Simulated Swarm Gold Team Design Document

Group 24

Fall 2021

Team Members:

Andrew Borg (AI & Pathfinding, CNN & Computer Vision, Machine Learning)

Bobby Pappas (AI & Pathfinding, CNN & Computer Vision, Machine Learning)

James Murphy (Project Lead, AI & Pathfinding, CNN & Computer Vision, Machine Learning)

Matthew Hubbs (AI & Pathfinding, Gazebo & Simulation, Machine Learning)

Sebastian Almeida (CNN/Computer Vision, Gazebo & Simulation, Machine Learning)

Sponsor:

Lockheed Martin Missiles and Fire Control Applied Research

Joseph Rivera

Executive Summary

This project is the second Senior Design project with the goal of making a swarm of 5 flying drones with artificial intelligence and advanced swarming algorithms map an environment and locate a target. The previous Senior Design team started with this goal, but it was such a herculean task that they had to adjust their scope in to fit their capacities and time frame. This semester three teams sponsored by Lockheed Martin have been given this same task and thrust into competition with each other. This team, the Gold Team has the same task, but we are armed with what the previous team left us. Between the guidance of the teaching faculty of UCF, the Industry-level advice of Lockheed Martin, and the head start that the previous team has provided, we hope to complete this year what the previous team started last year.

From a high-level perspective, this project will be packaged into a single code repository. By running a series of commands in an Ubuntu 20.04 machine with the appropriate simulation and robotics software installed, our simulation will begin. The simulation contains a couple of city blocks, complete with residential houses, storefronts, a telecommunications tower, a gas station, and many vehicles like trucks, cars, and ambulances lining the neighborhood streets. Our swarm of drones will search the environment until it either finds its target or searches the entire area without finding the target, and the simulation will end. The drone swarm will be completely autonomous and once started will run to completion without human input.

Several different algorithms and methodologies have been incorporated into the project. The drone swarm will be doing several things simultaneously as it works towards completing its goal. It divides the environment into subspaces using an octree then categorizes those subspace cubes into searched, unknown, and frontier cells, using the theta* algorithm to pathfind efficiently through the search volume. The swarm must communicate between individual drones in order to pathfind efficiently as a team.

It also must search for the target as it pathfinds through the environment. To do this, the swarm will be using an algorithm named YOLOv5 which is an artificial intelligence object detection algorithm.

The greatest challenges of this project is the scope of the project itself, and the challenges of getting a swarm of artificial intelligence agents to work together efficiently. We expect this to be the most uncharted part of this project.

Identification of the Project and its Significance

The Simulated Swarm is a project aiming to find ways for several robotic agents to work together to complete a task better than several independent robotic agents who could complete the same task on their own. Specifically, we will be using a simulated swarm of flying and/or land based robots in a 3D urban environment to search and destroy a hostile simulated robot. The following are statements from each team member describing what motivated them to choose this project and what they hope to get out of this project.

Personal Motivations

Matthew:

My interest in the military and its equipment goes all the way back to when I played with toy soldiers, tanks, and planes as a kid. I never stopped being a kid in the sense that I found these things to be cool, and as I grew up and learned more about what the military is and why our country has one, I grew to respect the larger than life bravery and heroics required to actually be in the military. My first experience with Lockheed Martin was at a high school programming competition that the company hosted on a weekend, and ever since I have always thought that working for such a company would be a way for me to support the military that I have admired as I have grown up, especially considering that I am not fit to enlist in any military branch due to inherited health considerations.

AI, drones, and robots have a very polarized public opinion. For me personally, I fall squarely into the category of overwhelming support and fascination towards all three. Having such an interest in the topic we are working on is sure to provide continued motivation to keep moving forward, even if the road ahead isn't all sunshine and rainbows.

In a more professional sense, my main goals of this project are to first and foremost pass the class and graduate, and to gain technical experience and networking connections with Lockheed Martin. I think this project is a perfect bow to complement my undergraduate career and move on to my professional career in the industry.

From a technical perspective, I have taken intro to AI and am currently taking Robot vision, both of which will likely help in the project moving forward. I have previously worked in linux in less formal pet projects and tinkering on my raspberry pi, and I have made games in both Unreal Engine 4 and Unity, and this experience will likely help with working in gazebo throughout the project.

Andrew:

What first caught my attention on this project was the robotics aspects. I don't have any background with the military, but throughout highschool I had greatly enjoyed building and competing over four years with robots through the FIRST Robotics Competition, and so this project caught my eye with the robots. My experience with robots in the past was primarily remote controlled, with some systems to aid the person driving it, so I expect it will be very interesting to make a transition to fully autonomous control where artificial intelligence systems are essential. After graduating I would love to work in robotics, computer vision, and/or artificial intelligence, so one of my goals in this project is to build up technical experience in these areas, and another is to form network connections within Lockheed Martin where experts on these topics are certainly present in many projects.

My prior history with robotics gave me some experience with using sensors such as encoders to inform automatic control, but I haven't worked with the reduced certainty of flight or combining multiple sensors together especially with statistical filters to use them all to determine a combined more accurate reading. ROS and Gazebo are both popular open source tools in work with robotics that this project will involve learning and practicing with. These represent a great space to expand my knowledge if I plan to explore robotics in the long term.

James:

My initial motivation and interest was to learn a ton during my senior design project. This project had a lot of elements such as Machine Learning, Robotics Simulation, Neural Networking, and Computer Vision; all of which I haven't done before and wanted to push myself to learn and master. My goal as a student here at UCF was to get my experience with as many different forms of Computer Science and not specialize in exactly one branch of the Major. A “Jack of All Trades, Master of None” kind of thing.

Through my internship I have gained many skills in the area of web development along with blockchain. Robotics and Data Science is something I was always interested in and this project, along with the proposal itself, had all I wanted and was instantly my top pick. I am very happy to be a part of this team and am excited to be able to sit back and say “I did this” at the end of the Spring semester. The fact that it will be a competition adds a unique level of motivation beyond the grade and pushes forward innovation just that much more and will be interesting to watch all teams progress as the time goes on.

Sebastian:

I have a lot of interest and some prior experience with AI and machine learning so I wanted to work on a Senior Design project that involved some sort of AI. What set this project apart for me was the interaction of AI with robotics. This cross-functional field is already in use with companies like Tesla and Uber having self-driving vehicles. I believe there are many potential uses of AI with robotics and that by working on this project I will gain the capability to do this work when I graduate.

The project premise is that given a randomly spawned target and drones within a simulation, the drones must find the target as quickly as possible. The simulation will be performed in Gazebo. This is an iteration on a project from last year that used ground vehicles to accomplish this goal so a large change will be the use of drones. In

order to accomplish this goal a multitude of machine learning algorithms will have to be used. The algorithm most unique to this iteration will be the swarm AI algo. Pathfinding is another portion of the project which in the previous iteration used A* and SLAM. Target recognition is the final piece which used YOLO V3 Neural Network last iteration. An additional UI component to show a pseudo-gps is also a stretch goal for this project.

Bobby:

What initially caught my eye was the implementation of AI and machine learning algorithms. I have some previous experience working with AI from CAP4630 which kickstarted my desire to dive deeper into the subject. I immediately knew that this would be my first choice because I would be getting a hefty amount of experience with such an interesting topic while also getting to put it into practice professionally. That paired with having the opportunity to get my foot in the door at a big company like Lockheed Martin made the decision a no brainer.

Broader Impacts

While the initial project pitch from Lockheed Martin mentions searching for and destroying a hostile drone, it is important to consider the military applications this could be adapted for, but the possible uses for this technology expand beyond the military. Weaponized autonomous drones would be incredibly capable of incredibly efficient harm, but could run into risks with falsely identifying between friend and foe, or between military and civilian. While successful deployment of drone swarms like this would likely have massive impacts on a tactical and strategic scale, it is important not to look past the ethical and humanitarian issues with allowing an autonomous weapons system to deploy lethal force. Another much safer military approach would be for

intelligence purposes, which don't take such risks, and as swarms are not easily eliminated, makes for exceptionally low risk surveillance.

The use cases for the drone swarm search we are developing also include a great deal of purely civilian impacts that we can examine. Using a swarm of drones to find something in an urban environment has implications for finding missing persons, searching for criminal suspects that could be an active danger to the public, or finding and rescuing endangered survivors in the wake of a natural disaster.

In a more broad sense, this project could find ways for multiple robotic agents to work together in a general sense even outside of the task of searching for something in a 3D environment. In general, robots are expendable, while human life is inherently not, and increased competency of robots directly translates to transferring more tasks of greater risk away from humans, so any advancement in these areas of robotics has the potential to lead to solutions which can save lives.

The detractors of artificial intelligence and robotics often argue that the economic impacts of developing this technology will be negative. It is the classic "robots are taking our jobs" argument. While this is a valid concern, for every door that is closed, a window is opened. It is possible that improved inter-robot teamwork will eliminate some human jobs, it is also possible that it will open up new opportunities and new jobs that are not even conceivable at this time. Ultimately, this technology is likely to have an impact, mostly among those who work in lower technically-skilled positions to the benefit of those who work in higher technically-skilled positions, this is nothing new. Since the dawn of humanity, people have learned to create and use tools, and this project is no different. After all, deciding not to develop this technology would be putting most of the people in this class out of a job, so it doesn't make sense to not develop this project because doing so could put someone else out of a job.

While the team believes that developing this technology is likely to be beneficial to mankind as a whole, any technology, including weapons system technology, can be used for both good and evil. While some bad uses of a potentially weaponized swarm of drones are obvious, some are less obvious. One example would be paparazzi using a swarm of drones able to recognize famous people in order to gather information about every part of a celebrity's life.

Legal, Ethical, and Privacy Issues

As is, we have complete control over this project as our own personal IP, not to be used by Lockheed Martin. With that we have to adhere to any type of other legal or ethical issues that might be coming our way. With the subject of finding, targeting, and destroying something, ethics can play a large role. Currently anything that we are doing, will be done completely in a simulation, not to be done in reality. This makes the ethics argument a lot easier on one as there is no implementation in the things we are simulating. Just showing that we have the ability and the knowledge to accomplish something of this threshold.

A specific ethical concern regarding swarm AI and especially for a defense contractor as Lockheed Martin is the potential for the use of our program in drones on battlefields. Since drones are anonymous it is important to ensure that they behave properly as even when not equipped with weapons drones can cause serious harm to people. Especially with so many drones behaving independently it is very important for the algorithms controlling the behavior of the drones to be tested for safety. Should a weapon be placed on the drone it is even more important that the drone software be designed to the highest of safety specifications.

