The architecture used in You Only Look Once (YOLO) consists of using 24 convolutional layers that are connected by 2 fully connected layers. In order to reduce the number of parameters it uses a 1×1 convolutional layers which is also what GoogleNet has been using and it was inspired by it. The variant Fast YOLO contains fewer convolutional layers compare to the previous version; it contains specifically 9 convolutional layers instead of 24

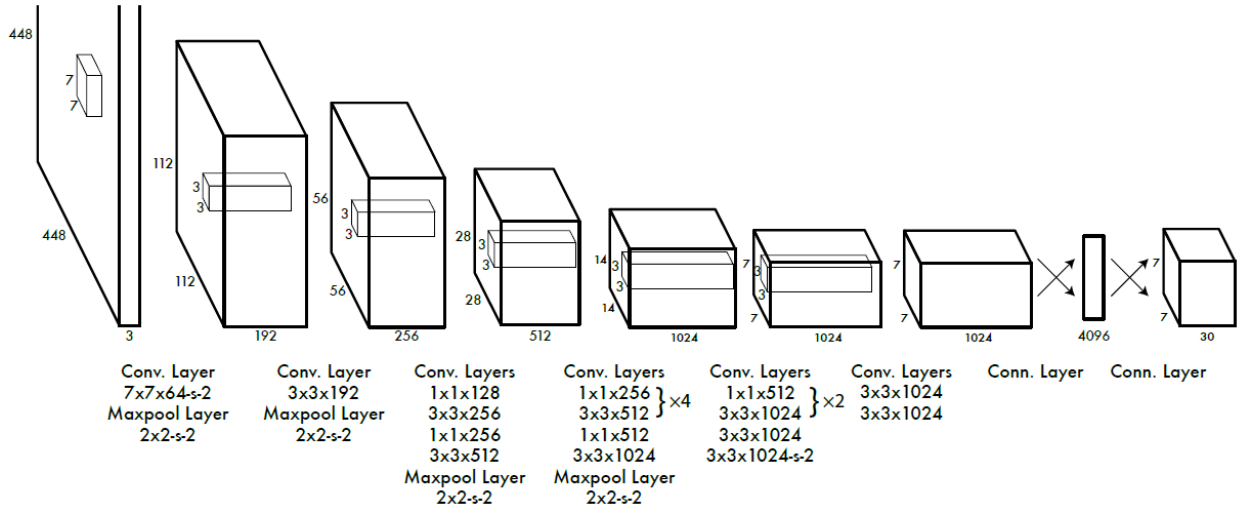


Figure: YOLO Architecture[53]

Image gets processed through a network pipeline. After convolution image is divided into an $S \times S$ grid and for all individual cell in the grid, it predicts B bounding boxes, confidence for those boxes, and C class probabilities. After all these, the results or predictions are encoded as the following form $S \times S \times (B * 5 + C)$ tensor.

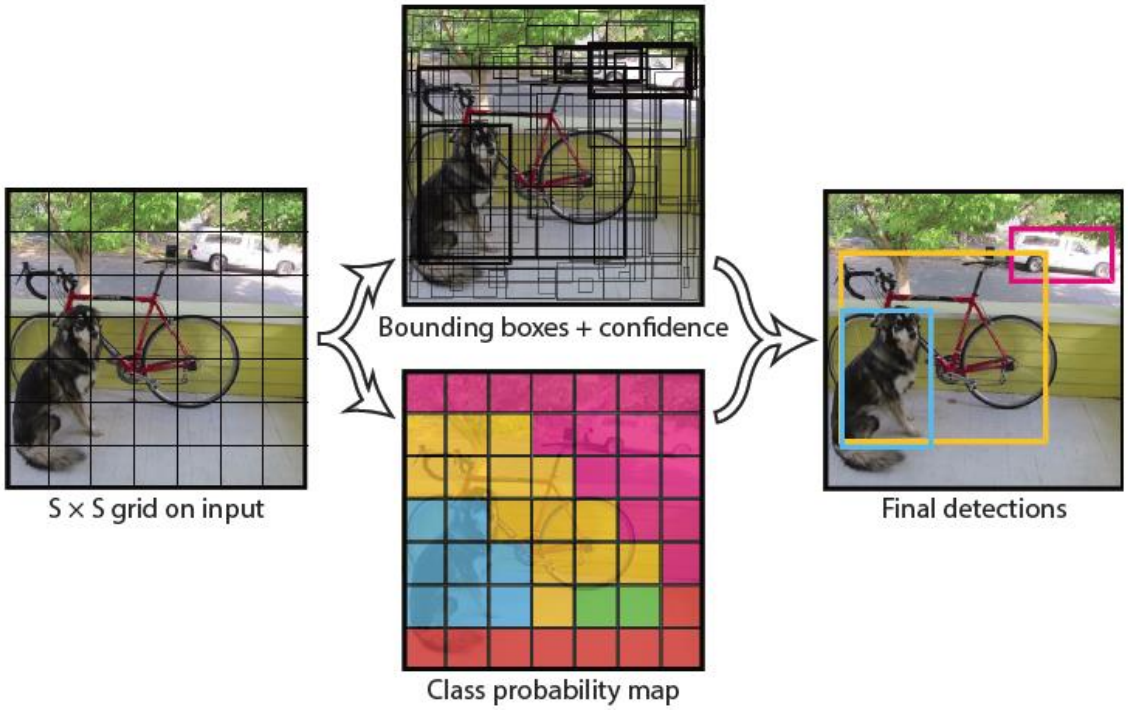


Figure: Image Model [29]

Here is the fully connected version network pipeline which includes the convolutions, input image grid, followed by bounding boxes and class probability with the result of the final detection.

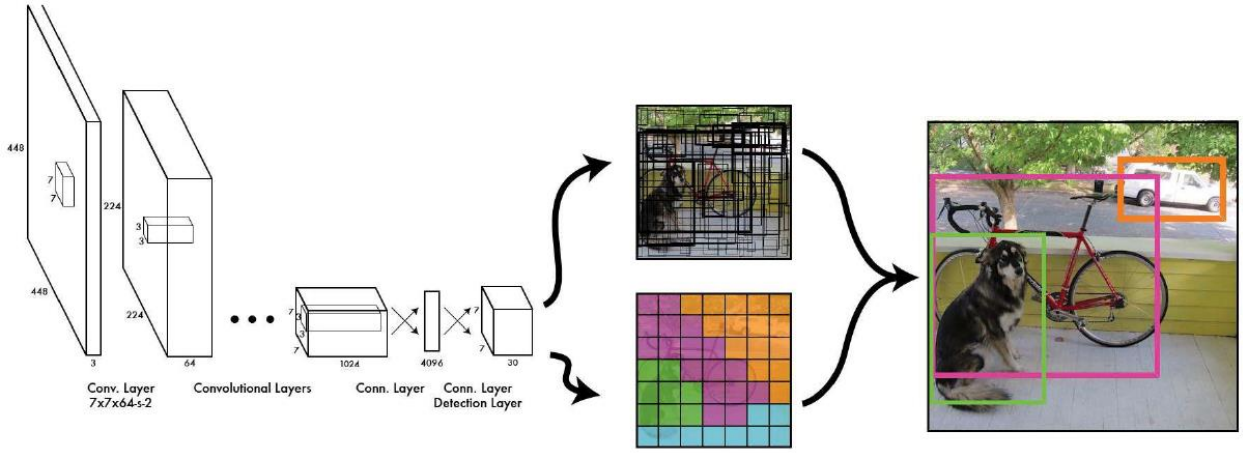


Figure: Network Pipeline [29]

YOLO Loss Function

YOLO is trained on a loss function that corresponds to the detection performance. There exist 5 terms that make the loss function.

1. The first term contains the bounding box and coordinates that together are denoted as (x, y) which is parametrized of a particular cell location in the grid bounded between 0 and 1. The sum of square error (SSE) is used only when there is an object present otherwise it will not be used.

2. The second term contains the bounding box width and height denoted as (w, h) which are normalized by the input image between the values 0 and 1. Here the SSE is also used when there is an object.

3. The third and fourth term represent the confidence.

4. The final fifth term is for the class probabilities using SSE when there is an object.

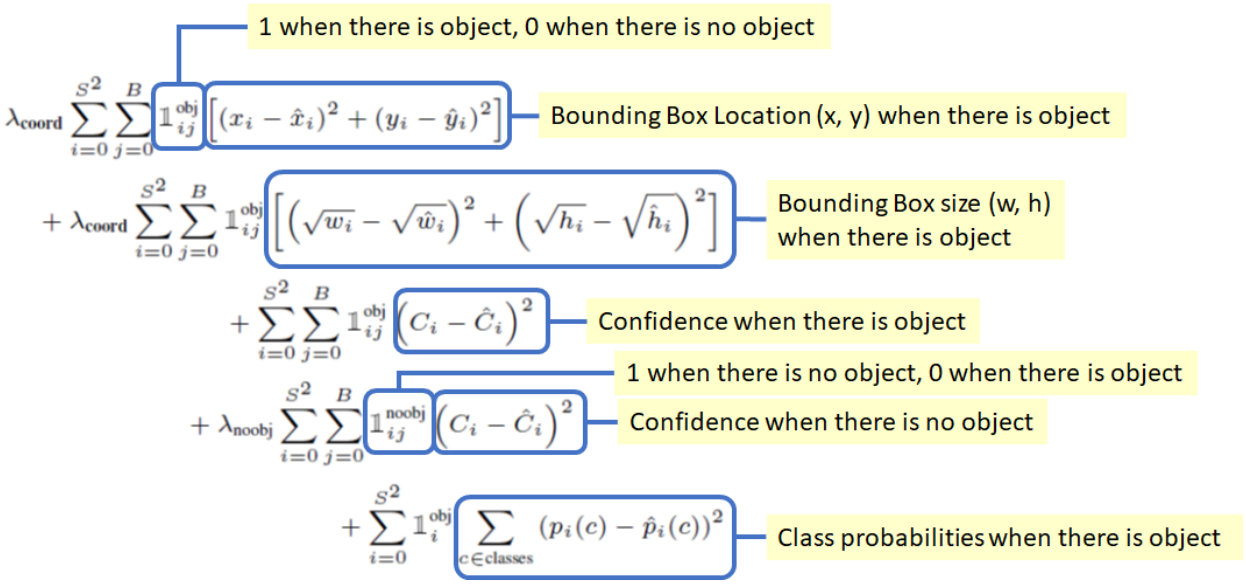


Figure: Loss Function[52]

YOLO Version Comparison

There have been 5 different versions of YOLO, one might be better than the other and we are going to compare and output their specifications and improvements. We are going to start from YOLOv2 and after since the previous version YOLOv1 or just YOLO was already discussed.

- YOLO9000 / YOLOv2:

    - Improvement of layers from the previous version YOLOv1 from 26 layers which is what originally had to an improvement of 30 layers.

    - Anchor Boxes were introduced. This is provided by Darknet which gives the pipeline network the idea about position and dimension of objects detected.

    - Random dimension for training images from 320 to 608.

    - Still bad for detecting small objects.

    - Compatible with Robotic Operating System (ROS).

- YOLOv3:

    - Improvement in neural network layers with 106 layers.

    - Detection is based on 3 scales for detecting objects of small to very large size.

    - Changes in the error function.

    - Better with small objects.

Compatible with Robotic Operating System (ROS).

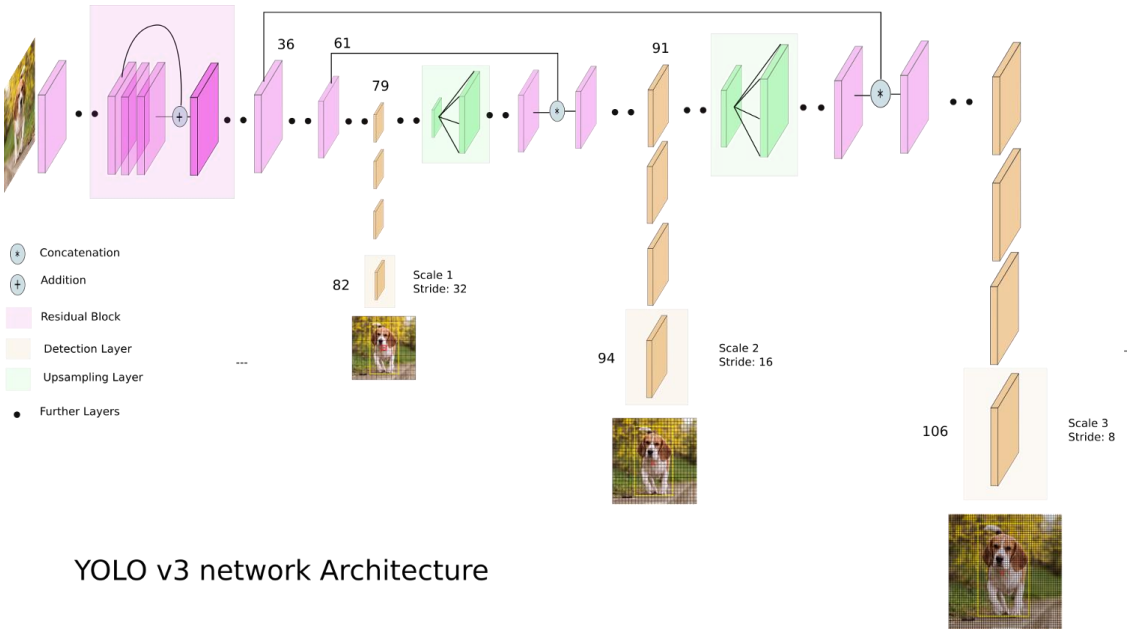


Figure: YOLOv3 Architecture[52]

- YOLOv4:
    - Improvement in the mean average precision by 10%.
    - Improvement in the number of frames per second by 12%
    - Contains 4 distinct blocks, this includes the neck, the dense prediction, and the sparse prediction.
    - Backbone features extraction architecture which is CSPDarknet53.

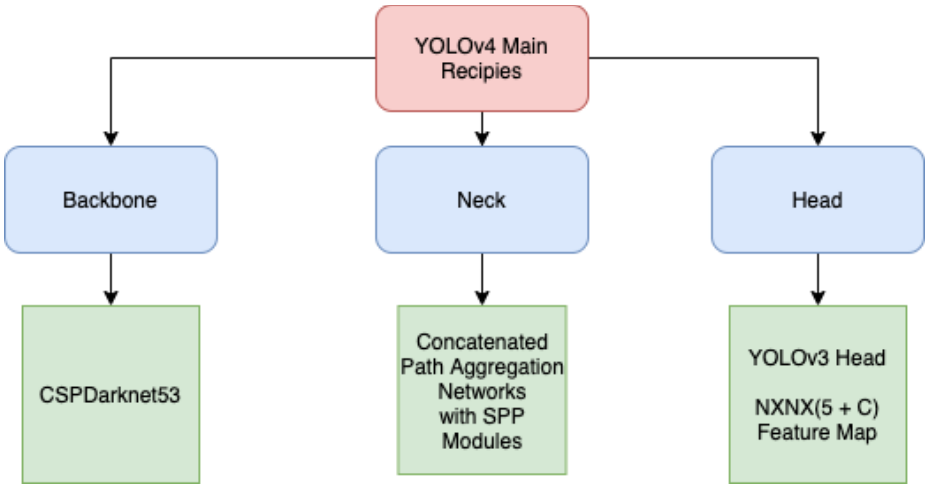


Figure: YOLOv4 Architecture[55]

- YOLOv5:
    - Porch implementation
    - CSP as backbone and PA-NET as the neck
    - Auto learning bounding box anchors
    - Most recent version created in 2020
    - Smaller than the previous version containing 27 megabytes instead of 244 megabytes

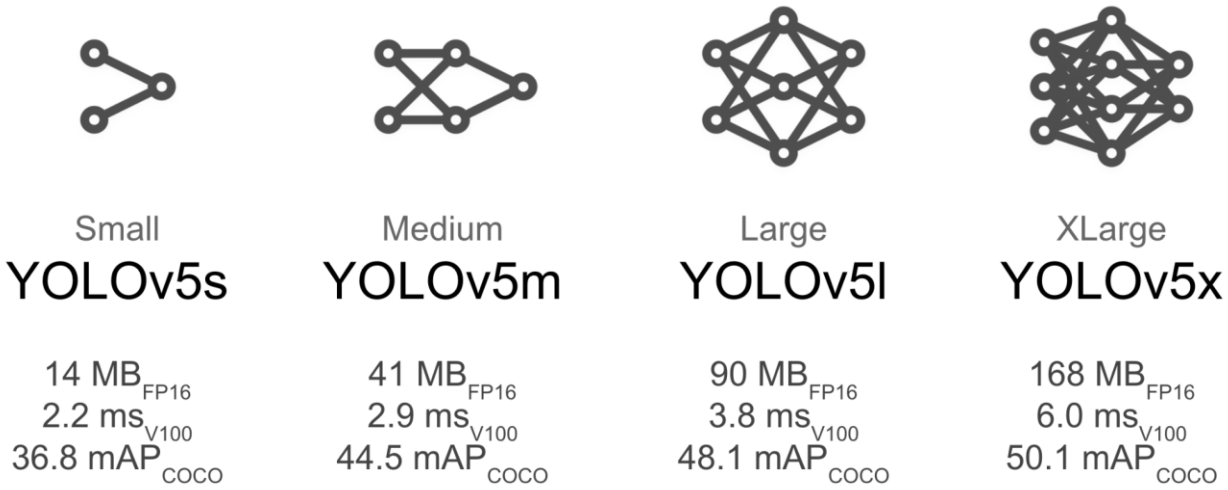


Figure: YOLOv5 Model Selection[55]

YOLOv3 and YOLOv4 in depth

Since there is a big controversy for the latest version of You Only Look Once (YOLOv5) because it has not been accepted by the community, there is not much testing data and that is why we are mainly focusing the last two previous versions which are YOLOv3 and YOLOv4.