This project does not have any privacy liabilities because no user data is stored other than information about the identity of the simulated target. Since the target is simulated and a model of Poe Dameron's adorable robotic sidekick BB-8, the target is not a real person, and as such the project does not need to consider any law or regulations that protect user data like HIPPA or COPPA.

ITAR is the International Traffic in Arms Regulations which restricts the export of defense and military technologies. This project does deal with weaponized autonomous drones, but they are simulated and therefore not real. This project is as much "military technology" as a simulation game like Arma or DCS World is "military technology."

Even if this project did come under the scope of ITAR, which it does not, this project is being shared over github with only ourselves, Lockheed Martin, and Professor Heinrich and Professor Leinicker, all of which are US citizens.

In the real world, the FAA requires all drones to be flown below 400 feet, within sight of its operator, and under a speed of 100 miles per hour. Luckily for us, the FAA does not have jurisdiction in our simulated environment.

There are many privacy laws in the real world related to flying drones over other's property and homes, but they do not apply to our project because the property our simulated drones are flying over does not exist in the real world.

At the current stage, the drones are simulated, the target is simulated, the environment is simulated, and since some people will be running the project in a virtual machine, the computer running the simulation might be simulated too. The project is so far removed from the real, physical world that there are no current legal, privacy or ethical concerns. However, as the project moves forward, out of the simulated environment of a simulated computer it resides in today and others build on what we do in our time in senior design, there are things to consider. Previously in this section, we have addressed how many specific laws do not apply in a world that is not real, but we also acknowledge that this technology may be brought to the real world, and we urge anyone reading this in the future who is considering doing so to think about how the things we discussed that do not apply in a simulation may apply in whatever situation you are applying our work. We cannot see into the future, but if you are reading this right now, you can see your present and have a much better position from which to make ethical and legal considerations.

Project Requirements

After the initial kickoff meeting with Joseph Revera ,the sponsor from Lockheed Martin, we came up with the following requirements.

Must Haves

In order for the project to be considered “working,” the simulation shall use Gazebo and ROS and use the pre-built urban environment produced by the previous senior design team. The target detection system must draw a bounding box over the target with an Intersection over Union (IoU) of greater than 0.5, and there must also be a way to compute a confidence score for identifying the target which must be above 0.7. There must be at least 5 drones in the swarm, and they must autonomously search the simulated environment and find the target within 3 minutes. The individual drones in the swarm must collaborate and behave as a swarm.

Stretch Goals

As a stretch goal, the swarm will create a tactical display with a pseudo GPS containing the coordinates of the target on a grid. This has obvious utility for all the suggested uses of a swarm suggested in the “Broader Impacts” section.

Initial Ideas

Matthew:

Ground vehicles could be used to carry things that would be useful to a swarm but are too heavy to put on a flying drone. If such a swarm is intended to work alongside humans, a ground vehicle could serve double duty as a pack mule for the human members of the squad with water and ammunition at the same time as serving as a computational pack mule for the swarm.

Distributed vs centralized decision making will be a key decision to choose between. I can see a master/slave relationship for the drones working just as well as an individualistic approach.

For the “destroy” part of “search and destroy'' we could use direct fire weapons like lasers and firearms, or indirect fire from a mortar or missile, or we could even use a more exotic approach like a small emp pulse to disable the enemy drone, or a net to ensnare it.

If two flying drones have the distance to the target, each other, and a third ground vehicle, they can triangulate the target and put the target on a coordinate system along with the three drones. This has obvious utility when trying to destroy the target.

Perhaps we don't necessarily need to link the sensory inputs directly to the control outputs of each member of the swarm. Maybe we can hard code specific actions and build a bank of actions each member of the swarm can perform, then use AI to decide what actions to perform. Just because AI and neural networks are powerful and can do many things does not always mean we should never hard code any behavior.

Andrew:

My initial thoughts were to take a decentralized approach to dividing the swarm of drones for search, in a manner somewhat related to how a swarm of insects divide themselves and assign tasks to themselves. Nature has evolved many good ideas, and I think it could be at least worth looking at them. Naturally, since the primary benefit of this is scalability, it will likely apply less strongly to our more constrained approximately 5 drones, leading to unnecessary sacrifices for insufficient benefit, since we may be able to allow the drones to communicate over any relevant distances.

The requirements lean heavily towards needing a simultaneous localization and mapping (SLAM) algorithm, which should be shareable between the drones. This shareability of maps creates an excellent opportunity to achieve significant gains in mapping and search speeds by sending each drone to search a different unexplored region before the space is fully mapped, communicating the target’s location relative to the shared map or efficiently navigating to it around even buildings a particular drone hasn’t directly observed yet once found.

I also had the thought that it could be useful to have multiple states for each drone, such as combined mapping and passive searching, searching (collaborative sweep of the area to find even a target that moves itself to actively attempt to hide), and

following and communicating target location to all other drones and any tactical display. If communication range is restricted, a regrouping mode in order to share maps would prove beneficial as well.

James:

One initial Idea I had is a Master/Slave implementation where four drones spread out and are searching at a faster pace and covering more ground in less time, while the Master drone is a bit more detail oriented and goes slower and checks for more edge case specific spots that the Slave drones might miss in the time of a fast search with a chance for fast find.

Another initial idea was to keep the target as BB-8 for our project so that we can use the previous data set as a head-start on our Train/Test Model and we can continue to implement on top of their data set so the Model has more to work with for higher confidence on object detection.

Also, an idea I had was to reach out to my professor for another class who knows about a lot of hard-to-find Computer Science / Cyber Security websites to see if he had any resources for us to use. This has resulted in us getting in contact with the president of the newly founded Drone Club here at UCF. This will allow us to learn more about drones for our implementation in this simulated environment

Sebastian:

Quality of visual data that a drone receives is correlated to its height. The higher the drone is, the broader the area is in which the drone can search, but in turn the lower the quality. The lower the altitude of the drone is, the better the quality of the images, but the narrower the swath of space it can analyze at any one time. This seems to suggest that a probability based model may be best for doing searches. Since the higher the drone is the faster it can scan but the higher the chance it could miss the target or false positive. So possibly an approach with drones at different heights working together.

One possible approach to drone communication is to have drones in higher altitudes share information with drones at lower altitudes whenever the probability that the target is present in an area exceeds a certain threshold. Since a gps/map is a goal for this project, this could be mapped out throughout the search. For the first round, the highest drone could mark what it thinks is in different areas of a grid representing the search area. If a probabilistic model is used, then this map could also include % chance it believes the target is in that area.

Drones may have wide angle cameras, thus it is important to look at the image as a rectangle since this allows for segmentation of the search area into equal parts sections. Another possible approach would be to have UGVs along with the drones. This is because UGVs are capable of getting higher fidelity imaging whereas the drones have lower fidelity but a broader range of data from their cameras.

Bobby:

The first idea that came to my mind was to have the swarm act in unison. While in certain scenarios it would be advantageous to spread out as individual drones, other scenarios might find an advantage from multiple drones flying together in a pack. Flying in a pack would create a combined larger field of view to take in more information sequentially. It would allow for more precise searching/mapping in the event where a drone had missed its marker through a blind sight in the field of view.

While having the 5 drones flying in the air mapping the data, I also envision having a ground vehicle having a similar task. It would be a sort of checks and balance system to allow for multiple perspectives looking at similar data and seeing how it compares. This would add to our models total accuracy in order to perfect the mapping aspect.

I'd say the most important part of these ideas for me would be to allow for different perspectives of the same overall data set. Combining those perspectives together is what I believe works best when the goal is to be as accurate as possible.

Budget and Financing

Joseph Rivera, our sponsor, said last year's project did not receive a budget. After inquiring whether we can request a budget this year, Lockheed Martin has paid UCF $300 for use by the team throughout the project.

At the current moment, we are using Github Student for version control and VMware Workstation Player for virtualization of Ubuntu 20.04 LTS so that Gazebo can run, and both of these are free software, while Gazebo and ROS are both open source projects that are also free for use.

Another free resource the team may utilize throughout the course of the project is Newton, a cloud based computing resource that could be helpful for machine learning training. Use of this service has recently become free for the senior design teams this year.

One thing that does cost money and will be useful to our project is a training course program from *The Construct* which Patrick Bauer, the project lead for the previous senior design group, specifically recommended to us. It costs 40 euros per month, which converts to close to 50 US dollars per month. We intend to use a single membership similarly to how most people use a netflix account, and don’t expect to need more than one or two months of the service. This leaves us with an extra $200 in the budget should we need it later in the project. Currently, we have contacted the two professors purchasing this learning resource, and are currently waiting for the UCF accounting wheels to turn. One thing for a future senior design team to consider if they are in a similar situation is that it has been about 35 days since first contacting the

appropriate people and we are yet to receive word about when we will have access to this resource. It may be wise to ask as soon into the semester as possible if you are able to be reimbursed for expenses and simply use your own money to make purchases, allowing UCF to reimburse you whenever they get around to it. If you choose to do this, be sure to make smart personal finance choices regarding this.

The Plan

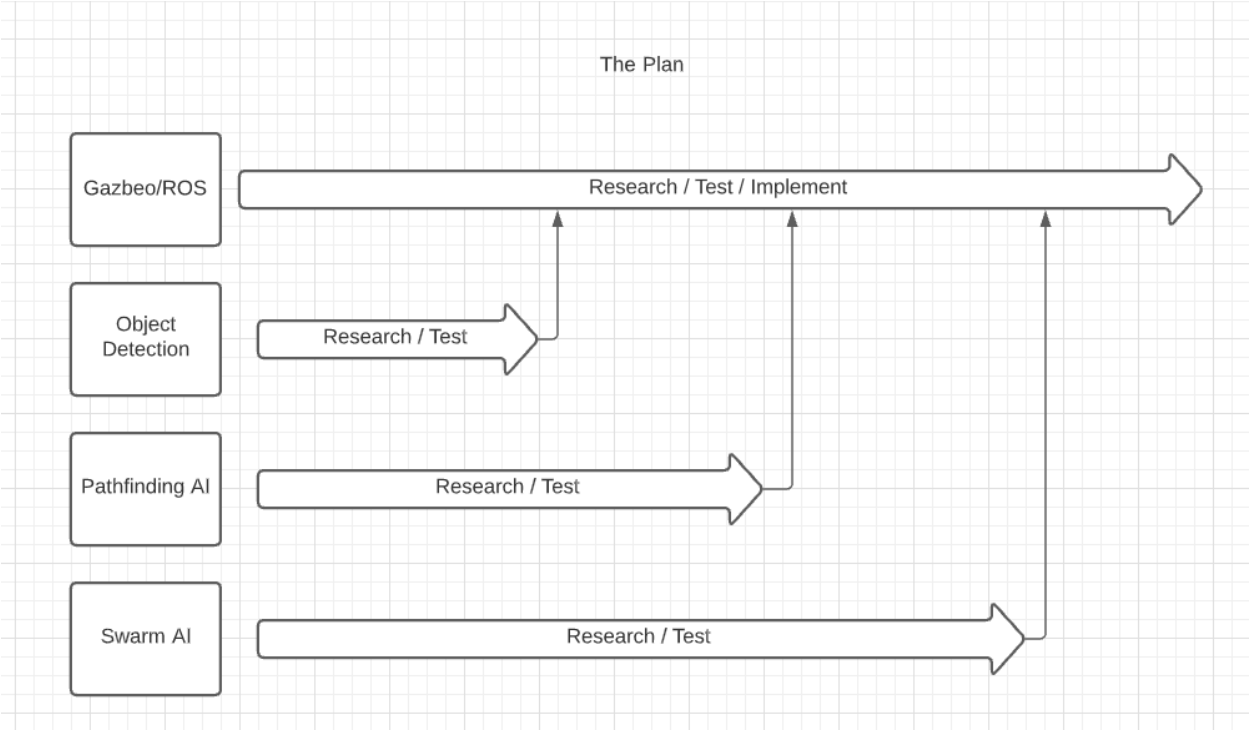


Figure 1: Early Planning

Overall, we want to tackle some very complex topics with efficiency and success. The most general goal is to divide and conquer where we spread out the members of the team to research and test every topic: Gazebo/ROS, Object Detection, Pathfinding AI, and Swarm AI. With the majority of the project being in Gazebo, it is important to have a group of people who will be there starting and being there to help merge the other elements of the project. To summarize the Block Diagram here are the steps:

1. Begin Research/Test for each subject

2. Once the Gazebo Environment is ready, begin gathering images for object detection implementation

3. Following above, begin to implement the pathfinding AI to find a target with a single drone

4. Following above, implement the Swarm AI for multiple drones

Overall, when a subject is ready to be merged with gazebo, it gets implemented iteratively. We will be using GitHub to allow for asynchronous development and merging of progress. This also allows us to have previous commits in case of mishaps and we will have 2 code reviews required for every pull request into the development branch.

When it comes to team communication and workflow, for our project we feel a more deadline based approach is better so we will be opting for a Gantt Chart with weekly meetings to make sure we stay on track. On top of this, we still plan to have some elements of AGILE, more specifically SCRUM. The benefits of knowing what people have done and what successes/failures took place before the meeting even starts allows for more efficient meetings and better awareness of the project overall. With this being said, all SCRUM reports will be in a backlog channel on Discord to allow for logged communication of the goals for the day, what got done as a result, and what problems or concerns came up that will be addressed in the future.

Retrospective Workflow

Agile was a great addition to our workflow as it allowed us to help each other whenever we needed it. While we each had to handle different parts of our project, in the early stages of researching when most of us didn't know much about our specific topic, we were able to aid where we saw fit. I think that this was a great decision and we look forward to using Agile in the future. Keeping each other updated on what was going on and what issues we were running into along with the Gantt chart allowed us to all have a general understanding of where we were at with our respective work.

Agile also allowed us to knock out a lot of our research due to the fact that we were constantly sharing resources we'd run into every day. It allowed us to stay motivated as well due to the fact that we knew what everyone else was doing on that given day. At least personally it kept me motivated knowing that my teammates were putting in a lot of effort to do what needed to be done, which inclined made me more productive and efficient.

Agile also helped us when our pace slowed down due to the other classes requiring more time because we were able to be clear with each other when we needed help. This sometimes goes unspoken in group projects as it is just assumed that everyone will be having an easy time getting their work done, but this isn't always the case. I found in my other group projects that I've taken part in while I was at UCF, that a lot of the time people were not willing to say when they fell behind or needed help which just in turn makes everyone's time working a much worse experience.

Below is the Gantt with more detail of the tasks that will be tackled to ensure success for the final implementation in SD2.

Gantt Chart

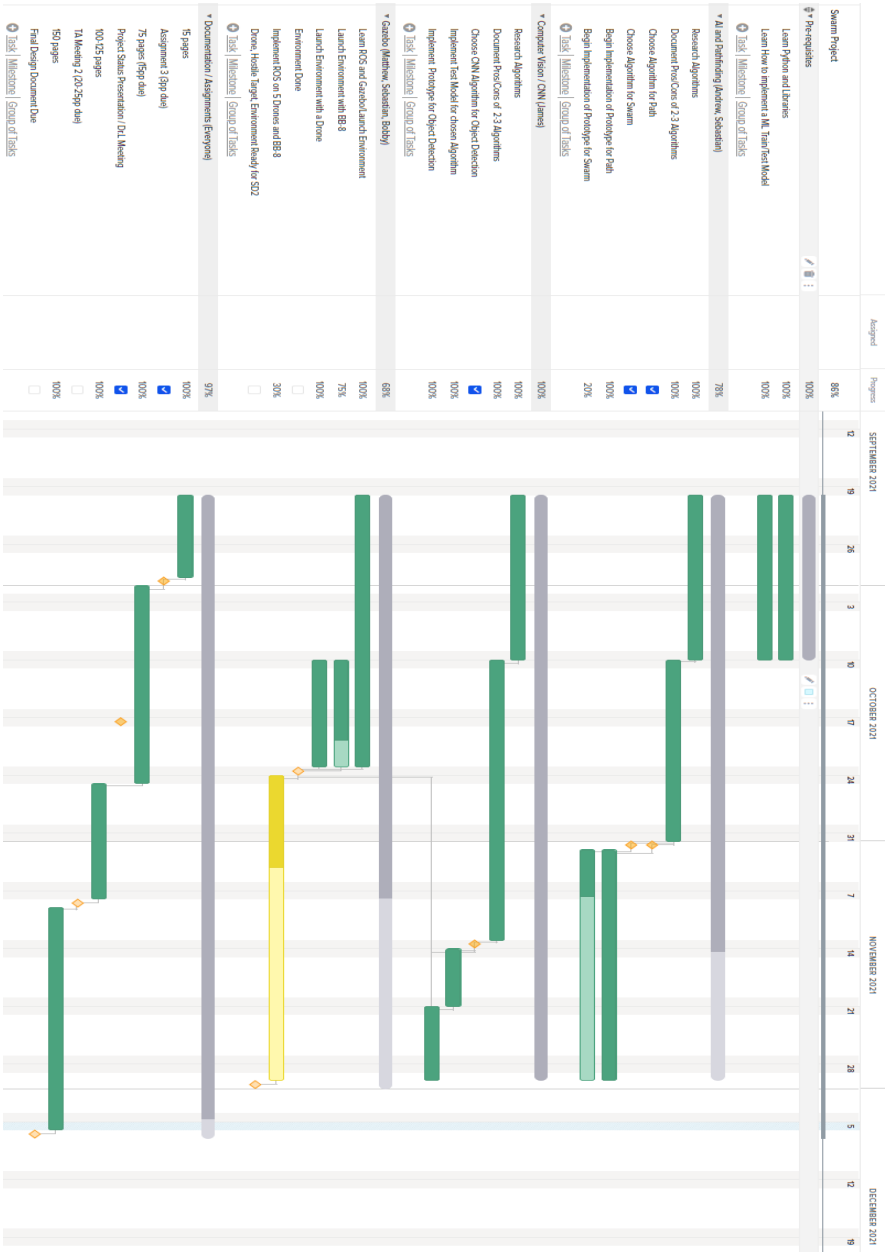


Figure 2: Up to date Gantt Chart

Project Milestones

Above is our Gantt Chart for the Senior Design 1 Fall semester. The roles timeline takes into account the amount of people on roles along with the course assignments/presentations/meetings we will have to do along the way. Green means “Complete”, Yellow means “In Progress”, and Blue means “Not Started”.

Prerequisites:

Due to a majority of the project requiring Python and an understanding of Machine Learning, the first task for everyone was to learn Python if they didn’t know it and then libraries associated with implementing custom Machine learning models along with learning Machine Learning if that was unknown

AI and Pathfinding:

The first goal is going to be to research as many algorithms for pathfinding and AI as possible in the beginning. During our weekly AGILE meetings, we will discuss what was found, the pros and the cons, and inevitably by a milestone have picked both. Then we will begin to implement it for the remainder of the semester.

CNN/Computer Vision:

Similar to the AI and Pathfinding roles, we will be researching the best algorithms possible for video object detection and discussing results at the weekly AGILE meetings. We will aim to have chosen one by a milestone within the semester. Then implementation will begin for the remainder of the semester.

Gazebo Simulation with ROS:

Luckily, thanks to our sponsor, we have access to the previous project's environment and are able to use their resources. This allows us to just have to worry about the implementation of ROS and getting drones in the environment. The goal for this role is to learn early on and run the previous simulation environment as a milestone mid-semester. Then we will begin to set it up to be done by a milestone and ready for Senior Design 2 by the end of this semester.

Documentation/Assignments:

The bottom Group of Tasks is our deadlines for all the course Turn-Ins we will be responsible for such as the Final Project Documentation milestones and Presentations/Meetings.

Robotics Simulators

Operating System: UBUNTU

Environment Visualizer: Gazebo

Robotics Simulator: ArduPilot

Middleware Solution: GzUAV

Ubuntu is necessary to use as the environment as the simulation software Gazebo is only usable in Linux. Gazebo is an open source plugin based simulator. This means that to add new robots and items to the simulation you must download and plug them in to use them.

GzUAV is the middleware solution that is pre existing for drone swarm integration in Gazebo. One benefit of this approach is that each drone can have a separate computer running the algorithms and computations for that drone. This potentially could help our cause immensely as there is no limit to the amount of drones we can put within the simulation as GPU power would be the typical bottleneck.

So far we have been able to open up the tutorial files included with GzUAV, this has allowed us to see the drone models in action and understand how to incorporate them for our uses. Several drone models exist, the ones with cameras will be most useful for our purposes. Another consideration is that since LIDAR is a usable and useful tool for

our purposes, we may have to adjust the drone files to also have LIDAR sensors on them.