YOLOv3 is based on DarkNet-53 because it uses a feature extraction network and is structured to adopt the strategy of a pyramid network. The inputs that it uses are pictures and annotations for coaching. The picture is sent into the extraction network, and the extracted feature vectors are then obtained which then is sent to a structure just like a pyramid network and this is how the grid cell is obtained at three scales. Furthermore, every cell in the gird predicts three bounding boxes, and each of the bounding boxes predicts a vector denoted as P. Therefore, it will contain the bounding box and coordinates as well as the width and height and the $Pr(Object)$ represents the likelihood that the object is in the range of the prediction box.

$$P=(tx+ty+tw+th) +P0+(P1+P2+\cdots + Pn)$$

$$P0= \Pr(\text{Object})\times \text{IOU}_{\text{predtruth}}$$

Figure: Bounding Box Prediction[55]

In contrast to YOLOv4, the main benefit is that v4 beats the existing method by a large margin in areas of detection performance and speed. The team that created YOLOv4 focused more on optimizing neural networks detectors for parallel computations. The architecture of this version is based on a backbone CSPDarknet53, a neck based on the spatial pyramid pooling module, and PANet path-aggregation, and finally the head which is based on YOLOv3.

The backbone is usually pre trained on ImageNet by using the extractor feature which gives the feature map representation of the input image. ImageNet consists of a database in which each node of the hierarchy is depicted by thousands of example images.

The CSPDarknet53 which is used in the backbone augments the learning capacity of the convolutional neural network (CNN), the pooling is attached to overhead for improving the receptive field and distinguishing the important context features. Here is an example of a research of blood cell dataset which outlines the precision, the recall and accuracy of the different versions of YOLO v3, v4, and v5.

Figure: Comparison of Parameters of YOLO v3, v4, and v5[56]

Darknet

Darknet is an open-source software that uses YOLO as a framework and is written in C and CUDA, two of the faster programming languages, in order to obtain an even faster speed. Using darknet algorithms that are readily available makes training a dataset using yolo very easy. The first step to training a model using darknight is to have all of the images and their respective labelled files in a folder and ready for use in the algorithm. The way the model will be trained will be using google colab, so these image files must be located in a directory that can be accessed in the notebook. The next step is to clone the respective darknet repositories in order to run the algorithm on the image files. Following the cloning of the repository there may be some configurations that need to be changed in order to correctly train the model. The .cfg files are the configuration files we need, here are some of the different configs that can be edited and what they change in our model.

- Decay - This accounts for how much the weights are adjusted to manage the variance that the gradient can be calculated as, which can stabilize a model that has a very quick changing gradient or one that does not have a large enough gradient.
- Batch Size - The number of images that will be processed at a time before updating the gradient and then changing the weights.
- Subdivisions - The number of blocks that a single batch is broken up into. This relates to how the images will be processed on the gpu and creates an efficient process for the algorithm.

There are many more configurations that can be changed before the model is run, these are just some of the most relevant configs to our problem at hand. We will need to incorporate and test a lot of different methods in order to obtain the fastest one, therefore these will be useful.

5.4 Pathfinding/Mapping

Our original task was to not only locate the target but also disable the target via our option of choice. After speaking with our sponsor, he has decided that he would rather us focus on getting our drones to just identify the target successfully and not worrying about disabling it. This removes much of the headache of not only trying to come up with a way to stop the target, but also performing these actions during the simulation in gazebo. I am not sure if we could simulate having a weapon attached to our drone. But this does sound like an interesting problem that maybe the next iteration of this project can tackle.

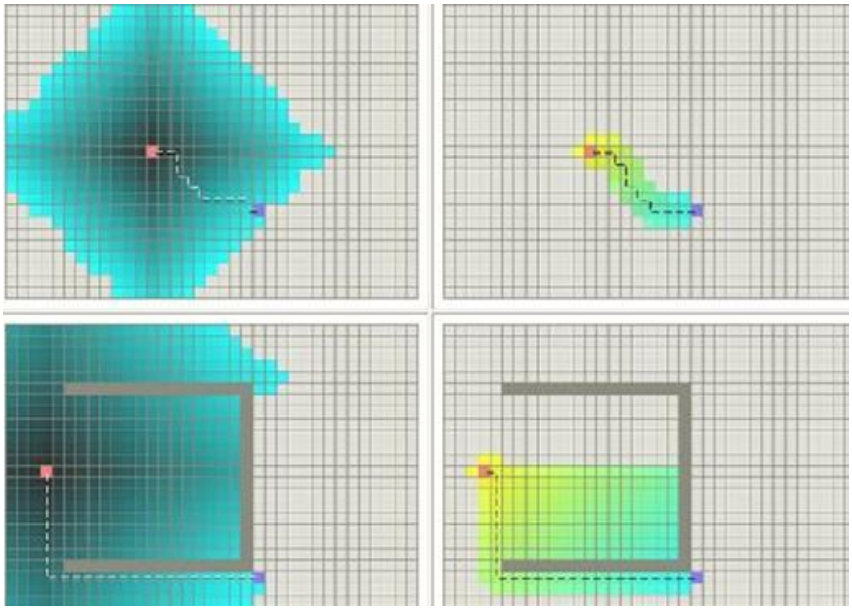
Pathfinding is the mapping out of the shortest route between two points done by a computer application, like how someone looks for the exit to a maze. This problem can be represented as a graph with nodes and weighted paths connecting nodes. The goal is to start at a node and find the cheapest path to the destination node. Calculating the cheapest path can be time-consuming if we were going to evaluate each path exhaustively. Luckily there are algorithms that can be applied to this problem that are able to throw out certain paths known not to be the solution.

Dijksra's Algorithm

The first common example of such a pathfinding algorithm is Dijkstra's pathfinding algorithm. It works by taking the shortest path from the start at each node and keeping track of what nodes have been visited. Since the lowest value nodes are traveled through first the first path that is found will be the shortest path. The problem here is that the algorithm examines the current shortest path from the start and maybe leads in the opposite direction of the end node wasting time.

A* Algorithm

A variation of this algorithm is called the A* algorithm, this pathfinding algorithm was recommended to us by Professor Leinecker. The variation of this algorithm is a built-in heuristic that can identify getting closer to the destination node. What the heuristic represents is the distance between a node and the end node. Using this information, A* can examine paths more efficiently as it is almost guided toward the end node instead of searching the cheapest options first.



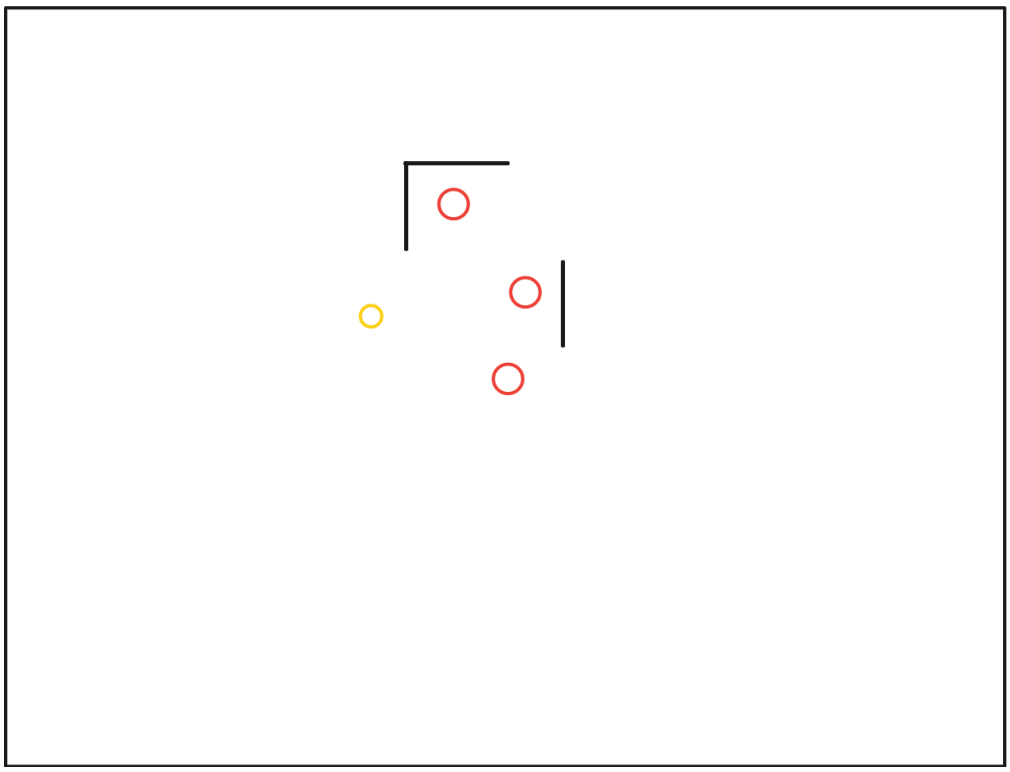
Dijkstra V.S. A [48]*

This image represents the use of Dijkstra's algorithm vs A* the color means a path was examined. Less of an area was searched when using A* making it the more efficient algorithm. The drone will be able to use lidar sensors and cameras to check if its path is blocked and using the A* algorithm it can make a change to its path until the destination is reached.

Tactical Display

The tactical display didn't have any listed requirements on what it should need in order to be successful. There are many ways to implement our tactical display and the more features we want to include in our display, the harder it is to implement.

The plan for the display is a topographical view of the area that displays the locations of each of the drones as well as any obstacles that they would have to go around instead of under or over. It will test our SLAM algorithm to ensure that the drones are properly mapping its surroundings and the location of the drones are accurate.



Base Tactical Display Concept[48]

The image above is an image of the basic concept of our tactical display. There is a preset scale to the map with the starting point of the map (yellow dot) being around the center of the display. The drones (red dot) will then map out the obstacles that they face throughout their search and mark these on the display (black lines). Note that the display doesn't need to label the target because it is assumed that once the target is found, the simulation ends.

Multi Agent Pathfinding

While A* is widely used in drones because we are attempting to use multiple agents, we may require a different type of pathfinding algorithm. We could treat each individual agent as a single joint agent and then apply a variant of the A* algorithm, as this has been done successfully in past experiments. There is a new multi-agent pathfinding algorithm that has been created, the Conflict Based Search. An excerpt from the paper written on it “CBS is a two-level algorithm that does not convert the problem into the single ‘joint agent’ model. At the high level, a search is performed on a Conflict Tree (CT) which is a tree based on conflicts between individual agents. Each node in the CT represents a set of constraints on the motion of the agents. At the low level, fast single-agent searches are performed to satisfy the constraints imposed by the high-level CT node. In many cases this two-level formulation enables CBS to examine fewer states than A* while still maintaining optimality.” This algorithm will consider the position of the other agents and be able to minimize conflict between the multiple flight paths.

```
   Input: MAPF instance
 1 Root.constraints = ∅
 2 Root.solution = find individual paths by the low level()
 3 Root.cost = SIC(Root.solution)
 4 insert Root to OPEN
 5 while OPEN not empty do
 6     P ← best node from OPEN // lowest solution cost
 7     Validate the paths in P until a conflict occurs.
 8     if P has no conflict then
 9         return P.solution // P is goal
10     C ← first conflict (a_i, a_j, v, t) in P
11     if shouldMerge(a_i, a_j) // Optional, MA-CBS only then
12         a_{i,j} = merge(a_i, a_j, v, t)
13         Update P.constraints(external constraints).
14         Update P.solution by invoking low level(a_{i,j})
15         Update P.cost
16         if P.cost < ∞ // A solution was found then
17             Insert P to OPEN
18         continue // go back to the while statement
19     foreach agent a_i in C do
20         A ← new node
21         A.constraints ← P.constraints + (a_i, v, t)
22         A.solution ← P.solution
23         Update A.solution by invoking low level(a_i)
24         A.cost = SIC(A.solution)
25         if A.cost < ∞ // A solution was found then
26             Insert A to OPEN
```

Pseudo-code of the CBS algorithm [29]

It is unclear at this time which pathfinding algorithm would best suit the needs of this project as our drones may have a flight path that would never come into conflict with one another, and we would not have to worry about implementing such an algorithm.

Localization and Drift

For our agents to be able to make their flight path from start to finish we are going to have to keep track of a few things during their run. The agents will need to know their position in our simulated environment. This can be done by keeping track of the agent's orientation, acceleration and speed and estimating the next pose based on this information as well as added information from a camera or sensor. This is what is known as visual inertial odometry (VIO). While this method works it is not perfect and will experience what is known as "drift" in the robotics community. Drift is the accumulation of small estimation errors as the robot continues its path and will give an inaccurate estimation of the robot's location. This can be helped with different sensors such as a lidar sensor, which measures the distance between the surrounding environment by emitting pulsed light waves. By measuring distances between known landmarks and the agent you can account for drift and should be able to draw a more precise location of the agent. We should be able to simulate such a sensor on our drones in Gazebo for testing.

SLAM

Simultaneous Localization and Mapping (SLAM) is a series of algorithms that can be used to map out our simulated environment and track the location of the drone. VisualSLAM is a version of SLAM where both of these are done simultaneously. SLAM does this by utilizing data from the robot's sensors and any other relevant data-gathering tools that the robot may have to map out the environment. It will also include calculating how the robot will need to move to transverse the remainder of the environment.

SLAM fits the criteria for the problems in our project. The drones will be placed into an unknown location and be able to map themselves and the area around them, which is the main purpose of SLAM. Additionally, for our drones, SLAM will specifically use the sensors in our drone to map out the environment and avoid repeating over finished areas. It also decides how our drone will fly around to map out an area.

SLAM has many different versions, however, some algorithms that we are interested in are Collaborative SLAM and the previously talked about VisualSLAM. Collaborative SLAM is meant to combine images from multiple robots to create a complete 3D image from the images. This may be important for the swarm component of our project. Having

multiple drones work together to create the map would greatly speed up the data collection for map processing.

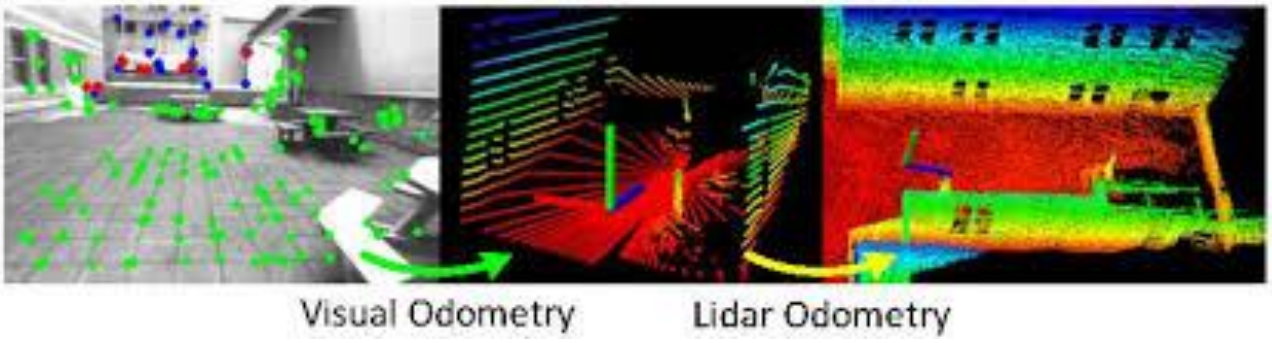
Implementing SLAM effectively is critical to keeping our search algorithm within the time frame and may also be useful in our implementation of the tactical display because of its mapping capabilities. The main issue with SLAM is that as the calculations become more complex, the more difficult it will be to perform in real-time.

Visual SLAM

Visual SLAM is a version of SLAM that performs the algorithm using some form of visual measurement. In our case, the visual element will be the cameras that each drone is equipped with. The drones will take photos of the environment and map where they have been and store that information to make the target finding process fast and efficient. By keeping track of landmarks and different aspects of the area being mapped out, the algorithm can track both new areas that it has not seen before and be able to tell when it is in an area that has already been explored. This algorithm will be run in parallel with the machine learning algorithms for object avoidance and detection so the computations will have to be quick and precise in order to meet our time limit requirement.

LIDAR SLAM

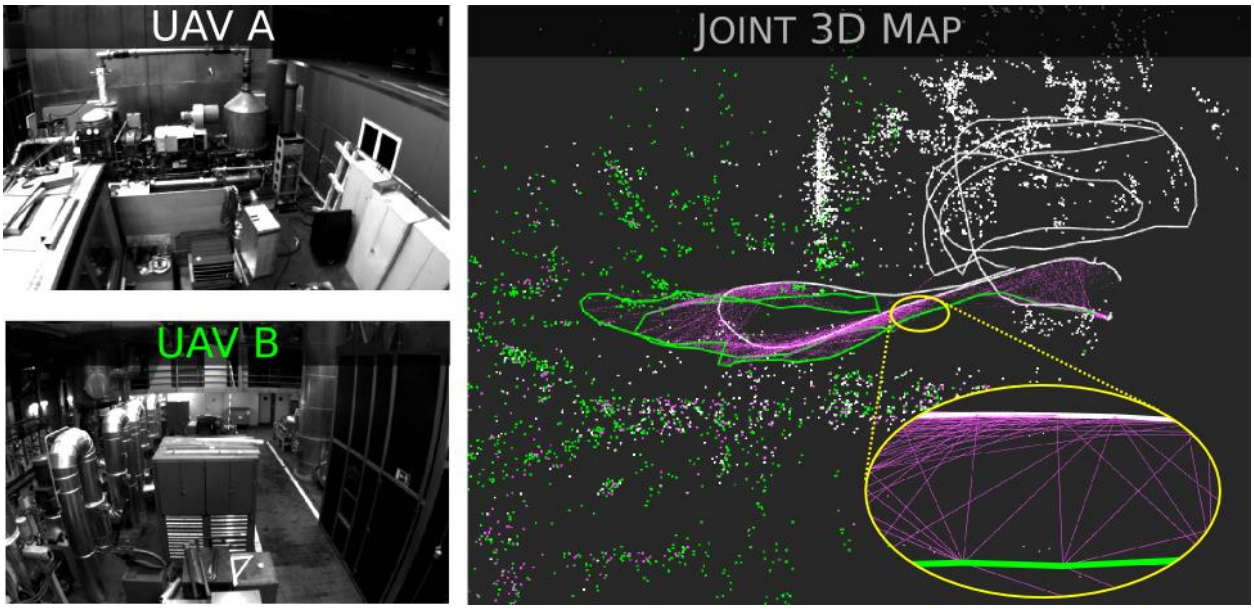
LIDAR SLAM is yet another version of SLAM that is very prevalent to our project. LIDAR SLAM works with the LIDAR range sensors to map the environment beneath a vehicle. Because the only input the algorithm will be getting is how close the nearest object is to the drone, there is a lot less computation involved, making it faster. However, the fast computation comes at a cost with the efficiency. The stored information will be less descriptive as certain landmarks and other factors that the visual aspect provides will be non-existent, therefore only relying on the map that is created by the sensors.