One area of trouble has been opening the .world file from the previous group. But upon further investigation we learned we needed to have the assets that the file was dependent on downloaded and placed in the Gazebo Models path folder. Versioning has also been an issue with the .world file since it is written in SDF which has many versions and each one corresponds to a different version of Gazebo. SDF is a markup language similar to HTML and is not too difficult to understand. Although luckily Gazebo will just ignore pieces of the code it does not understand and versioning may not present any roadblocks. Further investigation found that the previous group's world file is a pre-existing Gazebo environment that had walls placed to bound it into certain dimensions. Another

Currently we have opened up the drones with the supporting swarm client. Separately we were able to open the .world file from the previous group in Gazebo. The next step will be to open up the world file with the GzUAV client as well. From there we also need to load in the target, in our case a BB8 drone. From there we will have to load up our own python files responsible for pathing, computer vision, and swarm algorithms.

Another consideration will be adding the LIDAR sensor to the currently existing drone models. This will require some analysis of the benefits they would provide since their use would most likely increase processing power needed. The main benefit that a LIDAR sensor would add is enhanced depth perception for the UAVs.

Figure 3: Sample LIDAR sensor results

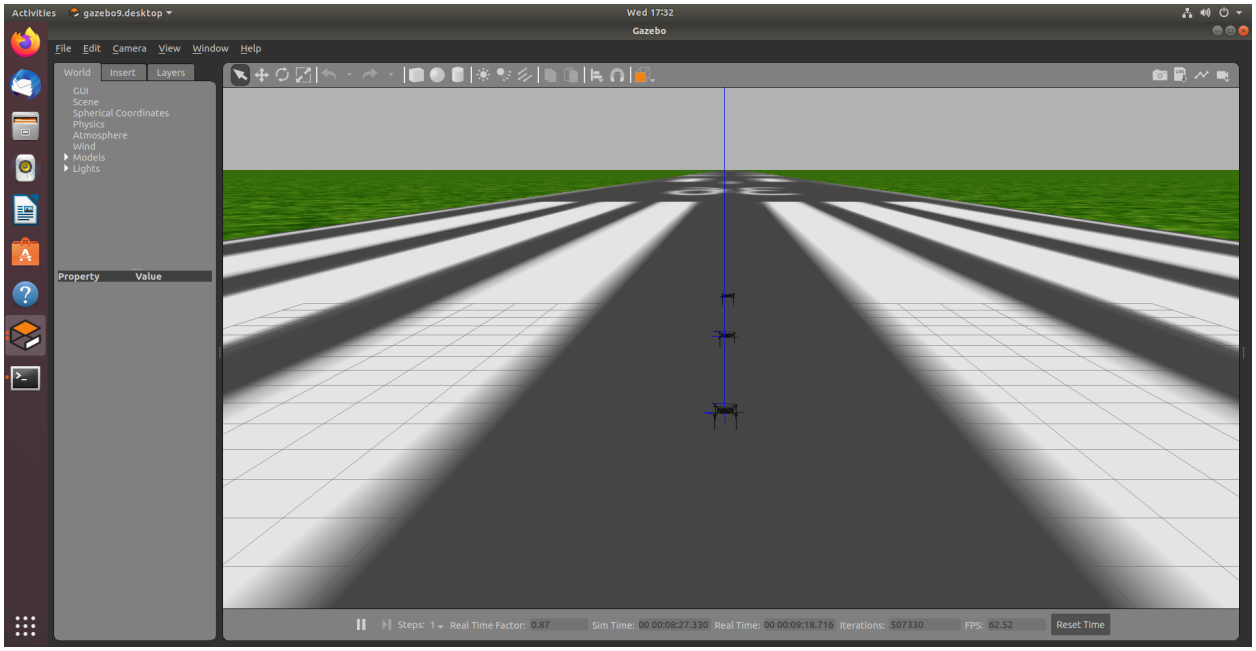


Figure 4: Drones open in Gazebo environment

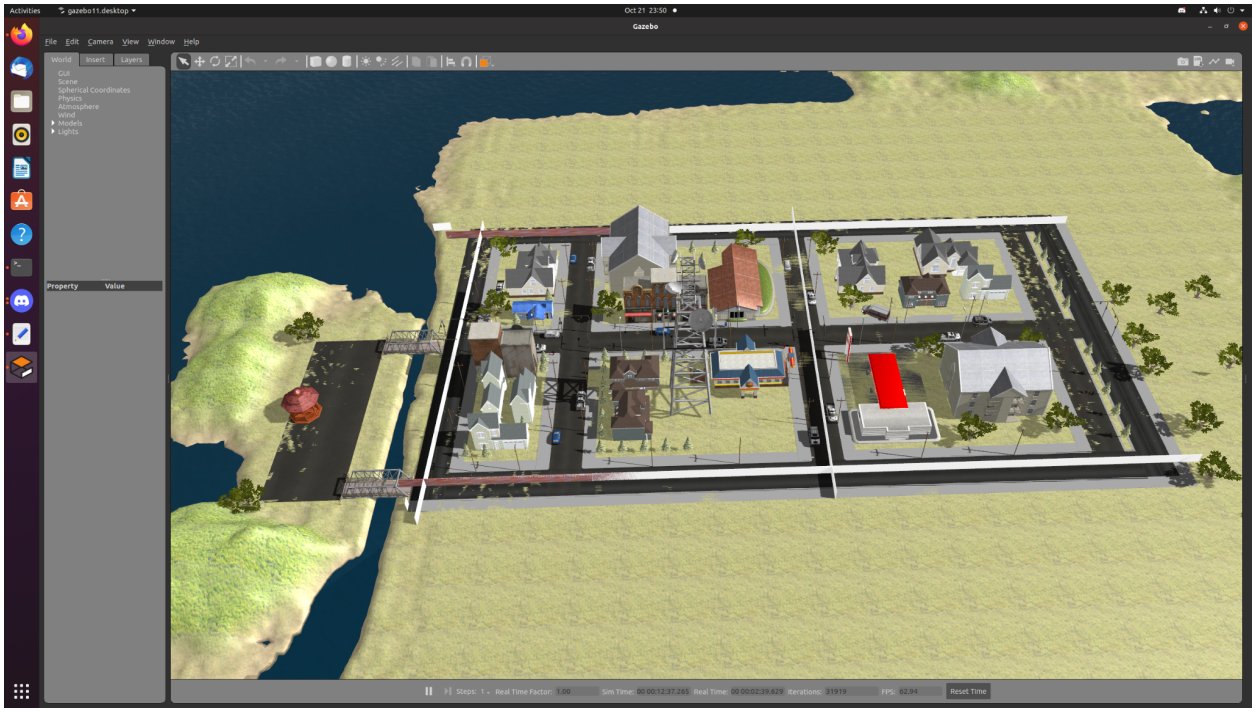


Figure 5: Last teams .world file open in Gazebo environment

Middleware Setup (Alternative/Unused Solution)

I. Environment Setup

1. UBUNTU 18.04.6 (Bionic Beaver) Setup

- Download ISO File
- Download VMWare
- Open ISO File with VMWare
- Open This Doc From Within VM firefox browser

3. GAZEBO 9 Setup

- Download Gazebo 9
- Little Gazebo 9 Might have to be downloaded separately
- Ensure you are familiar with models path

3. Python Setup

- PIP INSTALL MONOTONIC
- PIP INSTALL DRONE KIT

4. Gzuav Setup

- Download GZUAV
- Export SVGA_VGPU10=0(Before every run)
- Follow setup guide

5. World File

- Download World file
- Place file into the folder where you will be calling the world file from

Python To Interact Algorithms With Drone Flight Stacks

The current middleware implementation has python 2 files running, but for our purposes we will use python 3 as there are more advanced algorithms available on this version of python.

Drone Kit is a good potential package which we did not use, to run computationally expensive tasks like computer vision algorithms to run between the drone and the GPU. Drone kit was built to run with the ArduPilot UAV platform. ArduPilot is built for fully autonomous flight of UAVs. it is possible to use ArduPilot to control the movement of the multirotor drones. Both of these softwares are open source and thus well documented and supported by the community so any problems we face will hopefully have been faced by someone else before. Another added benefit of using ArduPilot is that it will be very easy to translate the code from simulation use to real life use with physical drones. Dronekit has functions such as goto that allow for easy control of the UAV within the ArduPilot simulation that works within Gazebo. Other functions that it allows for use are arming of the motors as well as landing.

The sh file must be altered to call on the python code, Gazebo, the GzUav client and initializing the world and drones. This is where we will alter the number of drones as well as calling our own python files.

GzUAV is the middleware that ties all of these technologies together, particularly it helps to keep the clocks synchronized across all of the programs (python,gazebo,

ardupilot) using the SimTime library. The program works by first initializing the GzUAV client and then it launches each drone separately. Since timing is realistic to real life, this is another factor that will ease transition to physical drones. To keep time coordinated between all of the robots, Robot Net Sim is used. It is very important for the robots in particular to be synchronized in time since that would allow them to communicate, especially for the swarm AI algorithms since it will rely heavily on neighboring drones communicating with each other in a timely manner.

While using GzUAV would be optimal from a realism and distributed processing point of view it is not the only way to go about adding drones and having them run algorithms. Another possibility would be to download ArduPilot by itself and reverse engineer/follow another existing guide for manually running all of these programs together.

Drones in real Life

Real drones have flight boards which is a hardware board that has the flight stack software downloaded on it. This software is responsible for autonomous flight of the drone. In our case the flight stack will be Ardupilot. A telemetry radio is usually also included on drones in order to communicate with the ground control station. The ground control station allows for monitoring of the drones' statuses and implementation of commands. In our case, we will not be implementing any commands during runtime but this console will be useful for debugging any errors the drone models have in flight. Along with this, a typical drone will have a companion computer in which our code can run, this is where machine learning algorithms will run and tell the flight board what decisions to make. The companion computer is usually hooked up to sensors such as cameras and LIDAR in order to use these as inputs to our own code. Ardupilot is GPLed so if we change the source code, we would need to make

them open source. Ardupilot was chosen due to its bigger community and thus more support for bugs.

Alternative Drone Implementation

Ardupilot speaks to sensors with serial connection in real life but with udp connection in gazebo. A different UDP protocol is necessary for each different drone in the simulation. Gazebo has an ardupilot plugin which must be installed to integrate the two softwares.

Alternative Flight Stack: PX4

While we chose ardupilot due to its larger community size and support. The due diligence of finding all of our choices should be done. PX4 is the only other viable option to use as the flight stack. This flight stack is much more experimental and less supported by devs and the community. However, should we make any changes to the source code we would not be required to share them with the world since the code is not GPLed.

ROS Basic

ROS, short for Robot Operating System, is a set of software libraries that allows for the building of complex robotic applications. ROS is open source and has a large community of support. ROS is very broad in its applications with most packages being for ground robots. For our purposes, we will want to focus on drone packages.

Ros nodes are executable programs within a ROS environment. They are named nodes since they are all grouped in a graph. The nodes communicate through topics, services and actions. The nodes allow for a very robust robot application since if one node fails, the other nodes are not dependent on it and thus will not also fail. Beyond this, nodes will allow us to have much more organized code since each subsection of the robots actions will be contained within a node.

ROS for Drones

Ardupilot is the flight stack we will use and ROS is capable of expanding its capabilities. MAVROS is a ROS node for performing MAVLINK communication through ROS. MAVLINK is the communication protocol that is used between drones and the ground station.

roslaunch mavros apm.launch fcu_url:=udp://:<UDP Adress>@

The above command will connect Ardupilot to ROS using the MAVROS node. With the last part of the command including the UDP port which is open for the drone to communicate with the ground station. mavROS then gives the option to arm or disarm the vehicle as well as change the mode it is in.

Another ROS node of some interest would be camera and LIDAR sensor data processing. The nodes will help return maps of the environment based on the raw

input. Along with this, ROS nodes allow for calibration of sensors. This may not be a large issue in our simulated environment but is definitely something to consider for a real life application.

Python To Interact With Drones General (What We Actually Did)

In order to interact with the UAV and get them to fly there are several layers of communication necessary. On the most basic level, the drone has a cmd_vel topic which is essentially a variable that controls the speed and direction in which the drone moves. This can be published using no external packages other than the UBUNTU terminal, however since there are obstacles in the world and in any real world implementation, it is important to take pathfinding and obstacle avoidance into account.

To deal with an obstacle filled world to get somewhere the first thing that must be established is the drone and any sensors. Next some sort of localizalization must occur with the sensor input. The localization gives the drone a sense of where it is in the world and where everything else is.

We used Move Base as the navigation stack to move the robot. This is a 2D solution that takes the costmap generated by localization into account along with where you want to move and tries to get the robot there.

The exploration algorithm is the decision maker in terms of where to try and move to. It uses the costmap to determine where frontiers exist so that it can expand its knowledge of the world. We used the exploration algorithm Explore Lite.
Drone->Sensor->SLAM->Move Base->Exploration Algorithm

Multi Agent Considerations

It is also very important to know about name spaces when communicatingh

AI Swarm Algorithm

High Level Approach

When initially approaching this algorithm the thought is whether to do a centralized or decentralized approach. A centralized approach involves having one drone or computer being responsible for coordinating the other drones. A decentralized approach involves having each drone autonomously exhibit simple behavior leading to intelligent behavior on the swarm level. Decentralized approaches benefit from being less error prone since taking out one drone or computer will not cause the entire system to fail, instead the other drones can continue to do their job. For this reason, a decentralized approach was chosen.

A key question to be addressed concerning the decentralized approach will be how they should communicate. Currently I am thinking it would be good to communicate with neighbors regarding positioning so that the drones are as efficient as possible and not double checking the same areas. Also, if we were to implement a multiple altitude approach, doing so in a decentralized manner will require some thinking since it seems to be inherently centralized when determining the altitude that each drone would fly at. Another possibility could be having all of the drones working decentralized and then commanding a UGV in a centralized manner to confirm targets that the drones cannot confirm is not the target.

So far it does not look like there are many existing libraries to easily implement swarm behavior. Thus, much of this work may have to be done originally by us. So it is very

important to spend lots of time focusing on this problem since we will not have as many resources for troubleshooting.

Medium Level Approach

Some specific approaches could be to have the drones communicate distance from each other and keep to an optimum use of resources. The LIDAR sensor would be very useful for this approach since it can tell distance. One consideration is what positioning data the drones have access to, if it is relative what is the best way to communicate this. Also, the world is bounded by walls so would using LIDAR with this be a fair use of resources or would it be unrealistic and not implementable in real life.

Another piece of information that drones could share is that regarding computer vision algos and confidence scores for what they are seeing. If some drones find an area with a relatively high confidence score, it may be worth communicating this to other drones so they can focus on this area.

Saving information is another problem that needs to be approached. Once a drone determines that an area does not contain the target, how does it share this with the other drones? This is where the minimap stretch goal may be of use not only for humans watching the simulation but for the drones in deducing areas that the target is not located and thus decreasing the area needing to be searched. Although, this approach delves slightly into centralized territory since it would be one map. One solution may be to have the map be made simultaneously on all UAVs so that the swarm continues to benefit from its decentralized design.

Basic Steps

- Sensing the search region and updating the search map by individual UAVs,
- Making decisions about next search area based on the available information,
- Sharing local information.

Milestones

1. Get drones to communicate with each other.
2. Have the drones act on this information to efficiently search.
3. Establish autonomous behavior for each drone to exhibit. This will probably be tied to the other algorithms that we are using.

Swarm Algorithms

A key requirement for our swarm algorithm is decentralized behavior. This means that each actor is acting autonomously and those behaviors they exhibit individually become intelligent at the swarm level. This is very helpful in terms of computational capacity and communication bandwidth because as the number of drones increases, the computations and communication necessary for a base station to perform becomes a burden. Centralized approaches also makes the system more prone to error since there is a point at which the system could fail as opposed to every drone having to fail in a decentralized approach. Due to this, centralized swarm algorithms are not very scalable and thus will not be explored.

Decentralized swarm algorithms are often modeled on how animals swarm in nature such as how individuals behave within schools of fish and flocks of birds, benefiting from knowledge of the group. Another example being ants communicating locations of food sources leading to benefits to the swarm at large.

We will choose the algorithm that implements our goals in the simplest way possible. Key performance objectives of our algorithm are relatively simple hyperparameters to avoid excessive tuning. The algorithm should be robust to any problems an individual drone could experience. The algorithm must be easily scalable to larger areas as well as more drones. Also, the algorithm's speed is another factor as depending on the use case, speed at which the user is found versus chance they will be found must be considered.

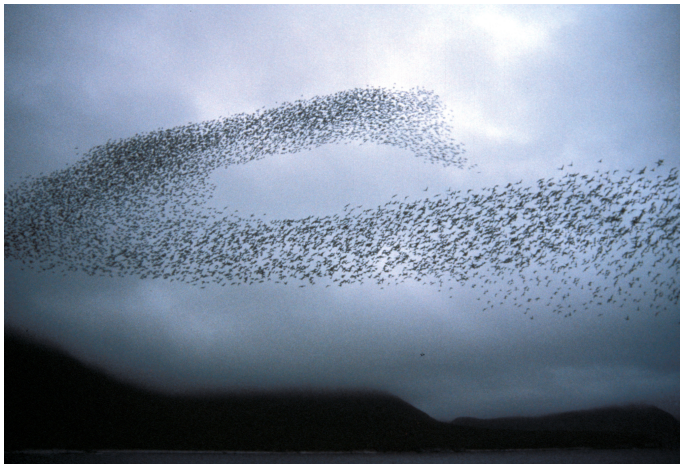


Figure 6: Flock of birds swarming to migrate

Figure 7: Ants Swarming to find food

BOIDS Algorithm

BOIDS is an early swarm algorithm based on the flocking of birds. While we would most likely not use this algorithm unmodified to do the swarm since it does not have a search component. However, we could take some of its principles into whichever swarm algorithm we choose, especially since we are using drones and this is an algorithm based on flying individuals.

The three principles BOIDS runs on are separation, which in our case would be collision avoidance between drones. Alignment, which means the drones would move in the same general direction. Cohesion is the final principle which would manifest in drones moving towards their relative center of the swarm. Alignment and cohesion are important to create flocking behavior, which we may not be after depending on the number of drones we have available. However, separation is a key principle for any algorithm we use since drone collisions need to necessarily not happen for our simulation to run smoothly.

Genetic Algorithm

The genetic algorithm is an algorithm directly inspired by Charles Darwin's Theory of Evolution, specifically natural selection. This follows the survival of the fittest trend where only the fittest individuals pass on genes to the next generation.

A fitness function is used by the algorithm on all individuals of a generation in order to determine how fit they are. Individuals are then selected to reproduce based on their fitness.

The chromosomes of the selected individuals then perform crossover. This produces offspring with the two parents' chromosome data. After this, random mutations occur to add some exploration to the process. Mutations will randomly flip certain bits in the child chromosome.

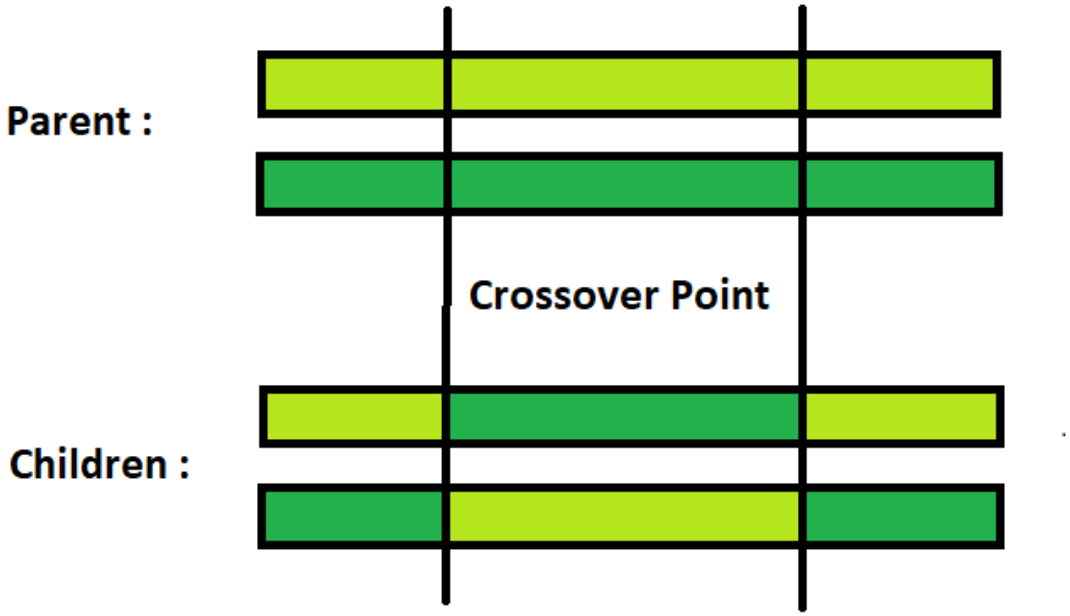


Figure 8: Crossover Example

The Genetic Algorithm works for a multitude of optimization problems and can be applied specifically for each of them.