[74] The difference between what a visual vs LIDAR SLAM algorithm uses

Collaborative SLAM

In SLAM algorithms that use one drone to map an area the basic principles are simple. A drone traverses through a given area mapping the areas it's already been to in order to fully understand a given location. With a target involved, the algorithm would map a location to mark where the target is not and continue its search to eventually find the target by searching every possible area. With one drone this process may take a long time and in the worst case an entire area may have to be searched alone in order to find a given target. That is why the use of more than one drone can be so effective. This is the general idea behind what collaborative slam is. Multiple drones will work together to map out an environment and know the unsearched locations to travel to. If done correctly and efficiently the time to search for a target can be drastically decreased as the collaboration between the drones can map out an area very quickly.



A real-world example of a collaborative SLAM [77]

CONVINS SLAM System

There is one system of collaborative SLAM that I researched that was published very recently (August 2021), COVINS. I will give a brief system overview and then go over specific tasks performed by COVINS. The system architecture of COVINS is shown running a VIO (visual inertial odometry) front-end maintaining a local map of each agent to have autonomy for each agent. At the same time, global map maintenance and expensive processes are handled by a backend server. The communication module establishes peer-to-peer (p2p) connection, allowing the server to run on a locally deployed computer as well as on a remote cloud service. COVINS provides the freedom to interface with any custom-built keyframe-based VIO system, so that multiple agents can collaborate on a SLAM estimation. The server forms a map manager, which controls access to the global map data in the system. It maintains this map data in one or multiple maps, as well as a keyframe database for efficient place recognition. It also provides algorithms to merge maps once overlap is detected and the ability to remove keyframes that are redundant holding no valuable information. Place recognition modules process all incoming keyframes from the agents to see if there is any visual overlap between re-visited parts of the environment. The server can perform different optimizations like Pose-Graph Optimization (PGO) to better find the pose of the agent. COVINS implements an optimization strategy regularly performing PGO, while executing Global Bundle Adjustment (GBA) to further refine maps less frequently. The server also provides an external interface, allowing a user to interact with the system.

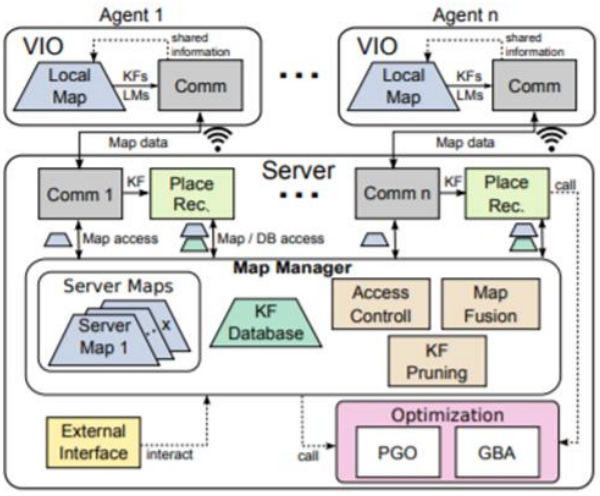


Diagram of the COVINS system architecture [42]

Map Structure

The map structure used by the back-end server of COVINS maintains the data of the collaborative estimations from agents. The server map is a SLAM graph, holding a set Vx of KFs (keyframes) and a set Lx of LMs (landmarks) as vertices, and edges induced either between two KFs through IMU (Inertial Measurement Unit) constraints or between a KF and a LM as a landmark observation. While multiple agents can contribute to the one server map, multiple server maps are created simultaneously until all agents are co-localized. The shared LM observations create dependencies between KFs from multiple agents, while IMU factors are only inserted between consecutive KFs created by the same agent.

Visual-Inertial Odometry

COVINS works by obtaining map data from keyframe-based VIO front-end systems. For the agent to share its map data with the backend server COVINS framework provides a communication interface that allows for the use of any VIO front-end system. For our project I would like to use the ORB-SLAM3 system as I have seen work done with it in the past and in this paper, experiments were performed also using ORB-SLAM3.

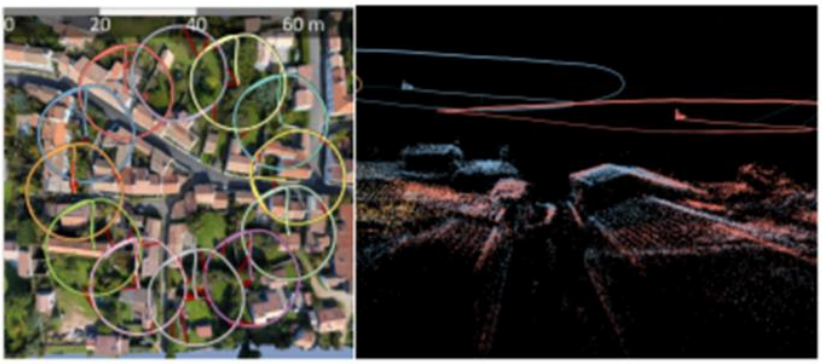
Communication

The communication module used is based on socket programming using the TCP protocol. The server listens to a designated port for incoming connection requests from agents. For sharing map data from the drone to the server COVINS will account for static parts of keyframes and landmarks by not repeatedly sending the data to reduce required network bandwidth. This can be things such as extracted 2-D feature key points and related descriptors. All the map data to be shared with the server is collected very quickly and given to the server batch-wise at a fixed rate. Then the server will integrate transmitted map information into a collaborative SLAM estimate to be sent back to agents.

Multi-Map Management

The map manager holds all the data given by agents in one or more server maps. Each agent that enters the system a new map will be initialized. COVINS will use place recognition to detect overlap between two different maps. When this happens the map manager will fuse the two maps into one server map. The map manager also holds the keyframe database needed for place recognition. It oversees giving access to server maps and the keyframe database, which is important in tasks like restricting map access to perform map fusion or optimization.

This system is open source, and we would be able to use this in our project to map out the urban environment as well as determine where all our agents are in the environment. In this paper they performed two experiments, one with 5 real drones in a flight mission and a simulation of 12 agents scanning an area from overhead. Being able to take in data from all 12 drones and process that into a collaborative SLAM estimate is what impressed me the most about this system. Hopefully that means when we attempt the same thing with only 5 agents it will run just as smooth.



(a) Final 12-agent collaborative SLAM estimate (trajectories only), superimposed on the scene view.

(b) Side view on the trajectories and LM at the intersection between two trajectories, illustrating the accurate alignment of individual map data.

Example of Multi-Map Management [42]

5.5 Data collection

In order to build and train our model we must first collect images in the Gazebo using the simulated drone camera. We want our drones to be able to track the target object in real time and under a certain time limit, so the algorithm in place for object detection must be quick and accurate. To try and encompass as many different situations as possible that the drone may encounter, multiple pictures from various angles, lighting situations, and with different objects in the images will be taken in order to train the model in the best way possible. In the simulated urban world, there will be countless objects that may make it hard for the algorithm to detect only the target, meaning a lot of testing will have to be done with the fully trained model as well. This can be accounted for using distractors or false objects that the model could mistake for as the target. These distractions should be placed wherever the target object could appear.

Machine Learning Techniques

There are two methods of training a machine learning model that are commonly used today. Both are used based on different factors and what a sample set does and does not have. The two types of machine learning are supervised and unsupervised learning. Both types have a reinforcement aspect where a model is “rewarded” or “punished” based on how it performs. This evaluation is done using an evaluation method. In both kinds of machine learning there are features, also known as the x variable or input data, in the training set. The differences between the two types are described below.

- Supervised Learning - In supervised learning, the target vector is known alongside the features. This means that if given a set of input values, the corresponding output value is already known. In our case, the input vector would be the set of pixels that the image is composed of, and the output vector is the bounding box that exists around the object that can be found inside of that image.

- Unsupervised Learning - In unsupervised learning all of the input data is not labelled or classified. Therefore, no definitive outcome can be made as an output with a given set of data. The purpose of unsupervised learning is to identify patterns within data which can lead to discoveries on the correspondence between a number of the features. An example of unsupervised learning would be an algorithm that makes correlations between customers and what they consume based on the other items they consume.

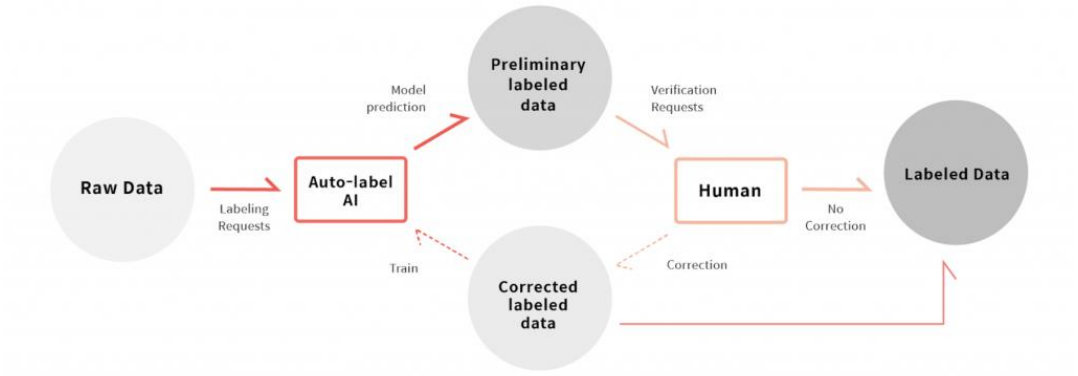
In this project we are using supervised learning. We will have labelled data that will be used to reward or punish our model based on how well it makes its predictions. The way to know how much to punish or reward an algorithm is determined by that model's performance metric, which in our case is intersection over union. The “ground truth” or 100% accurate bounding boxes for the target will have to be given for a certain sample size in order to accomplish our goal of an accurate machine learning model. We then also want to train our dataset in a way where it does not get only familiar with the training data and is unable to predict accurately on new data. In order to solve this problem, we will split the data and in order to obtain a given sample of training data there will be a large set of images taken that will need to be labeled.

Labeling Data

In most of the most popular object detection algorithms a given boundary box must be present in order to evaluate, refine and update a given model. In order for a machine learning model to be the most accurate it can be, a sizable dataset must be used for the most accuracy. To have to manually label all of the images might take a lot of time and be very tedious, so further investigations were made focusing on autonomous ways of labeling objects in images, however in the end, we manually labeled all of the data.

Autonomous Data Labeling

Data Labeling can be extremely tedious and time consuming to do manually, and the best machine learning models are usually trained with large datasets. There are new AI assisted algorithms that can help a person label images with computer vision. By manually labeling a small portion of a given dataset that can perform a smaller scale version of the main model, as the model learns and draws bounding boxes, the user can indicate whether the bounding box predicted is correct or not on the new data that is predicted on. This means that a person would only have to hit yes or no and only label the incorrectly labelled images, and if a model is predicting with an accuracy of only fifty percent, half of the work that a person has to do is almost fully relieved. Below is a diagram that can better illustrate how automated data labeling works between the raw sample set, the AI labeling model and the human verifier of the bounding boxes on the images.

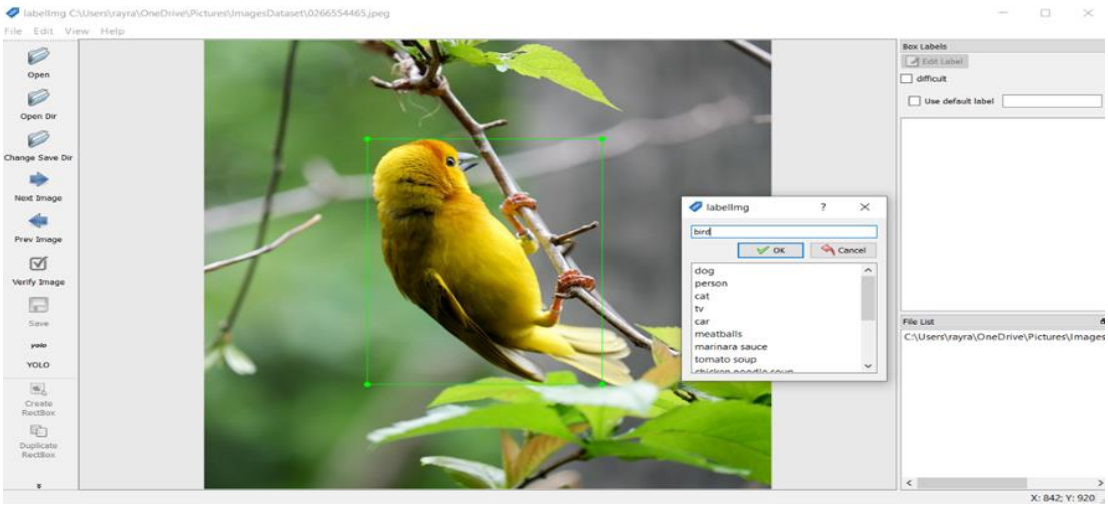


[2] Flow Diagram of how Autonomous AI Labeling Works with Unlabeled Raw Data

In this project we decided against this approach in order to ensure that each image is thoroughly examined and is optimal for the machine learning model that we are trying to train.

Manual Data Labeling

The approach that we decided to do in this project was manually labelling the input data. Using a third-party application, we upload all of the raw images from the dataset for further processing. Selecting the certain algorithm that will be used to train the model, the application will format the labeled files in the correct way so that they can be interpreted. Once the process starts the first picture is presented to the user. The user then draws the ground truth bounding box around each of the detectable objects inside of the images. Accuracy is very important and can have a major impact on the efficiency and accuracy of the model, which is why diligently labeling each image correctly is very important. Once all of the objects have been correctly labelled the user moves on to the next picture and repeats the process until every image in the training dataset has been labeled. In our project, we are only trying to detect one object as the drones are flying and will not need to worry about detecting objects to avoid. However even though our drone will not be detecting other objects there will still be other objects within the environment, meaning that our model will need to be trained for these situations. An example of labeling an image is given below.



Example of labeling an object using LabelImg [52]

*****

In this example photo a bird is being labeled. Using the mouse cursor, you start the drawing of a boundary box by clicking and then dragging the cursor diagonally to create the rectangle that encompasses the object in question. Once this drawn box indicates the object with as much accuracy as possible, it must be given an object name or key. In our case the only object will be the target, but this is the exact process that will be used to label our dataset.

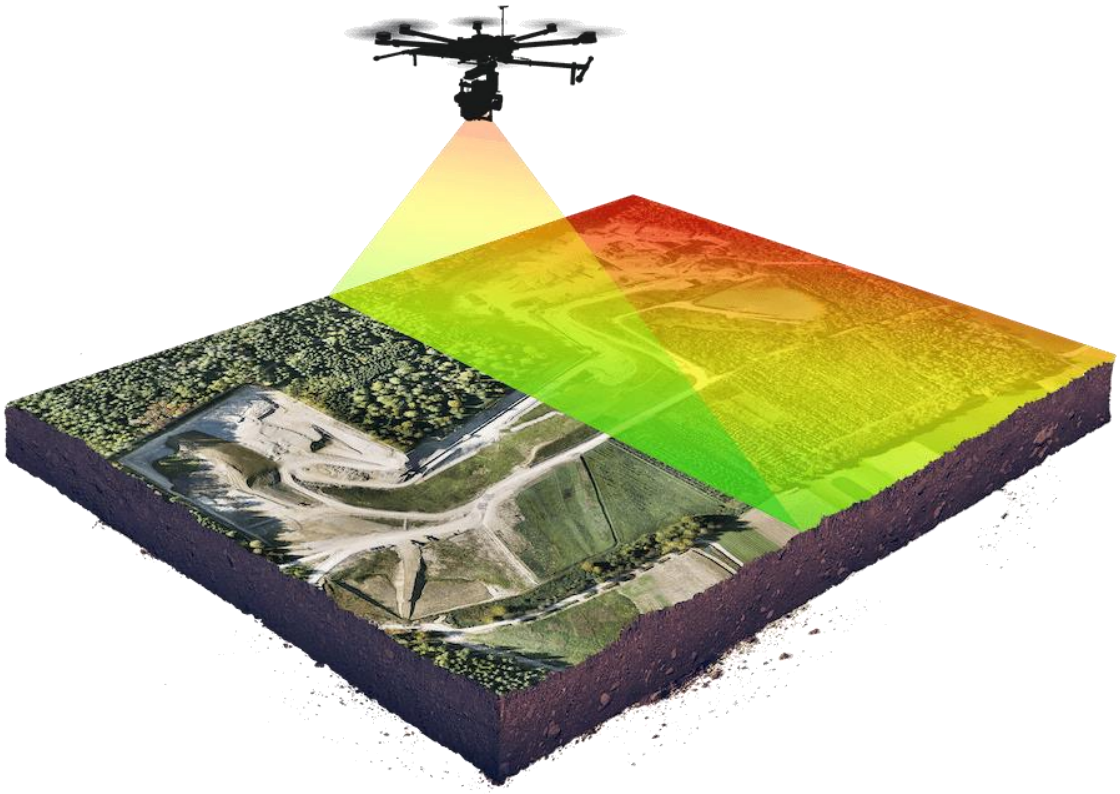
5.6 Swarm Navigation

Overview

Swarm navigation, also known as swarm robotics, is the general term for the method that multiple drones use in order to be able to fly in a coordinated way together. When pairing this navigational system with other algorithms and techniques, drones are able to accomplish things in an efficient way that humans would not be able to accomplish. There are many different existing swarm navigational algorithms available that can make the pathfinding and mapping process a lot faster. In this project we will implement our own swarm navigation, utilizing sensors and other communication techniques so that the target can be quickly identified.

LIDAR Sensors

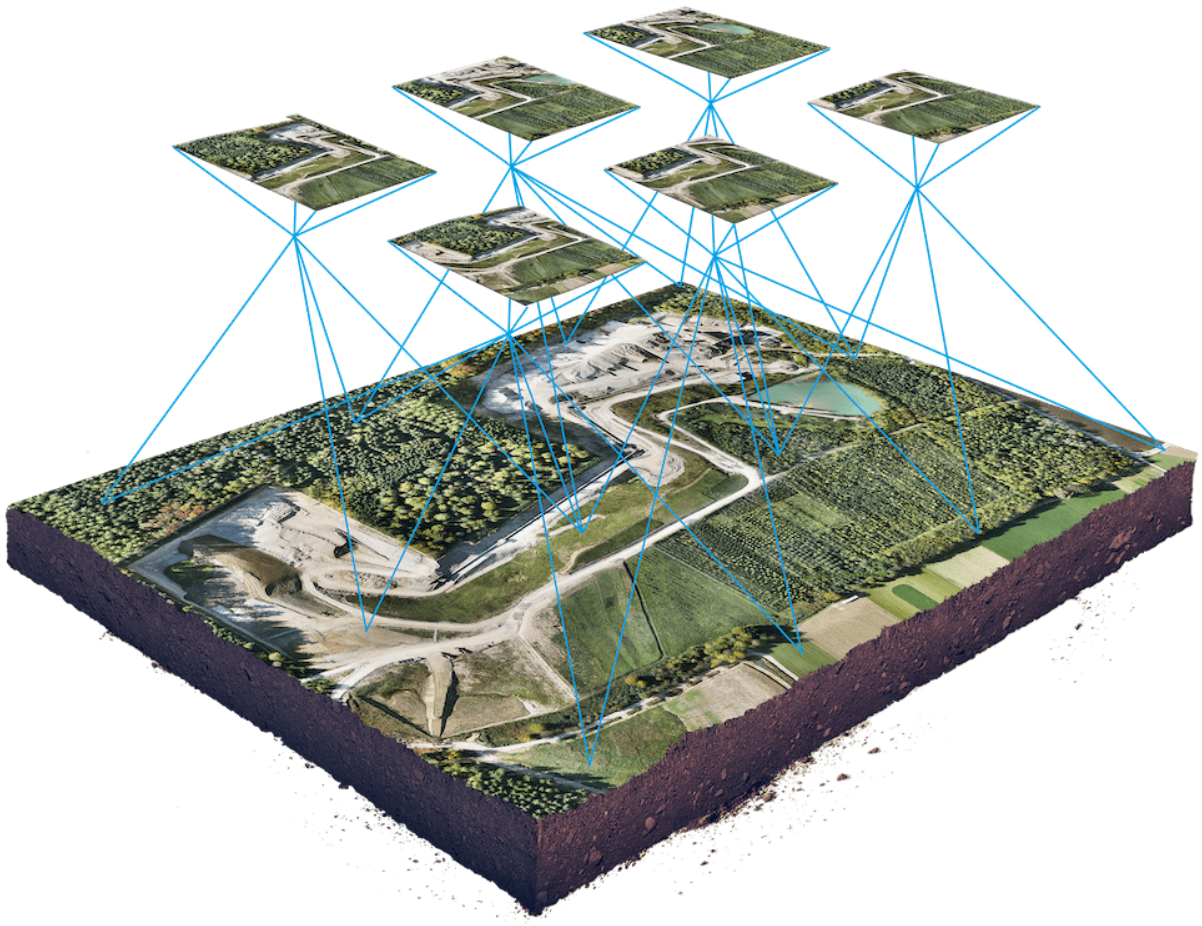
The drones should be autonomous and therefore must have a way to understand their environment. The main method we have come up with is just using LIDAR or using lasers to determine the distance. LIDAR works by sending out pulses of light in a sheet and by reading the returning light, a design of the area is made. LIDAR can be used to judge whether an object is too close to the drone as well as the altitude of the drone. It is precise with high resolution and accuracy and the data that is acquired can be easily transformed into a 3D map. It also has the benefit of not needing to be processed making the sensor very fast, which is useful for our goal of real-time drone processing. The reason why LIDAR isn’t used more is the high cost of the devices and attaching multiple LODAR sensors to each drone would result in a high expense.



Sheet of light to scan terrain using LIDAR [81]

*****

The other option we looked at was using high-resolution RGB cameras for photogrammetry. It works by taking multiple pictures over an area and using the overlap between the images to create a 3D map of the area. Compared to LIDAR, the result is a high-resolution 3D map that displays further detail such as texture, shape, and color. The accuracy of the photogrammetry may be much higher; however, photogrammetry is very slow compared to LIDAR.



Photogrammetry: taking in multiple images [81]

****

After looking into both sensor types, we chose LIDAR as our choice of sensor. The accuracy of photogrammetry is a benefit; however, our goal is to locate a target within a certain timeframe. LIDAR, on the other hand, has the benefit of being faster making it optimal for the timed constraint we have. Ultimately, we decided that the speed of the map creation outweighed the need for an accurate map.

Crash Avoidance

An important component for the search to be successful is the movement of the drone. Within our environment, other things may be moving such as another swarm drone. As a result, we need to ensure that our drones can adapt to the hazards and reroute accordingly. While this would be rare, drones would have to account for the flight paths of other drones to avoid crashing into one another. A drone has many variables to account for such as whether it should go around, stop, or fly over an obstacle. Some of the things we need for our drone to successfully make the right judgment are multiple sensors, object avoidance algorithms, and SLAM.

Having multiple sensors will make sure that the system doesn't have a blind spot and since it is the only view that the drones have of the real world, it has to be through. It should be able to gather all the data from each of the sensors together into a detailed picture for the situation for the drone to accurately make the right decision.

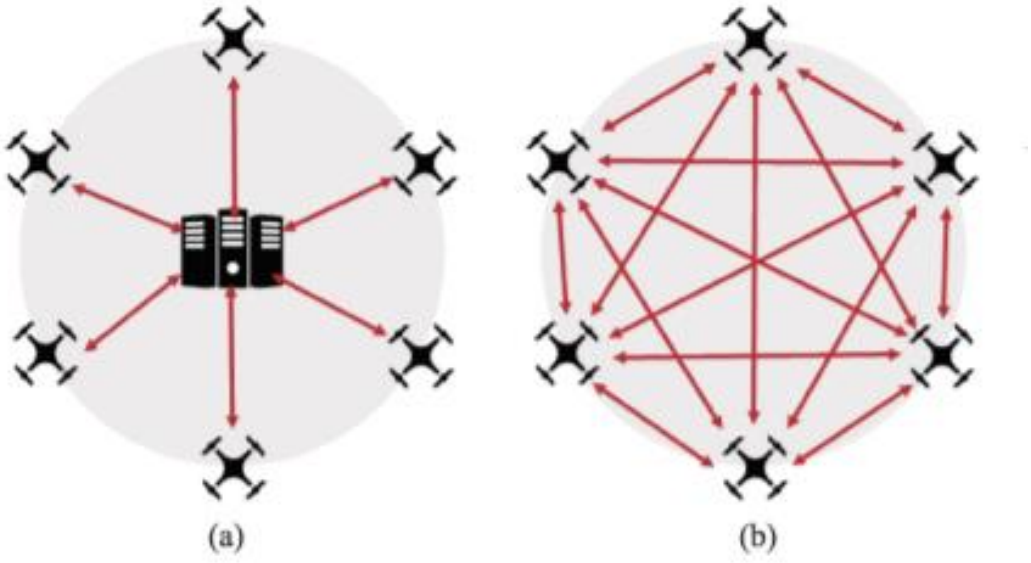
This is where the object avoidance algorithms begin to make the appropriate calculations. The algorithm will detect all of the objects within the data, stationary or in motion. It will then build upon these objects and determine what is a hazard and what is open space.

SLAM is also important in crash avoidance because it maps the surroundings of the drone while simultaneously orienting the drone in its environment. As the drone moves through its environment, the map of the surroundings is updated accordingly. These features are important to make sure the drone knows how to move and what is around it. SLAM is essential because the drone wouldn't know where it was, therefore, it wouldn't know where it was mapping from.

Drone Interaction

A swarm of drones must have a method for how they should interact with each other. The main method that we have been suggested is a central drone that controls the other worker drones around it. One of the problems that were brought up was that a centralized system would fail if that one drone decides to fail.

Our team's idea was to use a distributed system and waypoints for the drone to scout out the environment while mapping it out. This would allow for the drone system to continue even if one of the drones did fail. After the drone has reached a waypoint, the drone would then be assigned a new waypoint.



Centralized vs Decentralized [78]

(a) represents a centralized SLAM system that communicates with a server. (b) represents a decentralized SLAM system where the drones communicate with each other for information and are interconnected

An idea that Dr. Leinecker had was implementing a fast drone and a slow drone. The fast drone would quickly scout an area and if something looks like the target then mark it down. The slow drone is more careful and will look into whether the object is.

5.7 Gazebo Simulator

About Gazebo

Gazebo is an open-source 3D robotic simulator that will help to set up the environment for our drones with the help of ROS. We chose Gazebo as our simulator which was also recommend by our sponsor Lockheed Martin

Gazebo contains different features inside:

- **Dynamics Simulation**

  - Access to multiple high-performance physics engines

- **Advance 3D Graphics**

  - It uses an open-source 3D graphic engine named OGRE. With this Gazebo helps to create realistic environment including high quality lightning, shadows, and textures

- **Sensors and Noise**

  - Gazebo can generate sensor data with the option of noise from lasers range finders, 2D/3D cameras, Kinect style sensors, force torque, and more

- **Plugins**

  - Gazebo can develop custom plugins for robot, sensor, and environmental control

- **Robot Models**

- Gazebo provides some robot models like PR2, Pioneer2, DX, iRobot, and TurtleBot. But you can also build your own

- **TCP/IP Transport**

- It gives the ability to run simulations on remote servers and interface gazebo through socket message passing with the help of Google

- **Cloud Simulation**

- Using CloudSim to run Gazebo on Amazon AWS to interact with the simulation through browser

- **Command Line Tools**

- Gazebo contains an extensive command line tools that facilitates simulation introspection and control

Gazebo environment for MacOS

Why Gazebo?

Gazebo makes it easier to simulate an environment for our robotic system. The simulator makes it possible to test our algorithms faster, design robots, perform regressions for model testing and training. Gazebo provides an AI system which is based on realistic scenarios. It also gives you the ability to simulate populations of robots in complex indoors and outdoors environments. Besides being an open-sourced software for people who have a limited budget, Gazebo comes with a lot of features to use already mentioned.

Setting Up Gazebo environment

- Gazebo contains different versions of itself. Please look at the latest version on http://gazebosim.org/tutorials?cat=install&tut=install_on_mac.

- For this specific installation, the latest version is 11.0.

- To download Gazebo environment for MacOS, follow the next commands:

```
curl -sSL http://get.gazebosim.org | sh
```

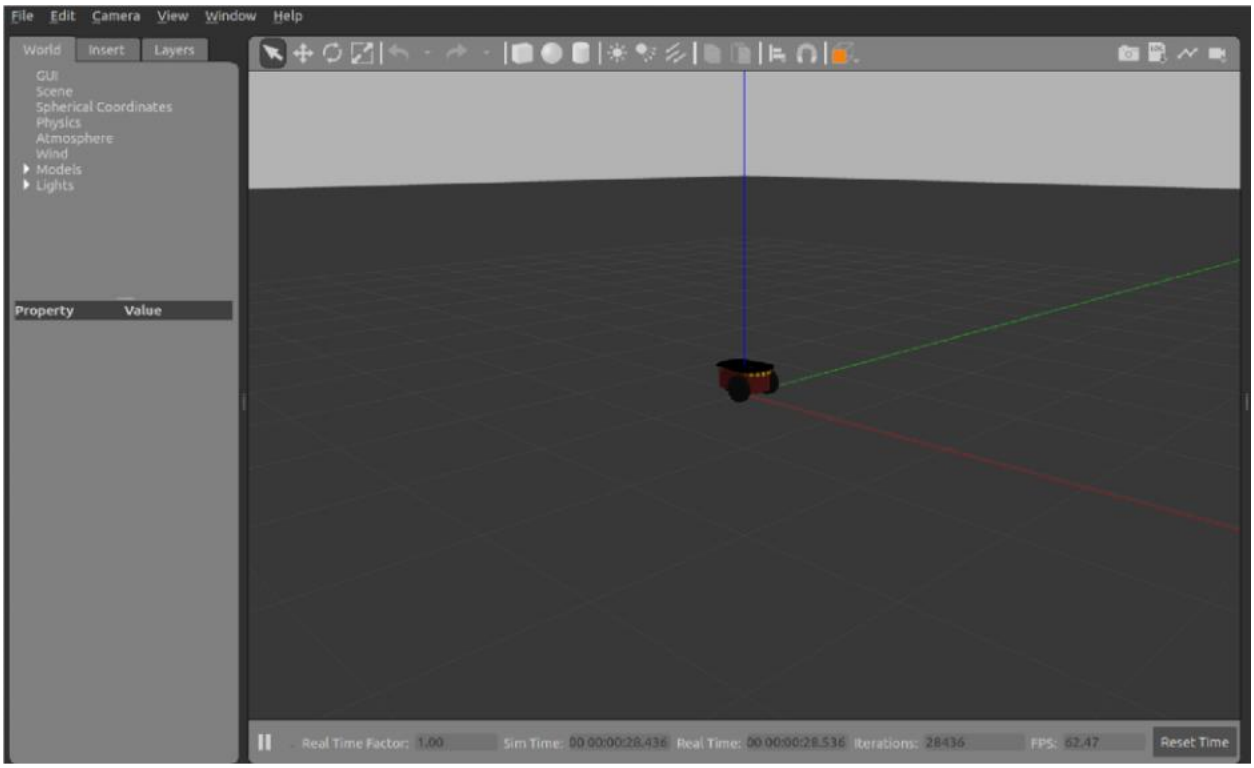
- With this command line, everything that you need for Gazebo to run in your OS will be installed and downloaded. Be aware that there are other ways to install gazebo where you might need to download optional dependencies.

- Finally run Gazebo to start creating the environment

```
gazebo
```

- This an example of gazebo with a robot named pioneer2dx

```
gazebo worlds/pioneer2dx.world
```



Gazebo environment with pioneer robot

Gazebo environment with pioneer robot

- Gazebo already contains some worlds created by default. Most of them can be located inside world folder. It also contains elements in a simulation, including lights, robots, sensors, and static objects with the **extension** ".world"

Client and server separation

The gazebo command runs two different executables. One of these is "**gzserver**" and the other is "**gzclient**". These commands are different and run in two separate terminals.

Gzserver runs the physics of an update-loop and sensor generation. This can be used independently of a graphical interface. The gazebo server reads world files to generate and populate a world.

Gzclient runs a cross-platform QT based user interface.

Model Files

Model Files uses the same SDF format like the world files, but these model files should only contain a single <model>. Once a model file is created it can be included in any world file.

- SDF format example:

```
<?xml version='1.0'?>
<sdf version='1.8'>
  <model name='my_model'>
    ...
  </model>
</sdf>
```

- SDF syntax:

```
<include>
  <uri>model://model_file_name</uri>
</include>
```

SDF Models can vary from simple shapes like rectangles to complex robots. The tag <model> is a collection of links, collision objects, visuals, and plugins.

Links: A link can contain a physical property of one body model. Each link can contain collisions and some visual elements.

• **Collision**

- A collision element is the one that encapsulates a geometry that is used for collision checking. This can be a simple shape.

• **Visual**

- A visual element is to visualize parts of the link.

• **Inertial**

- The inertial elements mostly describe the dynamic properties of a link. Properties like mass and rotational axis.

• **Light**

- Light elements describe the light source attached to a link.

• **Sensors**

- A sensor collects the data from the world so it can be used later in plugins.

Joints: Joints can connect two links. A child and parent relationship can be established along with other parameters like rotation and joint limits.

Plugins: Plugins is shared library created by a third part to control a model

Environment Variables

Gazebo uses several environment variables to locate files and set up communication between the server and the client. Most of the variables are set by default which means that there is no need to set any variables.

Variables

- **GAZEBO_MODEL_PATH:** Variable that sets directories where Gazebo will search for models.

- **GAZEBO_RESOURCE_PATH:** Variable that sets directories where Gazebo will search for other resources like world and media files.

- **GAZEBO_PLUGIN_PATH:** Variable that sets directories where gazebo will search for the plugin shared libraries at runtime.

- **GAZEBO_MASTER_URI:** This variable specifies the IP and port where the server will be started and also tells the clients where to connect to.

- **GAZEBO_MODEL_DATABASE_URI:** URI where Gazebo will download models from.

Gazebo User Interface

Gazebo contains an easy-to-follow interface that can be used for most users. Getting familiar with this interface will be crucial for our project since Gazebo will help us collect all the data according to the environment we create.