Genetic Algorithm Hyperparameters

Generations is a hyperparameter that determines the number of crossovers that would occur. Possibly we could set this hyperparameter to infinity as we would keep trying to look until we found the target or timed out. Population size would be the number of drones and their paths used each iteration. Also, probabilities for where crossover occurs and how likely mutations are to occur.

Genetic Algorithm UAV Flight Planning

For UAV flight planning, each generation would be a UAV flight path. The fitness function would then be based on the amount of land scanned by the UAVs path. The paths would then crossover and the process would repeat. Since an array is the easiest data structure to pass to the Genetic Algorithm, it will be hard to make a three dimensional leveraging algorithm. The array would look like key value pairs of moving up or down and rotating left or right.

Particle Swarm Optimization

The Particle Swarm Optimization algorithm is highly flexible and capable of solving a multitude of problems. This is due to its ability to deal with multi-dimensional data. Also, the algorithm is decentralized and is run in parallel on each particle in the swarm.

Particle swarm optimization is an algorithm in which multiple actors are present in a search-space. Particle location is influenced by each individual particle's own best known position as well as the entire swarms best known position. The benefit of particle swarm optimization comes from the broad search areas it can deal with since there are no assumptions about the problem passed to the algorithm. This is beneficial in our case since it means that even though the target BB8's position will be random, this algorithm will make no assumptions that could lead certain target spawn cases to be less efficient. Also, having multiple actors converge on the target allows us to have multiple points of confirmation.

```
//initialize all particles
Initialize
repeat
  for each particle i in S do
    //update the particle's best position
    if f(x_i) < f(pb_i) then
      pb_i = x_i
    end if
    //update the global best position
    if f(pb_i) < f(gb) then
      gb = pb_i
    end if
  end for

  //update particle's velocity and position
  for each particle i in S do
    for each dimension d in D do
      v_{i,d} = v_{i,d} + C_1 * Rnd(0,1) * [pb_{i,d} - x_{i,d}] + C_2 * Rnd(0,1) * [gb_d - x_{i,d}]
      x_{i,d} = x_{i,d} + v_{i,d}
    end for
  end for

  //advance iteration
  it = it + 1
until it > MAX_ITERATIONS
```

Figure 9: PSO Pseudocode

Experimentation with hyperparameters for any instance of a particle swarm optimization algorithm is very important as they are very influential on the workings of the algorithm. C1 is a hyperparameter known as cognitive coefficient, this parameter is responsible for determining how much an individual particle should weigh its own personal experience when choosing its velocity. C2 is a hyperparameter known as social coefficient, this parameter is responsible for determining how much an individual particle should weigh other particle's experiences when choosing its velocity. C1 is associated with more exploration in the algorithm since each particle is more heavily influenced by its own knowledge of the area. While higher C2 values are associated with exploitation since the swarm will have a higher tendency to converge on high reward areas once enough exploration has occurred. W is another hyperparameter responsible for deciding how much each particle should remain at its current velocity. Also, due to there being a relationship between the size of search space and number of particles affecting the efficiency of the algorithm, it will be important to be explorative enough.

The number of drones used is also a very important parameter for the algorithm. Since each drone will represent a particle and the number of particles is very influential on the results of the algorithm.

Distributed particle swarm optimization

Distributed particle swarm optimization is a variant of particle swarm optimization where there are multiple particle swarms each represented by a drone. This decentralized approach would allow for even more independence as the algorithm is being used for benefit of exploration but one lone actor can not over weight a certain area do to a false positive.

Robotic Darwinian particle swarm optimization

Darwinian particle swarm optimization is particle swarm optimization with Darwinian selection included in the algorithm. The logic is essentially natural selection where high performing particles are given higher status in the swarm and low performing particles are excluded from the swarm. This provides the benefit of defeating the local optima problem since it allows for the swarms relative performance to other swarms to determine its influence on the results. The local optima problem leads to suboptimal solutions being chosen.

PSO for Aerial Vehicles

In order to implement PSO for aerial vehicles several factors have to be accounted for.

Velocity would need to be tracked for every drone so that this can be changed by the algorithm in response to that particle's and the group's best known position. Target presence will have to be defined in a way that it can be optimized for, maybe some form of uncertainty. This value would provide the reward that would be the feedback for the algorithm. Furthermore our implementation would probably be done in python as this seems to be the best language to connect machine learning algorithms to drones. Persistence of communication between each drone is also a factor since collisions could occur if the communication is periodic and two drones start a collision path between communication periods.

Due to their also being a z-axis in which the drone can move, velocity would have to be calculated in three dimensions. Euclidean distance and cosine similarity may be useful functions when comparing these vectors. The search area would also have to be gridded in an almost blank map sort of way since this algorithm relies on this grid to run.

Collision and Object Avoidance with PSO

One solution may be to use time as a fourth dimension for PSO so that none of the particles inhabit the same area at the same time.

For PSO a distributed system would mean having swarms of swarms. It will be key for the drones to avoid other drones within their own swarm and to also avoid drones in other swarms. If a distributed approach is not used then each drone will just have to avoid each other particle in the one swarm. One research group led by David Paez used artificial potential functions to have swarms avoid each other.

Algorithm Comparison

In terms of simplicity, the PSO algorithm would be easier to implement for our use case since it is built for arbitrarily multidimensional data and our drones will be moving around in a 3D space (4D if we account for time). An added benefit of a 4D algorithm is that moving targets could be accounted for since there are temporal factors. While

BOIDS is a simpler algorithm than PSO, it does not provide enough flexibility to apply to a search and rescue situation.

In terms of hyperparameters, both the PSO algorithm and the GA algorithm would require some tuning. Here, neither the PSO algorithm or GA algorithm would present a distinct advantage. The PSO algorithms hyperparameters would be more obvious in their effects are relatively straight forward. Whereas we would really have to study and determine an approach for GA algorithms to effectively use the algorithm and tune its parameters. While GA is possible to be used for a multitude of use cases, implementing UAV flight into chromosomes would be a serious undertaking . Another issue with GA is that it would be difficult to scale up to swarms of swarms since the algorithm is not built in a way in which it can be distributed and no current implementations exist doing so.

Due to having a simpler implementation and more flexibility for our case, PSO was chosen as the algorithm to implement our swarm. In order to account for collisions, a 4-dimensional PSO input will be used to implement the separation principle of the BOIDS algorithm. While a distributed PSO algorithm has shown in some papers to be better than a PSO algorithm where each drone is a particle. The feasibility of having swarms of drone swarms must be considered and would be reliant on how well we could distribute the computational load of all of the various subsystems in the simulation.

The darwinian approach to the distributed PSO swarm may be very useful in avoiding edge cases where local minima would cause drones to oversample areas where the score is getting disproportionate influence on the algorithms working. It essentially adds a relativity component to the personal best spot since it’s influence will be in reference to the other particles scores.

It may be useful to implement a combination of algorithms. For example a BOID swarm being the swarms that make up the distributed PSO algorithm's swarms. Though this would leave the question of how each swarm would determine it's reward, a possible solution may be averaging the groups reward.

Object Detection

For our drones to successfully find it's target, an implementation of real-time Object Detection. This was a task required by the previous project's team as well and the initial decision was to read over their documentation to see what they decided on and why. They had used YOLOv3 and said if they were to iteratively continue their project that YOLOv4 would have been the go-to pick without question. After our own research, it is fair to say that YOLO is still the way to go when it comes to real-time object detection due to its ridiculously fast speed for both training/testing and implementation. But, the version they were thinking about is now also outdated leaving us with YOLOv5 as our selected algorithm.

How to Choose an Algorithm for Object Detection

When it comes to object detection, there are a few different variables that one must consider. The main ones I want to put focus on are the following:

- Speed
- IoU
- Precision
- mAP

Speed:

When it comes to the efficiency of an algorithm, we want it to be fast. And for real-time implementation, we are talking within milliseconds. What we will need to consider is when it comes to object detection there are two variables that tend to be opposites of one another, this is one of them. To be faster is typically a trade off for mAP.

IoU:

When it comes to identifying a target, or when talking about object detection we call it a “class”, we care about the accuracy of our prediction. When you are testing an object detection model you will give it the image without what we call an image label. An

Figure 10: Example of a bounding box

image label, also known as an “annotation”, is a bounding box of four points that within it contains the class we are aiming to train our object detection model on. This is an example of an annotation below.

You can see that there are three elements that an annotation is made up of: The box, the class name, and the confidence score.

The bounding box, as stated earlier, is the given area where the target that we want is located from within the image.

The class name is the *classification* of the target. If you remember from Biology the topic of Taxonomy, where classifying organisms was determined by characteristics and qualities that are unique to each class. Object detection works very much the same way, hence the name class. One thing to note: for more robust uses of object detection, you can have multiple classes for a given model. The algorithms we have looked at *do* have these capabilities, but for our project, we will only be worrying about a single class. Finally the final category is the confidence score.

The confidence score is only shown on an *augmented result*. What this means is that our object detection model creates/generates, or as we will refer to from now on as augments, a guess or prediction of where the target is. The confidence score is a statistic that describes how “confident” the model is that the augmented result is correct compared to being incorrect. For the given example above, we can see that it is 99.99% confident that the augmented result is indeed a stop sign.

Now that all has been covered when it comes to what an annotation is, now we can talk about what IoU is. So IoU stands for *“Intersection over Union”* with is exactly what the name implies. Using set theory, we take what we call the “Ground Truth” annotation, which would be the annotation that we had given the model for train/testing purposes and is 100% accurate, and the “Prediction” (Same thing as the Augmented Result. There are many different names and all are correct) and generate two numbers.