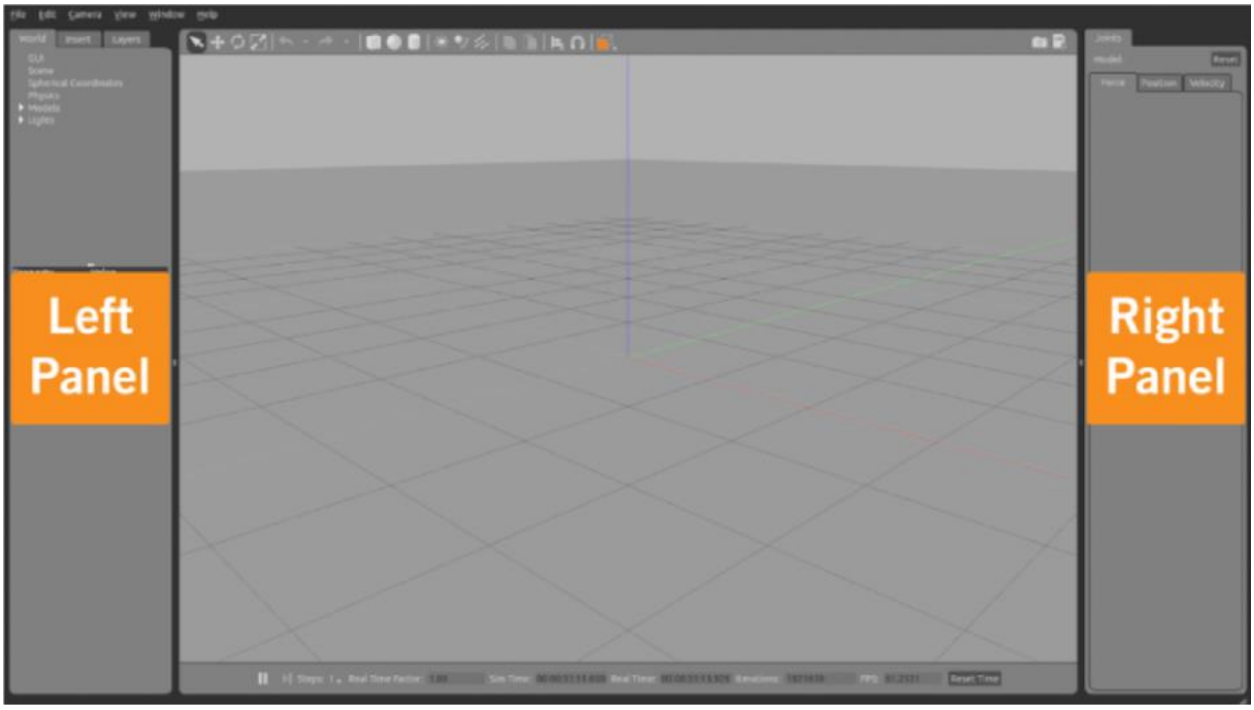
Gazebo UI contains different sections.

The Scene

The scene is the main part of our simulator. This is where all objects will be display and is located in between the left and right panel

The Panels

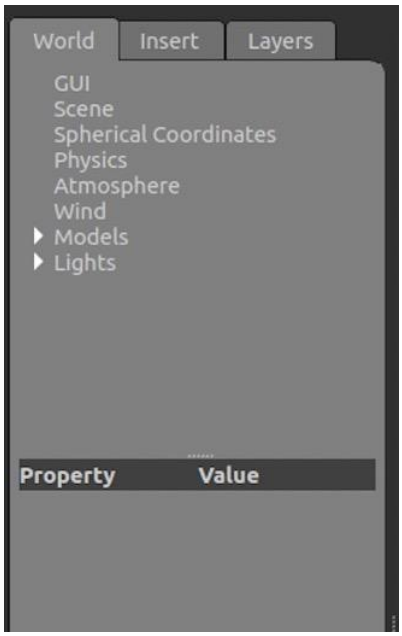
The panels will help to insert and give information of several objects, like force, position, and velocity.



Gazebo environment panels and scene

Left Panel

The left Panel appears by Gazebo default, so you do not need to look for it. Inside this panel we have the following features:



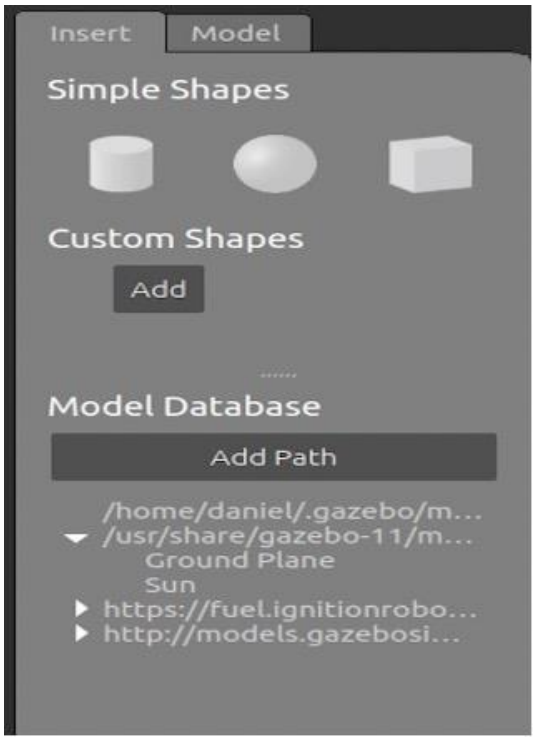
Left Panel

- **World**

    - The world feature(tab) displays the models that are currently in the scene. This allows the user to view and modify the model parameters like their pose, scene, spherical coordinates, physics, atmosphere, and wind.

- **Insert**

    - The insert feature(tab) is where the user can add new objects also named models to the scene or simulation environment.



Insert feature

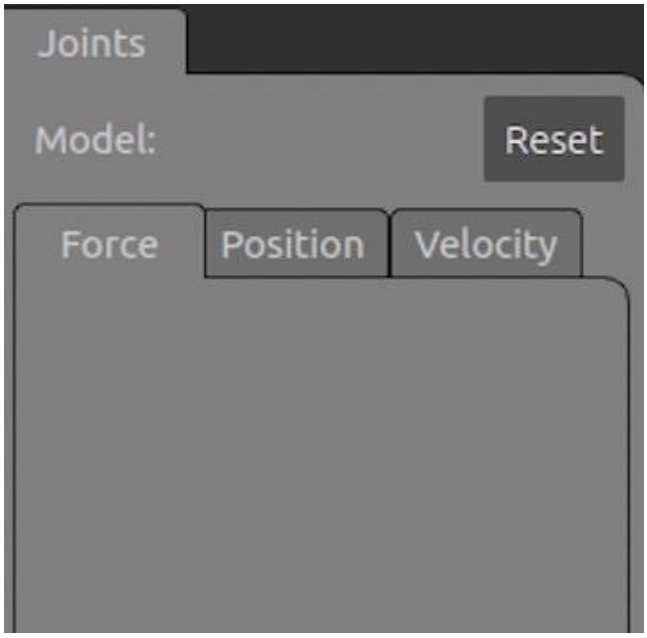
- **Layers**

  - The layer feature(tab) organizes and displays the different visualization groups available in the simulation. Layers can also connect one or more models.

Right Panel

The right panel is usually hidden by default, but the user can easily make it appear by dragging the right side of the environment.

The right panel contains the join model which includes force, position, and velocity.

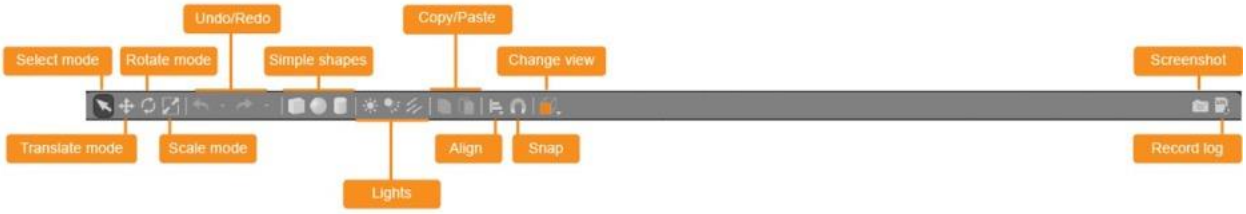


Right Panel

The Toolbars

Upper Toolbar

- The upper toolbar contains different sections that are helpful to create our environment. It gives users the ability to navigate through the scene, select models that you like, make model rotation in 3D, undo/redo actions, insert simple shapes, add lighting to the scene, align models, snap one model to another, and change the view for our model.



Upper Toolbar

Bottom Toolbar

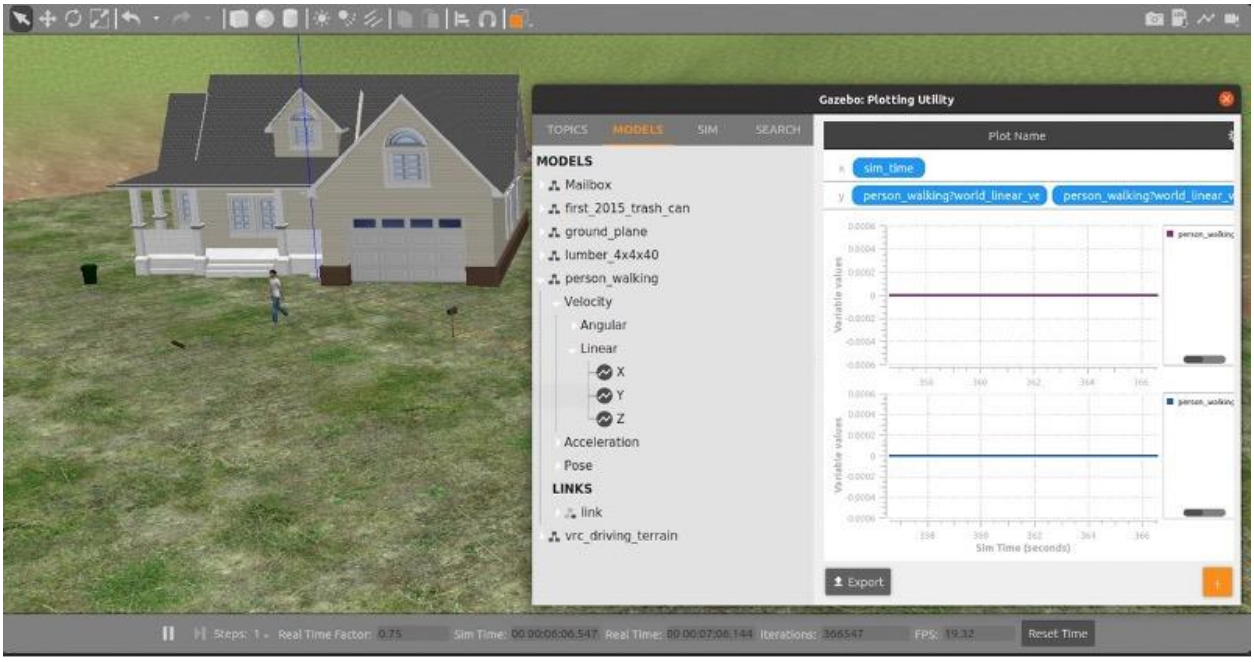
- The bottom toolbar also plays an important part of our simulation environment. The bar can display data about the simulation like the ratio of the simulation time to real time, time passed in the simulation, time passes in real life, the number of internal updates, and the frames per second

Bottom Toolbar

Simple Environment and Data Collection

Gazebo has some models already implemented in their database collection. Models can be objects such as people, cars, mailboxes, restaurants, houses, and anything else that can be displayed in a 2D/3D plane.

Gazebo also has a simple way to collect data. This data can be exported to any desired location. Data can be used to train our AI model. For every Gazebo model there exist different types of specification that you can insert in the x and y graph. Simulation time usually goes in the x-axis to see the changes of the simulation during time simulation which can be different from real time. If we use the model “person_walking” which is already implemented in the Gazebo database system, we can collect different but specific data for this model. The model has the features of velocity, acceleration, and pose. Inside these different features there can be more options. Velocity contains inside the options to graph angular or linear velocity in the x-y-z plane. This can be inserted in a graph where you can collect the data.



Environment and Data export

Training Data

To implement the AI portion of the system, we need an accurate set of training data that the system can learn from. We have training data from the previous team but from the advice of Dr. Leinecker, we determined that their training data is suboptimal. The other problem with their dataset is that their system was also a ground-based vehicle while our robot is an aircraft. As a result, we have to account for variables such as the altitude of the drone and the target. This adds in addition a need for top-down views of the target as well as bottom-up views of the target.

The previous team had a method of automatic data labeling, however, after talking to the sponsor we know that one of the most tedious parts of the training data is having to manually label a large portion of our data.

5.8 Consultants

Due to the novelty of this project, online resources and tutorials are sparse or do not accurately address the same problems and solutions that we are looking for. This coupled with the inexperience of our group in the area of swarm robotics and machine learning as well as the tools used in these areas, such as ROS and Gazebo made it hard to find a starting place and begin to build the solution. For this reason, we reached out to or plan to reach out to multiple consultants who could offer insight or experience to help with the completion of our project.

The first of the consultants we approached was the sponsor of the project, Joseph Rivera. We've kept in touch with him through email and bi-weekly meetings in which we discussed the overall progress of our team as well as the other teams involved in this competition. He guided us on what he wanted as requirements and what the best technologies were to accomplish those requirements. His technology recommendations included:

- Using ROS for the middleware between the drone and the swarm algorithms
- Using Gazebo for the simulation environment
- Using YOLO as an object detection algorithm
- Using SLAM as a mapping/navigation algorithm

He also guided us to the previous group who worked on this project and gave us their GitHub and contact information.

The second consultant we contacted was the project manager from the previous team assigned to this project, Patrick Bauer. We ran into some problems getting their old Gazebo environment up and running on ROS 2, so we contacted Patrick and scheduled a meeting to get some help on how to set it up. After meeting with him we were able to get the simulated environment up and running.

Other consultants we received ideas/help from are Dr. Leinecker, who we met during our status meeting. He gave us the idea about using two different object detection algorithms on our drone agents, one that has a high processing speed to keep up with the frame rate as the drone moves throughout the environment, and another slower but more accurate model that receives the images from the other algorithm that might contain the target. We also plan on meeting with Dr. Gita Sukthankar who is a professor here at UCF who teaches robotic classes.

5.9 Similar ROS Projects

Before starting our project, it was suggested that in our research we look at similar projects that incorporate ROS, drones and object detection. Through our research we were able to find very similar projects and analyze how they achieved their goal. The first project that was read into was a research paper titled “A Framework for Planning and Execution of Drone Swarm Missions in a Hostile Environment”. This paper focuses on a drone swarm mission and identifying hazards in the environment and replanning the swarm’s route. While we are not going to have to apply any rerouting due to hazards in our project, there were a few key insights on how this framework was built to be successful. The first being that route planning was managed by one drone that communicates to a ground control station. This is still a topic of discussion for us, how we should go about communicating between all the agents. The use of a central ground control station for heavier computing seems like a great idea. That way the drones can handle other computations such as object detection. They also used RetinaNet and the YOLO algorithm to process streaming video in real time. Using these resources, they were able to train the YOLO network to recognize military vehicles such as tanks. The paper was able to show the possibilities of using image detection and classification algorithms on lightweight and mobile platforms such as an aerial drone. Moving forward in the development of our project these examples will help guide us in the right direction.

5.10 PX4

Overview

PX4 is an open-source autopilot software for drones. The software is used in a large variety of applications, including even water vehicles and ground drones. By giving a way to maintain hardware along with the software that a drone contains in a useful way, this software is perfect for the project at hand. In this project we are using autonomous drones to search for and identify a target and keeping options open that allow for this type of drone flight is very important.

Creating a PX4 Environment for Ubuntu

In this project we are utilizing a virtual machine to run a Linux based environment. To install and get a PX4 environment up and running is very simple and can be done in just a few steps. The first thing to do is install the PX4 source code directly from the GitHub repository. These are the commands that must be run inside of the ubuntu terminal on whatever machine will be downloading the code.

To download the code:
```
git clone https://github.com/PX4/PX4-Autopilot.git --recursive
```

Running the Ubuntu .sh file in order to install everything needed:
```
bash ./PX4-Autopilot/Tools/setup/ubuntu.sh
```

Once these commands have been run, a simple restart of the computer will finalize the creation of the PX4 Environment and allow for simulations to be run as well as projects to begin being made.

PX4 with ROS Compatibility

We are using the Robot Operating System (ROS) as a middleware software that will allow for the inclusion of algorithms such as an object detection machine learning model to be added. PX4 has ROS compatibility and can be used within a Gazebo launch environment. There are packages within ROS that support the PX4 frameworks and must be downloaded to use the software with it. Another issue is that the PX4software does not have a compatibility with ROS 1, which we are using for this project, however, a bridge can be made between the two versions to make it work. These are the steps to take in order to get a PX4 environment compatible with ROS up and working:

Step 1: Make a workspace directory for ROS and PX4

```
$ mkdir -p ~/px4_ros_com_ros1/src
```

Step 2: Clone the repositories of the ROS 1 bridge packages into the workspace

```
$ git clone https://github.com/PX4/px4_ros_com.git
~/px4_ros_com_ros1/src/px4_ros_com -b ros1 # clones the 'ros1' branch
$ git clone https://github.com/PX4/px4_msgs.git
```

Step 3: Run the bash script to fully build the environment and create a working connection between PX4 and ROS

```
$ git checkout ros1
$ cd scripts
$ source build_ros1_bridge.bash
```

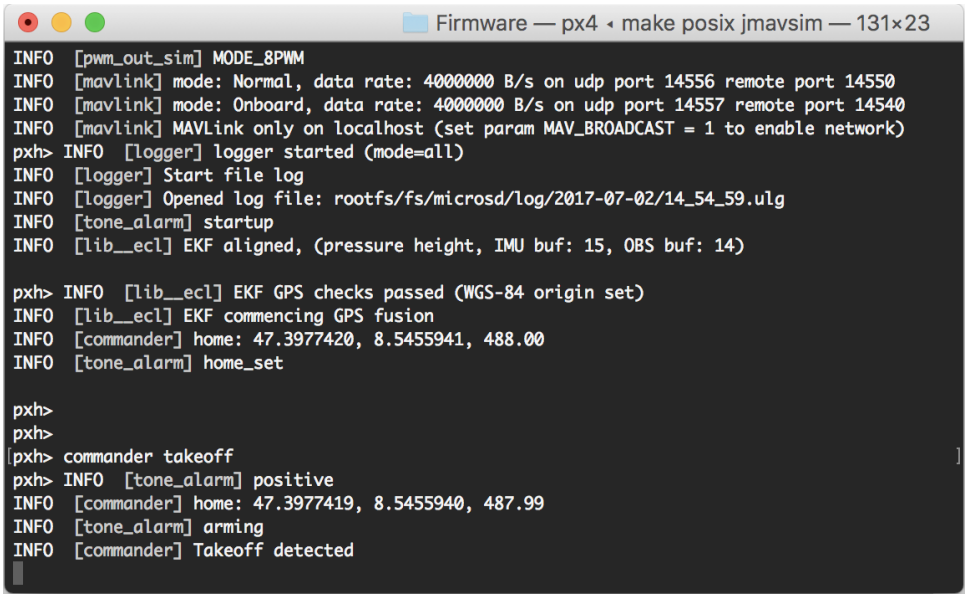
These are all of the steps needed to create a workspace that can utilize both ROS and PX4 together. Once everything is installed the next step is to test the environment and make sure that everything is working properly.