First we calculate the *Overlap* which is the intersection of the Ground Truth’s Area and the Predicition’s Area.

$$Overlap \ = \ AREA(Ground\ Truth) \cap AREA(Prediction)$$

Then we calculate the *Union* which as the name implies is the union of the Ground Truth’s Area and the Prediction’s Area.

$$Union \ = \ AREA(Ground\ Truth) \cup AREA(Prediction)$$

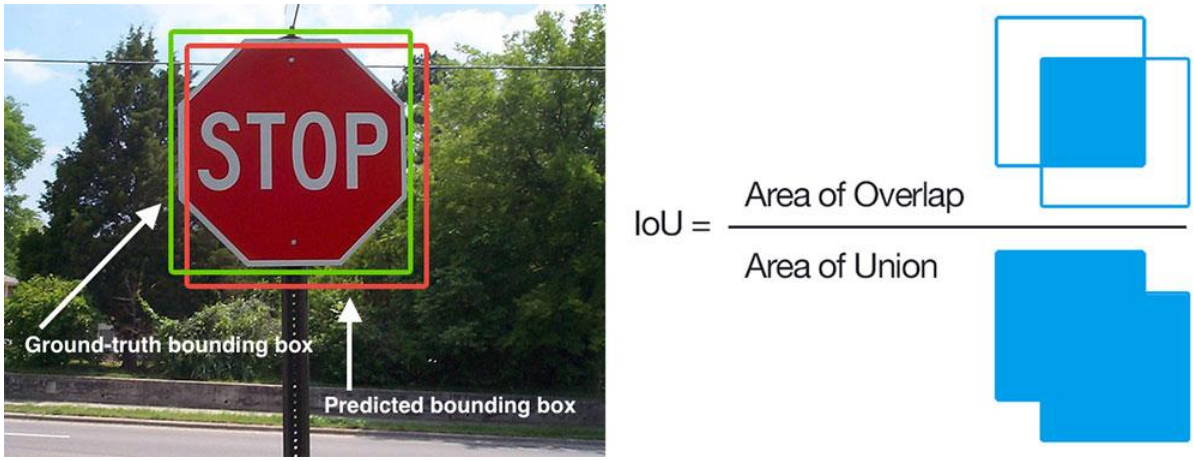


Figure 11: Further example of bounding box

Using the formula above, we would then generate our IoU. The larger the value, the more accurate the prediction was to the truth. Note, for our project the requirement is for the algorithm to have an IoU > 0.5 with a confidence score > 0.7. As explained later on, YOLOv5 meets these requirements.

Precision:

We just explained what IoU metric is and how it is generated. Now that is a great metric for accuracy but when it comes to any Machine Learning, there is always a chance that there could be a lack of *consistency*. That is where the Precision statistic comes in. What Precision does is take into account how consistently the model is accurate and consistent by determining whether or not the model successfully found the target but more importantly that it was found *intentionally*. There are four metrics that are used to help evaluate this: True Positives, True Negatives, False Positives, and False Negatives. A matrix below visualizes how we determine these metrics. Then an explanation of how those are found when it comes to Object Detection will follow.

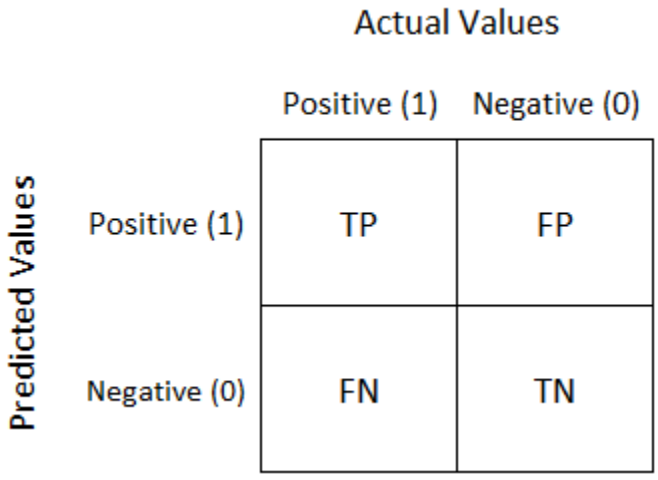


Figure 12: Precision matrix is shown

How to get a True Positive (TP): If a prediction is made and the class is there, it is a True Positive.

How to get a False Positive (FP): If a prediction is made and the class is **not** there, it is a False Positive

How to get a True Negative(TN): If there is no class present and no prediction is made, it is a True Negative.

How to get a False Negative(FN): If there **is** a class present and does not make a prediction, it is a False Negative.

So how does Precision come into play? Well Precision is created using the following equation:

$$Precision = (TP) / (TP + FP)$$

Precision is the ratio of True Positives to all Positive results for a given test. So for example, if you use a batch of 100 images to test the model, say there were only 50 Ground Truth Positive images and the others were all 50 Ground Truth Negatives. If the Model had 50 True Positives but it also had 10 False Positives, it would have a Precision of 0.83. So this was for a single batch but when you test you do what is

called an *epoch* which stands for how many times you run the test for a given batch size. We need a better way to combine Precision for large epoch counts on large batches of images.

mAP:

mAP stands for mean average Precision. So it takes the Average Precision of every epoch of a given test and puts that value over multiple IoU thresholds. Remember how we mentioned earlier how we must have an IoU >0.5? This is why we are not concerned about this threshold as the thresholds used for benchmarking Object Detection algorithms are from the range [0.5:0.95] where it is every threshold from 0.5 to 0.95 using a 0.05 interval. Why does this value matter? Well it is a very informative value that is the result of many calculations described earlier and accounts for accuracy, consistency, and makes sure it is above a given standard. The higher this value, the better it is at being consistently accurate for the given standard. This is typically a trade off for speed.

YOLO - You Only Look Once

Despite the very punny name, it is the right algorithm for us. When deciding on the goals of our project we want it to be able to detect fast and accurately. The two most recent algorithms to shine in the spotlight are YOLOv5 and EfficientDet. This is where the two values about speed accuracy come into play.

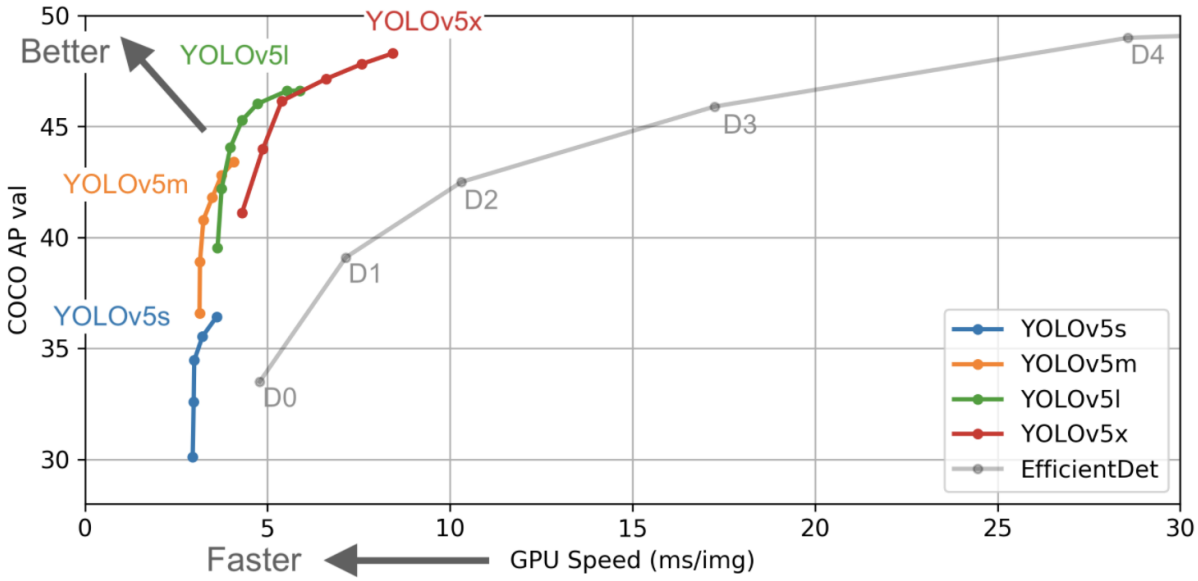


Figure 13: Comparison of YOLOv5 versions

First off, COCO is what is referred to as a benchmark dataset for testing object detection algorithms on a given standard. Simply said, COCO is that standard. As you can immediately see, YOLO algorithms **all** run faster than EfficientDet. Second off, the discrepancy of results from multiple tests all prove to be much more consistent compared to EfficientDet. The method that YOLO uses allows for it to be much faster and uncontestable. Although the graph doesn't show it, YOLOv4 also runs faster than EfficientDet! So as the graph shows, the top left is what is the best possible result: Fastest and has the highest mAP value (Note that sometimes the 'm' in mAP is dropped).

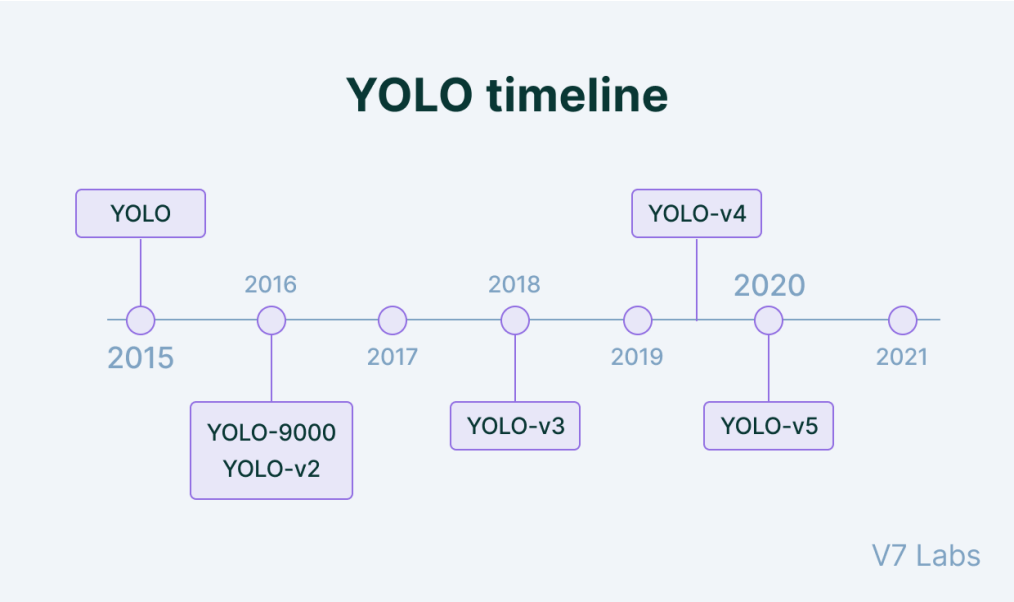


Figure 14: YOLO iterations are shown

So then the question is why YOLOv5 compared to YOLOv4? First off, YOLOv5 is the first to not be backed by darknet which is a neural network written in C. YOLOv5 has the benefit that it is 100% written in Python! This means the entire Convolutional Neural Network (CNN) is written in Python and they also built it on PyTorch, an open source Machine Learning framework allowing the entire algorithm to operate in Python. The other reason is that YOLOv5 was built as open source and is as simple as cloning their github repo, importing your images and annotations and running the train and the test models! The ease of use, efficiency, and entirety being in python allows for us to have even easier implementation with the python-opencv library for use of our model. At the moment of us working on our project, the decision for picking YOLOv5 is a clear choice. Remember how we mentioned it is open sourced? It is still being improved to date! And there are more versions of YOLOv5 since that graph was created. At the moment, YOLOv5 is uncontested and is actively contesting itself. If this project was to be continued, it would seem as simple as reusing the training data and running it on the latest model assuming that YOLOv5 is still that standard.