Building and Testing PX4 Software

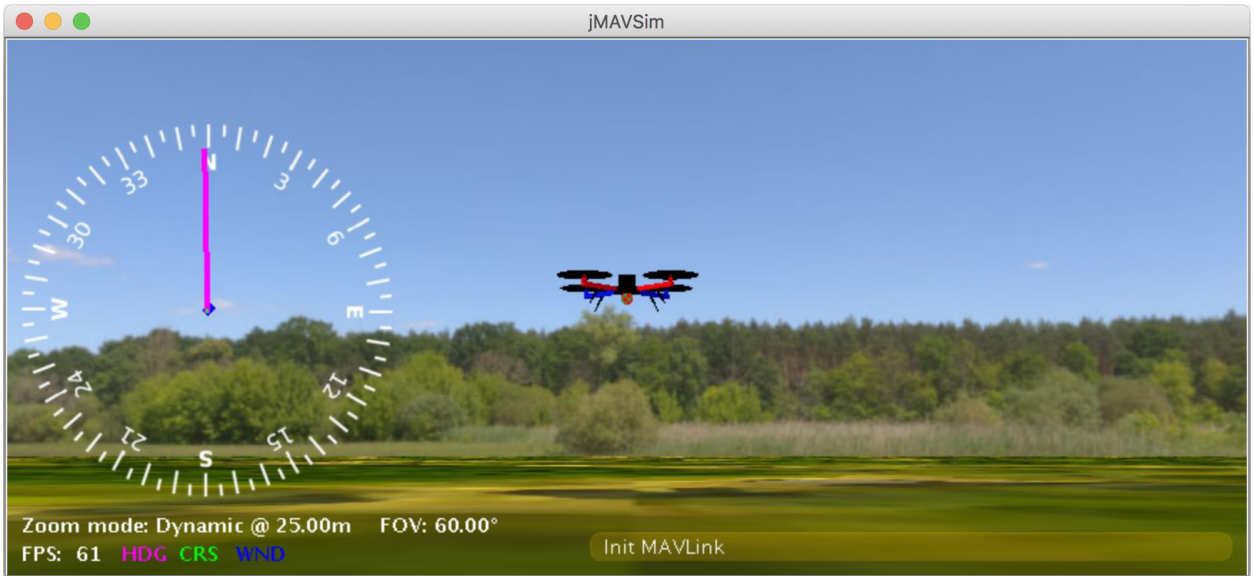
Once all of the code has been downloaded and the appropriate packages have been installed testing to make sure everything works is very important. To do that the PX4 document recommends using the built in jMAVsimulator. To run the simulation, navigate into the directory where the PX4 environment is setup using a command terminal and run the following code:

```
make px4_sitl jmavsim
```

This will start the simulation which can be confirmed by this message in the console and this window appearing containing the jMAVsimulation:



[26] Console Message Running jMAVsimulator in PX4



[26] jMAVsimulation Window

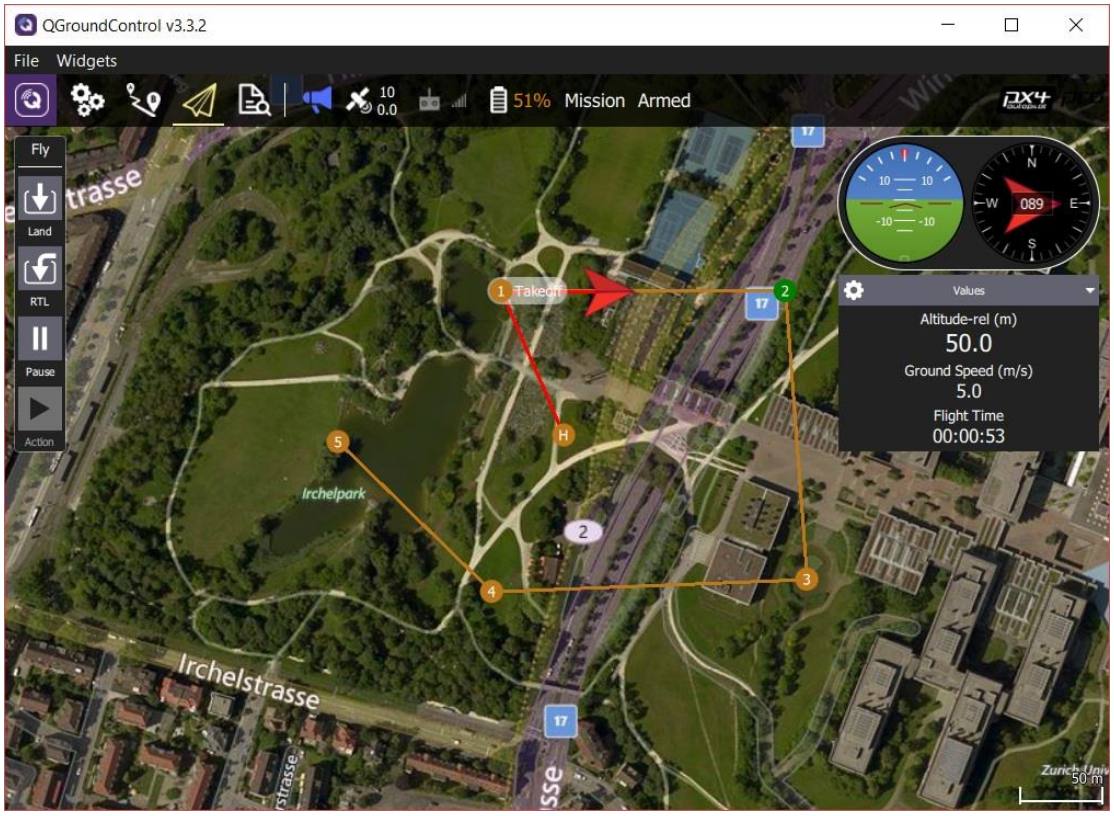
At this step, the user can type commands to control the drone that is being simulated. By running this command:

```
pxh> commander takeoff
```

The drone will begin its flight and the testing process for the workplace of PX4 and ROS will now be complete.

PX4 Missions

Missions are the manual or autonomous simulations or run throughs that a drone will be undergoing. Controlled through a user interface, the mission interface acts as the central hub for everything a drone will be doing during its actual mission. There are ways to access a terminal and gain manual control over autonomous bots which is useful for the testing and utilization we will be using the drones for.



[24] PX4 Mission Interface

In our case, we will be using a gazebo simulated world, however this interface will still look the same. The utility that the interface has works with gazebo flawlessly and allows us to utilize this software to help keep track of all of the different variables that we need to account for.

Controlling the Flight of the Drones

An obviously big part of what this project is involves the actual flight of the drones. Luckily there are many different options that PX4 provides to control the altitude and ways of flying that the drones will use. Switching between each mode will be something that the drones will be able to do automatically, but correctly implementing each of the ones our drone will be using is the important part. Some of the different modes and their uses are listed here:

- Takeoff Mode - This mode is used at the start of a mission. The drone will fly to a certain altitude away from the starting location. This mode ends once it has reached its target height where it will then wait for further instructions.

- Mission Mode - Once the drone has reached its target height in the air it will begin this mode which can be a mix of different modes. This will be the mode we will be using in this project as a mission will be predefined and created beforehand and then the drone will have to carry this mission out.

- Return Mode - The final mode before the drone is done with its task, this mode is used to have the drone safely fly to the starting position in a safe way avoiding obstacles and taking the shortest path possible.

These are only three of the modes that PX4 has available for use. They are the most prevalent to the task at hand as they can all work autonomously and are perfect for what this project is trying to accomplish.

Obstacle Avoidance using PX4

Our drone will be flying through the simulated world and be required to avoid all of the obstacles that come with doing such a task. Buildings, streetlights, and other physical objects may block a drone’s path and add an extra challenge to the overall objective of finding the target. Luckily PX4 has an object avoidance package included in the software that proves to be useful in our case. By importing the package and then activating the avoidance mode on the mission interface, the drone will search for any machine learning algorithms or sensors present. By utilizing the LIDAR sensors that the drones have, they can detect how far away the objects closest to them are. By implementing a simple algorithm that tells the drone to stay a certain distance away from all of the objects it comes close to, obstacle avoidance is achieved.

[69] Depiction of Drone Using a Sensor to Avoid an Obstacle

Multi Vehicle Simulation Using PX4

PX4 supports simulating multiple drones within ROS and Gazebo. By installing and implementing the correct firmware code that PX4 provides, having more than one drone in a simulation at one time becomes a breeze. Cycling between each of the vehicles and making sure that they are all doing what they are supposed to is a little challenging. By using the command line there are multiple ways to go from analyzing one drone to another. In our project this is exactly what we need as we will be using a swarm of multiple drones to work together and identify a target.

5.11 MAVlink

As there will be many drones in the Gazebo environment inter-drone communication is vital for successful operations. This creates a need for a standardized way for the swarm drones to efficiently communicate with one another. MAVLink solves this need. MAVLink provides a protocol for sending and receiving messages to and from drones. It follows a standard publish/subscribe and point-to-point design pattern where drones subscribe to which messages they want to receive and publish which messages they want their subscribers to see. The messages are sent as topics (see 5.1 for more info) with the mission protocol and parameter protocol being point-to-point.

The messages that are sent are defined within XML files and are tied to a certain MAVLink system denoted by its “dialect” [68]. Here is the general format of one of these messages:

```
<?xml version="1.0"?>
<mavlink>

    <include>common.xml</include>
    <include>other_dialect.xml</include>

    <!-- NOTE: If the included file already contains a version tag, remove the version tag here, else uncomment to enable. -->
    <!-- <version>6</version> -->

    <dialect>8</dialect>

    <enums>
        <!-- Enums are defined here (optional) -->
    </enums>

    <messages>
        <!-- Messages are defined here (optional) -->
    </messages>

</mavlink>
```

Sample MAVLink XML file

MAVLink also provides support for up to 255 vehicles making it perfect for a communication protocol to be used in a drone swarm. MAVLink has support for ROS in the form of a package called MAVRos which is what will be used on the drones. These are steps to get MAVRos setup [68].

1. Follow the MAVRos installation instructions
2. Uninstall the MAVLink package for ROS by entering the command below:

   ```
   sudo apt-get remove ros-${rosversion -d}-mavlink
   ```

3. In mavlink-gbp-release, add the new MAVLink message to common.xml
4. Run the command below:

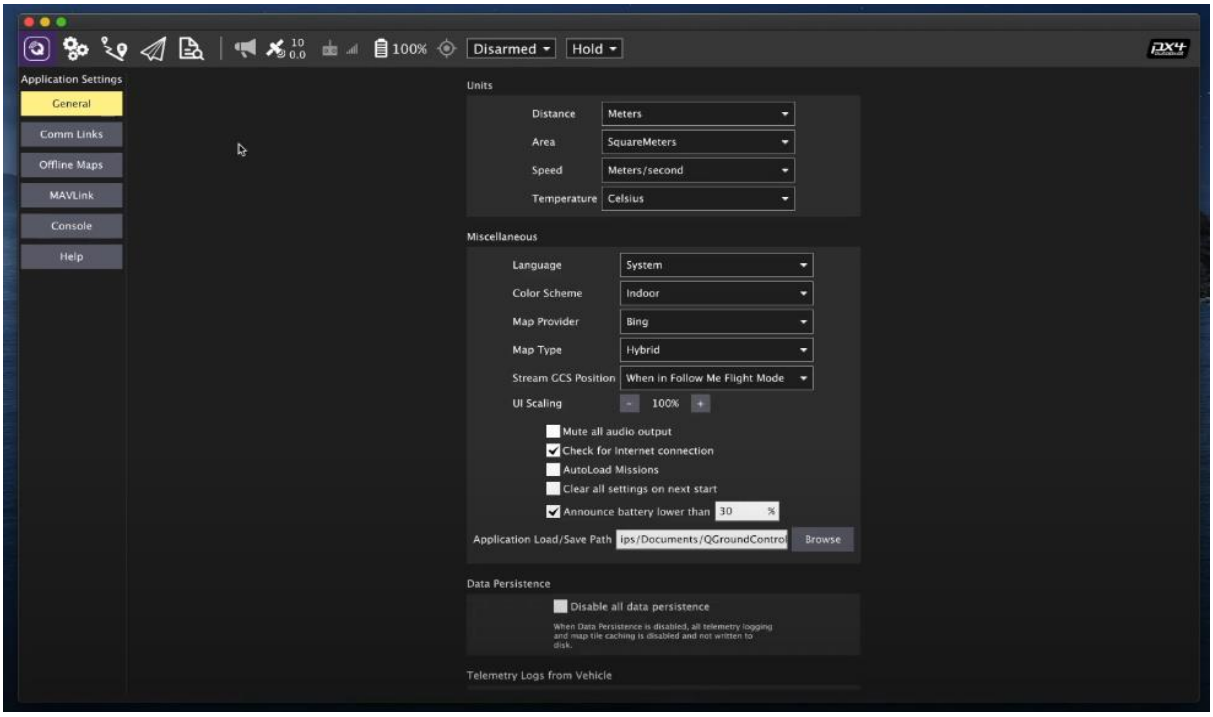
   ```
   catkin build mavlink
   ```

5.12 QGroundControl

QGroundControl is a full flight control system built for MAVlink based drones. It has an intuitive, easy to learn UI made for beginners. While also still being capable of high-end features for more skilled users. The key features listed on the QGroundControl website:

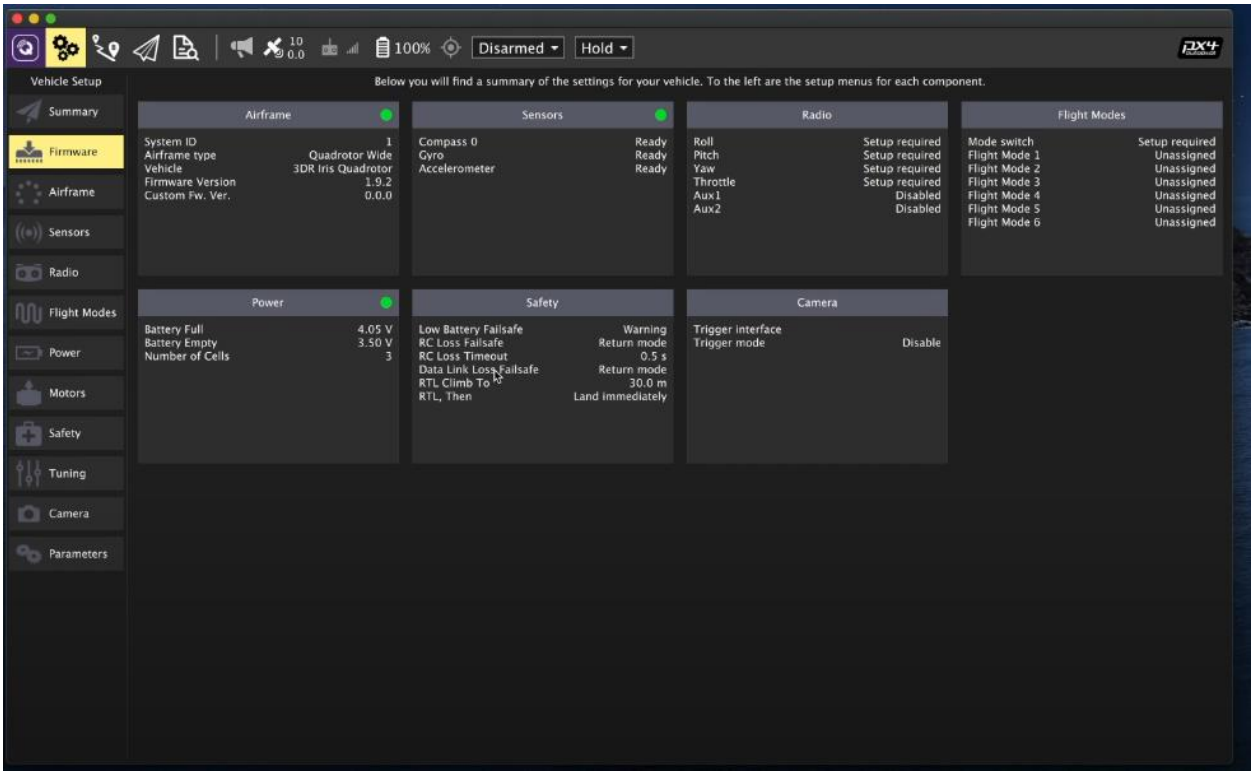
- Full setup/configuration of ArduPilot and PX4 Pro powered vehicles.
- Flight support for vehicles running PX4 and ArduPilot (or any other autopilot that communicates using the MAVLink protocol).
- Mission planning for autonomous flight.
- Flight map display showing vehicle position, flight track, waypoints and vehicle instruments.
- Video streaming with instrument display overlays.
- Support for managing multiple vehicles.
- QGC runs on Windows, OS X, Linux platforms, iOS and Android devices.

After installation of QGroundControl, opening the application will bring you to its settings page. From here you will be faced with 6 tabs on the right. General is the main place for application-level configuration. You can set values for settings like display units, fly view, data persistence and more. Comm Links allows users to make communication links and connect to them. Offline maps allow users to store map data to use when not connected to the internet. MAVLink settings allow users to configure all options for MAVLink communications. Lastly the console tab provides a console to debug any issues while using QGC.



Settings tab of QGroundControl

The next tab on the top of the UI is the vehicle setup tab. Here users are able to access all settings related to the drone. The first tab from the left that will be opened is the summary tab where you are able to see a full summary of setup options for the vehicle. Next would be the firmware tab where users can update or even downgrade firmware of the flight controller. Airframe allows users to select a basic structure of the drone they are flying. This will set up different tuning values for flight parameters. The sensors tab gives full access to all the different sensors attached to your drone. Flight modes provide a series of predefined flight patterns that a user can assign a drone to fly. These are just the important tabs that standout in the vehicle setup; there are also even more allowing for more control and customization for your flight mission.



Vehicle setup tab of QGC

The number of settings and options you can easily access while using QGC is what makes it such a useful tool for us. If we are going to use the PX4 flight stack or any MAVLink drone this software would be a must for mission control.

6 Explicit Design Summary

6.1 Images for Yolo Algorithm

Using our simulated world, we place the target in different locations that have different lighting and position specifics to hit a wide range of possibilities. We can take images with the drone's camera to imitate how the drone will be taking pictures during the trials.

For this project, we use LabelImg to label each image that we take. LabelImg is a software that makes the labeling process less tedious. It allows each photo from a file to be loaded, and then for a bounding box to be drawn around each image that will then be converted into a label file that our YOLO algorithm can use. The steps for using LabelImg as are follows:

Step 1: Run LabelImg.exe and then select the directory that contains the images to be labeled.

Step 2: Once a directory has been loaded the save directory must also be changed. For this step, the process is exactly the same as step one because the save directory is going to be the same as the directory where the images are.

Step 3: We change the setting to make sure that the format of the labeled images is correct for YOLO and then we begin to label the images by adding the ground-truth bounding box around each image. An example of drawing a bounding box is provided below.

This process is done for each image in the entire training set to obtain a fully custom dataset to train. The resulting file will contain each image with a txt file in accordance that describes the position of the bounding box around the object.

6.2 Training the Model on Google Colab

First you must create an account that has access to a google drive, and then create a notebook that will contain the code for training the model. For this step the tutorial on towardsdatascience.com was used to train a custom dataset of images. Each of the steps are going to be down in their own cell of the colab notebook.

Step 1: Connect google drive with the google colab notebook. Running the code below in the cell that it is in will provide a text box and link that must be clicked and followed through in order for the colab notebook to access the google drive.

```
from google.colab import drive



drive.mount('/content.gdrive')

!ln -s /content/gdrive/My\ Drive/mydrive

!ls /mydrive
```

Step 2: Make sure to convert the runtime hardware accelerator to use the GPU provided. This is done by selecting ‘Runtime’, then ‘Change Runtime Type’, then using the hardware accelerator drop down to change the input from ‘none’ to ‘GPU’. One this has been done the next step is to use a cell to confirm that you are using the GPU, done by the code below.

```
!nvidia-smi
```

Step 3: We clone the darknet repository into our notebook. Darknet is open-source and contains the neural network framework that the YOLO algorithm utilizes while training the models. The cloning process is as shown below:

```
import os
os.environ['PATH'] += ':/usr/local/cuda/bin'
!rm -fr darknet
!git clone https://github.com/AlexeyAB/darknet
```

Step 4: First we go to the darknet folder. We then edit some of the files and then remake the build.

```
%cd darknet
!sed -i 's/GPU=0/GPU=1/g' Makefile
!sed -i 's/OPENCV=0/OPENCV=1/g' Makefile
!sed -I 's/CUDNN=0/CUDNN=1/g' Makefile
!make
```

Step 5: We now configure darknet network for training YOLO

```
!cp cfg/yolov3.cfg cfg/yolov3_training.cfg
!sed -i ‘s/batch=1/batch=64/’ cfg/yolov3_training.cfg
!sed -i ‘s/subdivisions=1/subdivisions=16’ cfg/yolov3_training.cfg
!sed -i
‘s/max_batches = 500200/max_batches = 4000’ cfg/yolov3_training.cfg
sed -i '610 s@classes=80@classes=1@' cfg/yolov3_training.cfg
!sed -i '696 s@classes=80@classes=1@' cfg/yolov3_training.cfg
!sed -i '783 s@classes=80@classes=1@' cfg/yolov3_training.cfg
!sed -i '603 s@filters=255@filters=18@' cfg/yolov3_training.cfg
!sed -i '689 s@filters=255@filters=18@' cfg/yolov3_training.cfg
!sed -i '776 s@filters=255@filters=18@' cfg/yolov3_training.cfg
!wget https://pjreddie.com/media/files/darknet53.conv.74
```

Step 6: Create a directory in the drive to save the trained weights

```
!mkdir "/mydrive/yolov3"
!echo "target" > data/obj.names
!echo -e 'classes= 1\ntrain  = data/train.txt\nvalid  = data/test.txt\nnames =
data/obj.names\nbackup = /mydrive/yolov3' > data/obj.data
!mkdir data/obj
```

Step 7: Extracting the images for use. The folder containing the training images and positions should be in a folder called images.zip and be located in yolov3 on google drive.

```
!unzip /mydrive/yolov3/images.zip -d data/obj
import glob
import os
import re
txt_file_paths = glob.glob(r"data/obj/*.txt")
for i, file_path in enumerate(txt_file_paths):
        with open(file_path, "r") as f_o:
        lines = f_o.readlines()
        text_converted = []
        for line in lines:
        print(line)
        numbers = re.findall("[0-9.]+", line)
        print(numbers)
        if numbers:
                # Define coordinates
                text = "{} {} {} {} {}".format(0, numbers[1], numbers[2], numbers[3],
numbers[4])
                text_converted.append(text)
                print(i, file_path)
                print(text)
        # Write file
        with open(file_path, 'w') as fp:
        for item in text_converted:
                fp.writelines("%s\n" % item)
```

```
import glob
images_list = glob.glob("data/obj/*.jpg")
print(images_list)
file = open("data/train.txt", "w")
file.write("\n".join(images_list))
file.close()
```

Step 8: Start the training on the data

!./darknet detector train data/obj.data cfg/yolov3_training.cfg darknet53.conv.74 -dont_show

The model itself will need hours of training in order for the weights to be as accurate as possible, so the google colab notebook will run overnight.

Testing the Model on New Images

Step 1: Utilize the testing script that the YOLO tutorial provided. We first downloaded the "yolov3_training_last.weights" file that is located in our google drive. This file is the final result from training our model, and will be used in the ROS implementation of our drones.

Step 2: Edit the "yolo_object_detection.py" file to use the correct name of the object that we have labeled and the file location of the test image. For this project the name of the object being labeled is "target" so this is what line 11 of the file will be changed to. On line 14 we must edit the path to the images that will be used for testing.

```
7  # Load Yolo
8  net = cv2.dnn.readNet("yolov3_training_last.weights", "yolov3_testing.cfg")
9
10 # Name custom object
11 classes = ["target"]
12
13 # Images path
14 images_path = glob.glob(r"C:\Images\testdataset")
15
```

Step 3: Run the test. We run the script to see the resulting boundary box on the images included.

6.3 Environment Layout

The environment where our drone will search for our drone will be a prior team's environment. The environment has already been approved by our sponsor and therefore should meet all of the requirements set for the environment. Note that since the sponsor assumed we would use the previous team's environment, we were never given an explicit list of requirements for our environment.

Some of the requirements that were stated during our sponsor meeting are:

- Urban environment
- Should have buildings and structures
- 100 ft by 100 ft (previous teams required 400 ft by 400 ft)
- Random objects/distractors/clutter (to throw off the drones from the target)

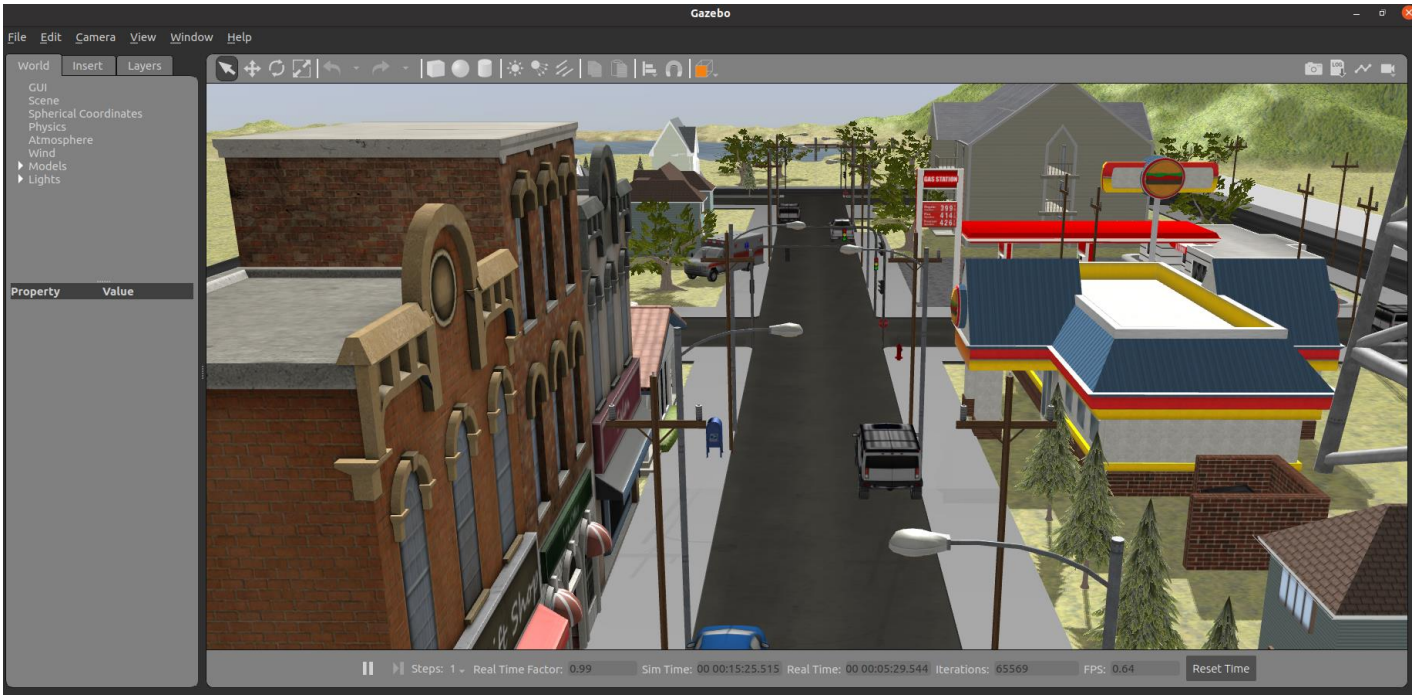


Figure: Gazebo Environment

Environmental Construction

Gazebo comes with some worlds already built and ready to use in the directory of worlds. This specific environment was made according to the specifications given by the sponsor. The world represents an urban environment where our drones will be able to navigate and find the desired target object location.

The urban is inside a script that can be called by command of “gazebo urban.world”, this command line will be executed, and the world will be displayed. The environment contains roads, cars, buildings that are going to be used as obstacles or objects in order to test our algorithm for image detection.

7 Administrative Content

Our project involves many moving parts and without proper management the project could become disorganized and end with failure. To avoid that, we created a plan for the future which describes where we want to be. We also decided to date every project-relevant meeting with dates and the tasks of that meeting. We also kept track of the cost of the project to avoid going over budget or using more than we are financially able to.

7.1 Meeting Schedule

The timeline for our project will require many meetings throughout its lifespan, especially with the addition of a sponsor, which means bi-weekly status meetings for our projects. We decided that it would be best to keep track of these meetings to see what we should meet for and what we should further put effort into. We can also track all of the external meetings we have outside of our regularly scheduled meetings.

Fall 2021

Dates	Tasks	Team meetings
9/15/2021	Team formation	Boot Camp
9/24/2021	Team assignments	First Sponsor meeting
9/29/2021	Complete first 15 pages	Writing 1st 15 pages and TA meeting
10/1/2021	ROS discussion	Team meeting
10/3/2021	ROS/Gazebo discussion	Team meeting
10/8/2021	Creating class presentation and learning requirements	Team meeting(presentation) & Sponsor Meeting (Requirements)
10/13/2021	Individual 15 pages and powerpoint	Team meeting
10/17/2021	PowerPoint for class	Team meeting
10/19/2021	Presentation day	Meeting with Leinecker
10/20/2021	ROS/Gazebo troubleshoot meeting (operating systems) and individual 15 pages	Team meeting
10/21/2021	ROS/Gazebo troubleshoot meeting (operating systems) and individual 15 pages	Team meeting
10/24/2021	ROS/Gazebo troubleshoot meeting (operating systems) and individual 15 pages	Team meeting
10/25/2021	Individual 15 pages submitted	
10/29/2021	No sponsor meeting/team meeting on environment	Team meeting
11/2/2021	environment discussion	Team Meeting
11/5/2021	No sponsor meeting/team meeting on ROS	Team Meeting
11/9/2021	Update TA/discuss our environment	TA meeting II/Team meeting on environment
11/15/2021	No sponsor meeting/team meeting on	Team meeting on environment

	environment	
11/16/2021	Fix ROS distribution and environmental troubleshooting	Team meeting
11/17/2021	Discuss Design Document and combine document together	Team meeting
11/19/2021	Speak to previous team about environmental problems	Meeting with previous team
11/22/2021	Combine document together and apply environment solution to team	Team meeting
11/23/2021	Meeting (fixing ROS launch)	Team meeting
11/26/2021	Brainstorming sections for design document	Team meeting
11/27/2021	Assigning sections to team members	Teem meeting
11/29/2021	Learn about our SD2 tasks and presentation for Lockheed Martin	Sponsor Meeting (Lockheed Martin Presentation)
11/30/2021	Brainstorming more sections for design document	Meeting (Design Document)
12/1/2021	Add pages	Meeting (Design Document)
12/2/2021	Add pages	Meeting (Design Document)
12/3/2021	Add pages	Meeting (Design Document)
12/4/2021	Add pages	Meeting (Design Document)
12/5/2021	Add pages	Meeting (Design Document)
12/6/2021	Design Document submission due	

7.2 Budget and Financing

The budget we have to complete this project is three hundred dollars funded by Lockheed Martin. The major technical requirements, ROS and Gazebo, are open-source and need no additional fees to operate. The reminding technologies are also open-source or offer a free version that satisfies our needs.

The free version of GitHub will be all we need to use to share our project amongst ourselves and lay out the tasks for the project. Python is the main language we will be using along with any of its machine learning algorithms, which are all open source. We will also be using and sharing this through Google Collab which doesn't require any cost. Discord is our main form of communication. We don't need any of the paid versions of discord because we only need the channels to separate out our tasks and the voice channel to hold meetings. To communicate with sponsors, we will swap over to Zoom, a free app. For our project management, we decided on Jira because out of the project management alternatives, it offered the most features on its free version.

Most of the technologies for the project are free or open source. The only possible cost we have is further educating ourselves through a course. While there are many free options for learning a skill, we may need a specialized course which could cost money to access.

Fall 2021

Software / Tool	Cost (without taxes)
ROS course, The Construction Sim (entire team) https://www.theconstructsim.com/robotignite academy_learnros/ros-courses-library/	$0.00
Intro Robotics Developer Course - Using ROS in Python (Bryce) https://www.udemy.com/course/intro-robotics-developer-course-using-ros-in-python/	$84.99 (original) $11.99 (with discount) Status: Pending
ROS Ultimate Guide for Beginners with TurtleBot3 and Robot https://www.udemy.com/course/the-ultimate-guide-to-ros-simulate-your-robots/ (Huy)	$84.99 (original) $11.99 (with discount) Status: Pending
ROS for Beginners II: Localization,	$17.99 (original) × 3 = $53.97

Navigation and SLAM (Keifer, Daniel, Raymond) https://www.udemy.com/course/ros-navigation/	$119.99 (with discount) × 3 = $357.57 Status: pending
Robotic Operating System	$0.00, Open-Source Software
Gazebo	$0.00, Open-Source Software
Python	$0.00
Google Collab	$0.00
GitHub	$0.00
Discord	$0.00
Total:	$0.00

Courses Explanation:

- ROS course, The Construction Sim: assigned to every teammate because every team member should have a general understanding of ROS. The course is also free, so it doesn't cut into our budget.
- Intro Robotics Developer Course - Using ROS in Python: assigned to the robotic (ROS) specialist because this course goes further into ROS and its tools and features.
- ROS Ultimate Guide for Beginners with TurtleBot3 and Robot: assigned to the project manager because it is outside of the specialization of the other team members but still goes over creating custom robots using Gazebo and LIDAR sensor data.
- ROS for Beginners II: Localization, Navigation and SLAM: assigned to the simulation, machine learning, and path-finding specialist because the course focuses a lot on the machine learning aspect of how the drone will move around. This will be very important to the three of their specializations and it will allow them to work together and discuss topics if they have a similar background in the topic.

Currently everything we have is pending because the financial advisor for the computer science department hasn’t processed our requests.

7.3 Project Milestones

We created a general milestone schedule for our project to make sure we were on track. This schedule is very ideal and made without much knowledge on how difficult and time consuming the implementation and learning process will take up. There will be many factors that might slow down our progress such as our other coursework as well as external hurdles. As we move further into the project and deadlines grow nearer, we will adjust or rush milestones to ensure the project is completed on time.

Fall 2021

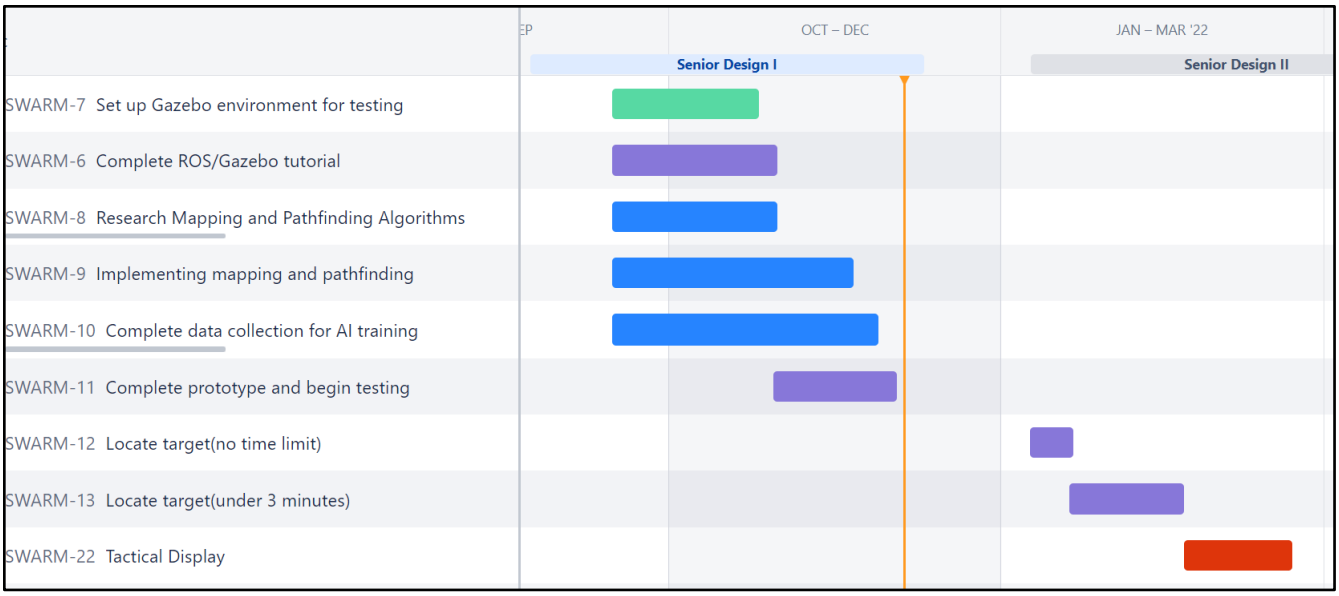
Date	Milestones
9-15-2021	Team Formation
9-24-2021	Sponsor Meeting
9-28-2021	Motivational Statements
9-29-2021	TA check-in I
9-30-2021	Initial Design Document Due
10-1-2021	Research: - Huy: Simulation, sensors, YOLO, additional pathfinding alternatives - Bryce: ROS, training data, robot communication, sensors - Daniel: Prior team environment, Gazebo, sensors - Keifer: YOLO, SLAM, additional object-detection and pathfinding alternatives - Raymond: SLAM, computer vision, sensors
10-2-2021	Begin installing and testing technologies

10-30-2021	Complete ROS/Gazebo tutorial
11-8-2021	TA check-in II
11-20-2021	Learn mapping/pathing algorithms
11-30-2021	- Setup functional Gazebo environment - Test drone with video attached - Test LIDAR
11-30-2021	Complete any necessary data collection/ create training set
11-23-2021	Final Design Document Draft, proofread
12-2-2021	Prototype, begin testing
12-6-2021	Final Design Document Due
12-11-2021	Class ends (winter break begins)

Spring 2022

Date	**Milestones**
1-10-2022	Class begin/first sponsor meeting of the semester
1-15-2022	- Video output from drones - Final drones chosen - Final sensor chosen - Apply pathfinding/object-detection algorithm
1-17-2022	Presentation due for Lockheed Martin engineers with: - Video status - Design Documents

	- Block diagram - Design diagram
1-25-2022	Presentation day for Lockheed Martin
1-20-2022	Drones locate target, any amount of time
1-30-2022	Begin tactical display implementation
2-2-2022	Finalize the pathfinding/object-detection algorithm
2-17-2022	Finalize implementing pathfinding/object-detection algorithm
2-20-2021	Drones locate target, under 3 mins
2-26-2021	Competition of different algorithms: - Measure time against precisions - Take into account what requires the least hardware
3-6-2021 to 3-13-2021	Spring break
3-20-2022	All components fully integrated
3-21-2022	Complete Unit Testing
4-??-2022	Final presentation
4-25-2022	Class ends



Graph: Roadmap of our major milestones [Jira]

8 Project Summary and conclusions

Senior Design I

We have completed the research necessary to begin our project, however, we didn't account for all the issues that would appear. Most of our issues involve compatibility, scope, and time. The technologies we tested throughout the semester were sometimes just not compatible with one another, so for example after setting up a version of ROS, we might later learn it might not be compatible with our drone. The other issues come with scope and time. We were working harder than we should be on pieces of the project that weren't within the scope of our project or may just take too much time to include. Now, we have a reliable list of the technologies/versions our team wants to use and where we want to take our project in Senior Design II.

9 References

1. *BB-8 Robot Simulation*. The Construct. (2021, January 26). Retrieved December 5, 2021, from https://www.theconstructsim.com/bb-8-robot-simulation/.
2. Auto-Label: Part 1. Introduction to Superb AI's Auto-Label Tech. (2020 July 10th). Retrieved December 4th, 2021, from https://www.superb-ai.com/blog/auto-labeling-data-workflow
3. "Wiki." *Ros.org*, http://wiki.ros.org/mavros#Overview.
4. "Introduction." *Introduction · MAVLink Developer Guide*, https://mavlink.io/en/.
5. "Wiki." *Ros.org*, http://wiki.ros.org/micros_swarm_framework#Overview.
6. Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016, May 9). *You only look once: Unified, real-time object detection*. arXiv.org. Retrieved December 5, 2021, from https://arxiv.org/abs/1506.02640.
7. "Wiki." *Ros.org*, http://wiki.ros.org/hector_quadrotor_description.
8. "Wiki." *Ros.org*, http://wiki.ros.org/hector_quadrotor_gazebo.
9. "Wiki." *Ros.org*, http://wiki.ros.org/hector_quadrotor_gazebo_plugins.
10. "Wiki." *Ros.org*, http://wiki.ros.org/roslaunch.
11. Common-3D-Mapping, (2021, February). Retrieved November 6th, 2021 from https://ardupilot.org/copter/docs/common-3d-mapping.html
12. "Wiki." *Ros.org*, http://wiki.ros.org/roslaunch/XML.
13. Mission-Planner-Overview, (2020, March). Retrieved November 8th, 2021 from https://ardupilot.org/planner/docs/mission-planner-overview.html
14. "Wiki." *Ros.org*, http://wiki.ros.org/catkin/conceptual_overview.
15. *Catkin - Ros Wiki*. http://wiki.ros.org/catkin.
16. "# ROS with Mavros Installation Guide." *ROS with MAVROS Installation Guide | PX4 User Guide*, https://docs.px4.io/master/en/ros/mavros_installation.html.
17. Linear Regression in Machine Learning, (March 30, 2020). Retrieved November 11th, 2021 from https://www.justintodata.com/linear-regression-machine-learning-python-tutorial/
18. 19. Sigmoid Function, (April 17, 2020). Retrieved November 13th, 2021 from https://medium.com/@ionutlang31/how-to-play-around-with-sigmoid-function-to-increase-its-y-max-and-shift-to-the-right-for-positive-ede40daf1fa2
19. Cpswarm. "Cpswarm/complex_behaviors: Complex Behaviors Library." *GitHub*, https://github.com/cpswarm/complex_behaviors.
20. "Build Software Better, Together." *GitHub*, https://github.com/topics/swarm-library.
21. "Area_division." *ROS Index*, https://index.ros.org/p/area_division/github-cpswarm-swarm_functions/#noetic.
22. "Target_monitor." *ROS Index*, https://index.ros.org/p/target_monitor/github-cpswarm-swarm_functions/#noetic.

23. PX4 User Guide, (July 17, 2021). Retrieved November 23rd, 2021 from https://docs.px4.io/v1.12/en/
24. By: IBM Cloud Education. “What Are Neural Networks?” *IBM*, https://www.ibm.com/cloud/learn/neural-networks.
25. Building PX4 Software, (July 17, 2021). Retrieved November 25th, 2021 from https://docs.px4.io/v1.12/en/dev_setup/building_px4.html
26. *Multi-Agentpathfindingfor Unmannedaerialvehicles*. https://upcommons.upc.edu/bitstream/handle/2117/176279/143602.pdf?sequence=1&isAllowed=y.
27. https://www.aaai.org/ocs/index.php/SOCS/SOCS19/paper/viewFile/18341/17457#:~:text=The%20Multi%2DAgent%20Pathfinding%20(MAPF,automated%20warehouses%20and%20autonomous%20vehicles.
28. Brownlee, Jason. “A Gentle Introduction to Object Recognition with Deep Learning.” *Machine Learning Mastery*, 26 Jan. 2021, https://machinelearningmastery.com/object-recognition-with-deep-learning/.
29. Sharon, Guni, et al. “Conflict-Based Search for Optimal Multi-Agent Pathfinding.” *Artificial Intelligence*, Elsevier, 1 Dec. 2014, https://www.sciencedirect.com/science/article/pii/S0004370214001386.
30. Solawetz, Jacob. “Breaking down Yolov4.” *Roboflow Blog*, Roboflow Blog, 4 Mar. 2021, https://blog.roboflow.com/a-thorough-breakdown-of-yolov4/.
31. Schwartz, Dan, et al. “Why Drones Are the Future of Outdoor Search and Rescue.” *Outside Online*, 10 Nov. 2021, https://www.outsideonline.com/outdoor-adventure/exploration-survival/drones-search-rescue/.
32. Wilselby. “Simulating an Ouster OS-1 LIDAR Sensor in Ros Gazebo and Rviz.” *Wil Selby*, 21 June 2020, https://www.wilselby.com/2019/05/simulating-an-ouster-os-1-lidar-sensor-in-ros-gazebo-and-rviz/.
33. “Wiki.” *Ros.org*, http://wiki.ros.org/catkin/workspaces.
34. “Wiki.” *Ros.org*, http://wiki.ros.org/Names.
35. Name, Your. “Robotis e.” *Manual*, https://emanual.robotis.com/docs/en/platform/turtlebot3/overview/.
36. “Wiki.” *Ros.org*, http://wiki.ros.org/turtlebot3.
37. “Wiki.” *Ros.org*, http://wiki.ros.org/Nodes.
38. “Wiki.” *Ros.org*, http://wiki.ros.org/Messages.
39. “Wiki.” *Ros.org*, http://wiki.ros.org/Topics.
40. Siemiatkowska, Barbara, and Wojciech Stecz. “A Framework for Planning and Execution of Drone Swarm Missions in a Hostile Environment.” *Sensors (Basel, Switzerland)*, MDPI, 17 June 2021, https://www.ncbi.nlm.nih.gov/pmc/articles/PMC8234058/.
41. Schmuck, P., Ziegler, T., Karrer, M., Perraudin, J., & Chli, M. (2021). Covins: Visual-inertial slam for centralized collaboration. 2021 IEEE International

Symposium on Mixed and Augmented Reality Adjunct (ISMAR-Adjunct). https://doi.org/10.1109/ismar-adjunct54149.2021.00043
42. “Wiki.” *Ros.org*, http://wiki.ros.org/hector_quadrotor/Tutorials/Quadrotor%20indoor%20SLAM%20demo.
43. Osrf. “Mac.” *Gazebo*, http://gazebosim.org/tutorials?tut=install_on_mac&cat=install.
44. Osrf. “Building a World.” *Gazebo*, http://gazebosim.org/tutorials?tut=build_world.
-
45. Osrf.“Gazebo Components.”*Gazebo*,http://gazebosim.org/tutorials?tut=components
46. - Osrf. “SDFORMAT Specification.” *SDFormat*, http://sdformat.org/spec.
47. *Introduction to A**, http://theory.stanford.edu/~amitp/GameProgramming/AStarComparison.html.
48. Tsang, Sik-Ho. “Review: Yolov1 - You Only Look Once (Object Detection).” *Medium*, Towards Data Science, 20 Mar. 2019, https://towardsdatascience.com/yolov1-you-only-look-once-object-detection-e1f3ffec8a89.
49. Tsang, Sik-Ho. “Review: Retinanet - Focal Loss (Object Detection).” *Medium*, Towards Data Science, 26 Mar. 2019, https://towardsdatascience.com/review-retinanet-focal-loss-object-detection-38fba6afabe4.
50. “ArcGIS API for Python.” *How RetinaNet Works? | ArcGIS Developer*, https://developers.arcgis.com/python/guide/how-retinanet-works/.
51. “Papers with Code - Retinanet Explained.” *Explained | Papers With Code*, https://paperswithcode.com/method/retinanet.
52. *You Only Look Once: Unified, Real-Time Object Detection*. https://www.cv-foundation.org/openaccess/content_cvpr_2016/papers/Redmon_You_Only_Look_CVPR_2016_paper.pdf.
53. Gupta, Mehul. “All about Yolo Object Detection and Its 3 Versions (Paper Summary and Codes!!).” *Medium*, Data Science in Your Pocket, 20 Apr. 2020, https://medium.com/data-science-in-your-pocket/all-about-yolo-object-detection-and-its-3-versions-paper-summary-and-codes-2742d24f56e.
54. *Yolov3: An Incremental Improvement - Pjreddie.com*. https://pjreddie.com/media/files/papers/YOLOv3.pdf.
55. Ampadu25.00, HAHyacinth, et al. “Yolov3 And Yolov4 in Object Detection.” *AI Pool*, https://ai-pool.com/a/s/yolov3-and-yolov4-in-object-detection.
56. Supeshala, Chamidu. “Yolo v4 or Yolo V5 or PP-Yolo?” *Medium*, Towards Data Science, 23 Aug. 2020, https://towardsdatascience.com/yolo-v4-or-yolo-v5-or-pp-yolo-dad8e40f7109.

57. “Tips for Best Training Results .” *Tips for Best Training Results - YOLOv5 Documentation*, https://docs.ultralytics.com/tutorials/training-tips-best-results/.
58. S. A. Dart, "Overview of Software Development Environments," [Online]. Available: https://www.ics.uci.edu/~andre/ics228s2006/dartellisonfeilerhabermann.pdf.
59. A. Bulajic, "An Effective Development Environment Setup for System and Application Software," Issues in Informing Science and Information Technology, vol. 10, 2013.
60. J. Morrow, "How to Choose the Right Operating System," 5 June 2017. [Online]. Available: https://www.hdt-project.org/how-choose-operating-system/.
61. LYNX SOFTWARE TECHNOLOGIES, "Real Time Operating Systems (RTOS) and Secure Hypervisors for Unmanned Systems," Unmanned Systems Technology, [Online]. Available: https://www.unmannedsystemstechnology.com/company/lynx-software-technologies/. [Accessed 23 October 2021
62. Flytbase, "Operating System for Drones," Flytbase, [Online]. Available: https://flytbase.com/flytos/. [Accessed 23 October 2021].
63. dronelife, "Auterion, Open Source Operating System for Drones, Announces New MAVSDK: Software Development Kit for Drone Communications," [Online]. Available: https://dronelife.com/2019/09/05/auterion-open-source-operating-system-for-drones-annou nces-new-release/. [Accessed 23 October 2021].
64. Z. Z. a. G. Xiao, "Evolution analysis of a UAV real-time operating system from a network perspective," Chinese Journal of Aeronautics, vol. 32, no. 1, pp. 176-185, January 2019.
65. ros.org, "About ROS," ros.org, [Online]. Available: https://www.ros.org/about-ros/. [Accessed 23 October 2021].
66. GitHub.com, "The tools you need to build what you want," GitHub.com, [Online]. Available: https://github.com/features. [Accessed 24 October 2021
67. K. Taylor, "Open-Source Virtualization Software: What are the Top Most Open Source Virtualization Software," [Online]. Available: https://www.hitechnectar.com/blogs/open-source-virtualization-software/. [Accessed 24 October 2021].
68. *Introduction*. Introduction · MAVLink Developer Guide. (n.d.). Retrieved December 6, 2021, from https://mavlink.io/en/.
69. Drone Obstacle Avoidance System, (March 19, 2015) retrieved December 1st from https://www.unmannedsystemstechnology.com/2015/03/panoptes-systems-introduces-ebumper4-drone-obstacle-avoidance-system/
70. Supervised vs Unsupervised Learning (March 22, 2018) https://towardsdatascience.com/supervised-vs-unsupervised-learning-14f68e32ea8d

71. By: IBM Cloud Education. “What Are Convolutional Neural Networks?” *IBM*, https://www.ibm.com/cloud/learn/convolutional-neural-networks.
72. Saha, Sumit. “A Comprehensive Guide to Convolutional Neural Networks - the eli5 Way.” *Medium*, Towards Data Science, 17 Dec. 2018, https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53.
73. “Convolutional Neural Networks.” *Weights & Biases – Developer Tools for ML*, https://wandb.ai/site/tutorial/convolutional-neural-networks?gclid=CjwKCAiAhreNBhAYEiwAFGGKPCNweRBU9jkLZno_7BBiX2KkTaH6YPuMvguUT5VU3RxhKxirxCl6NhoCr2YQAvD_BwE.
74. Visual Odometry (February 9th, 2016) http://www.cs.toronto.edu/~urtasun/courses/CSC2541/03_odometry.pdf
75. Sahoo, S. (2021, August 16). *Residual blocks - building blocks of Resnet*. Medium. Retrieved December 6, 2021, from https://towardsdatascience.com/residual-blocks-building-blocks-of-resnet-fd90ca15d6ec.
76. *Gapter*. robots.ros.org. (n.d.). Retrieved December 6, 2021, from https://robots.ros.org/gapter/.
77. *Collaborative 6DOF relative pose ... - margaritachli.com*. (n.d.). Retrieved December 6, 2021, from https://margaritachli.com/papers/ICRA2018_Karrer2_paper.pdf.
78. Zou, D., Tan, P., & Yu, W. (2019). Collaborative visual slam for multiple agents:A brief survey. *Virtual Reality & Intelligent Hardware*, *1*(5), 461–482. https://doi.org/10.1016/j.vrih.2019.09.002
79. Ctu-Mrs. (n.d.). *CTU-Mrs/MRS_UAV_SYSTEM: The entry point to the MRS UAV system.* GitHub. Retrieved December 6, 2021, from https://github.com/ctu-mrs/mrs_uav_system.
80. Express, C. (n.d.). Simulation. Simulation · Clover. Retrieved December 6, 2021, from https://clover.coex.tech/en/simulation.html.
81. Loosli, E. (2021, November 26). Drone photogrammetry vs. Lidar: What sensor to choose for a given application. Wingtra. Retrieved December 6, 2021, from https://wingtra.com/drone-photogrammetry-vs-lidar/.