How does YOLO work?

When it comes to Object Detection, other algorithms tend to use a Two-Stage process: Detecting possible object regions and classifying classes within the image based off of those object regions. This means that it must do one step before the other is even performed, meaning a longer runtime. YOLO, as the name implies, does it all at once by only looking at the image once and classifying at the same time. Below is the process of how YOLO tackles the problem.

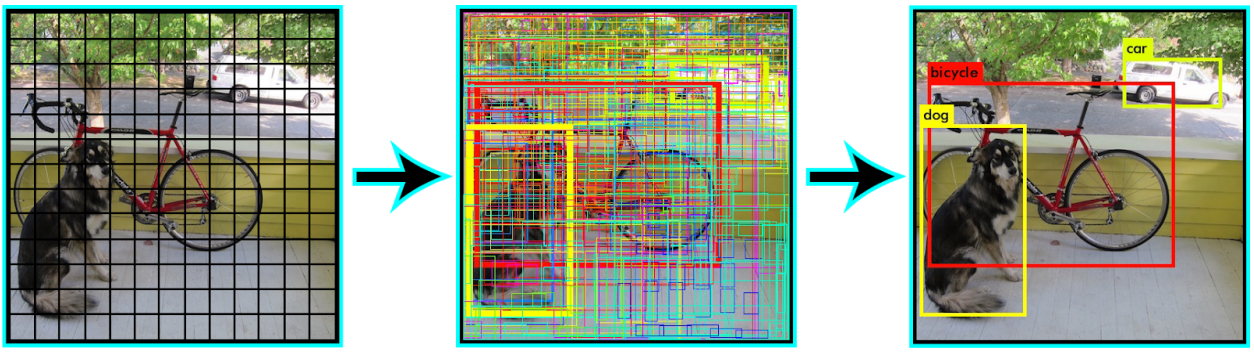


Figure 15: Example of YOLO algorithm in action

Step 1 - Gridify The Image:

The very first thing YOLO does is compress the image into a square using a CNN architecture of 24 layers as shown below.

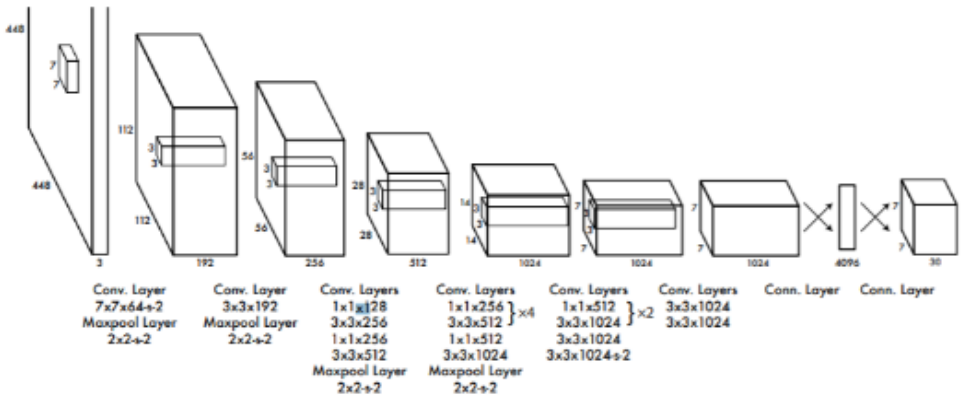
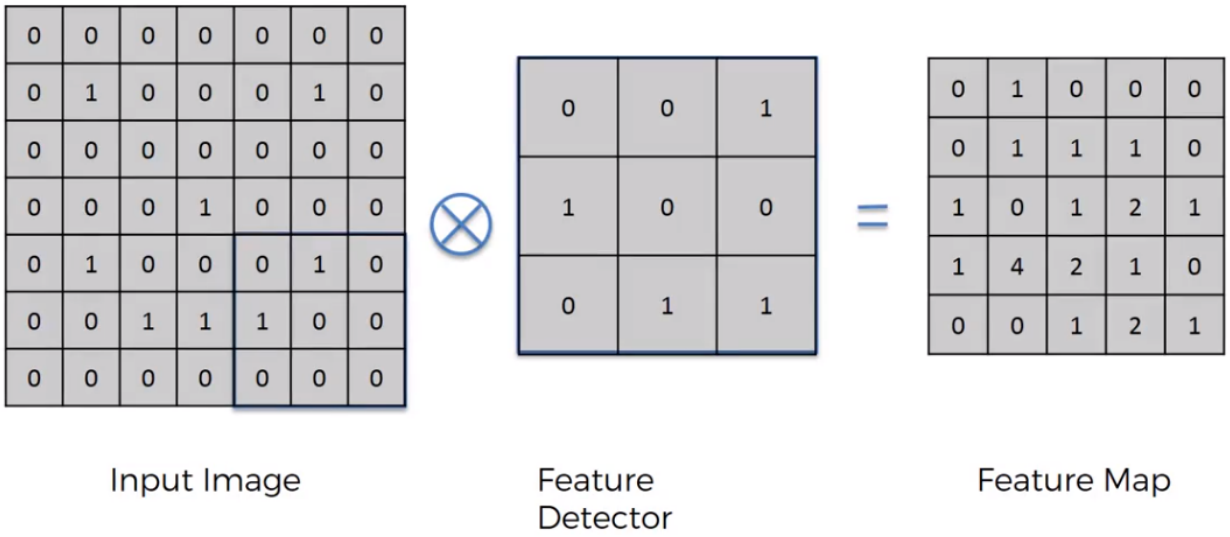


Figure 16: Example of CNN architecture

It then proceeds to create an S x S grid where S is the cell count vertical and across. For the example above it would be 13 x 13. Each of these cells are then responsible for detecting and predicting the classes within them by creating annotations relative to coordinates of the cell itself. The input is a 3 dimension feature map, or in our case the image; the R channel, the G channel, and the B channel. Using a sliding window technique, it utilizes a kernel or feature detector to multiply and decrease the features from layer to layer. An example of how the output feature map is generated is shown below.



The YOLOv5 model utilizes the Leaky ReLU and Sigmoid activation functions. The Leaky ReLU is utilized on the hidden layers of the network while Sigmoid is utilized for the fully connected, output layers. When it comes to training, the model utilizes Stochastic Gradient Descent by default and training the hyperparameters can be optimized accordingly using the “--evolve” flag in the YOLOv5 repository. Likewise, you have to option to train loss using Adam if you edit the hyperparameters using the “--adam” flag in the YOLOv5 repository. The loss function utilized is the PyTorch Binary Cross-Entropy with logistic loss for calculating the loss of the class probabilities by default. You can opt to use Focal Loss if you so choose by setting it’s respective hyperparameter to true.

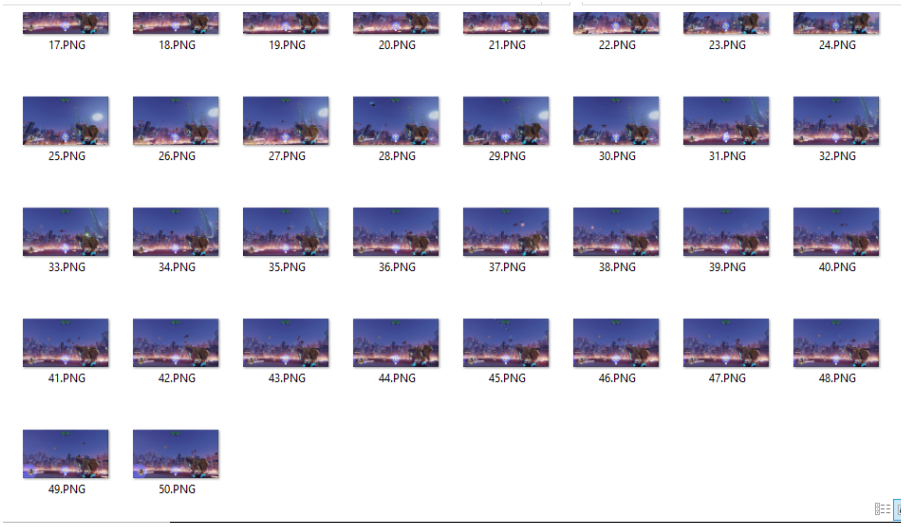
Step 2 - Non Maximal Suppression

As you can see in the second image, after all the cells have generated their predictions with different scores of confidence, good and bad, it has many duplicates. YOLO has a process called Non Maximal Suppression which does two things. First it filters out all low confidence score predictions and then it proceeds to look at the largest Confidence Score predictions if any other guesses that are lower have a large IoU and filters those out as well. This process is repeated until the final bounding boxes are all that remains. These final bounding boxes are the annotations seen in the last image.

Due to all of this happening in one runthrough of the image, this allows the YOLO algorithm to be significantly faster than other Two-Stage algorithms such as Faster R-CNN. Whereas Faster R-CNN tends to barely run at 30FPS on some implementations, YOLO is capable of running at 45FPS and sometimes even 52FPS! That is really fast! YOLO, as stated earlier, is uncontested in the current state of Object Detection research and doesn’t to be going anywhere anytime soon.

Testing - Using YOLOv5s Train/Test Model on a Custom Dataset

After learning about Machine Learning, Object Detection, we proceeded to verify that YOLOv5 will work for us on a custom dataset. First step was to go and gather images for a specific class with different angles and locations. In a game called Overwatch, there is an aim trainer that randomly tosses various characters in the air to practice tracking but for our purposes, it was an approachable way to successfully gather data on a target with different angles and positions. The character that was chosen to be the "class" was Reinhardt and gathered data is shown below:



Each image was named from 1 to 50 leaving us with a 50 image sample size. Following this process, a site called Roboflow was recommended on the YOLOv5 GitHub repo for creating annotations for each individual image. After signing up for free, you import your images and it will bring you to an annotation creator screen. An example of creating an annotation is shown here:

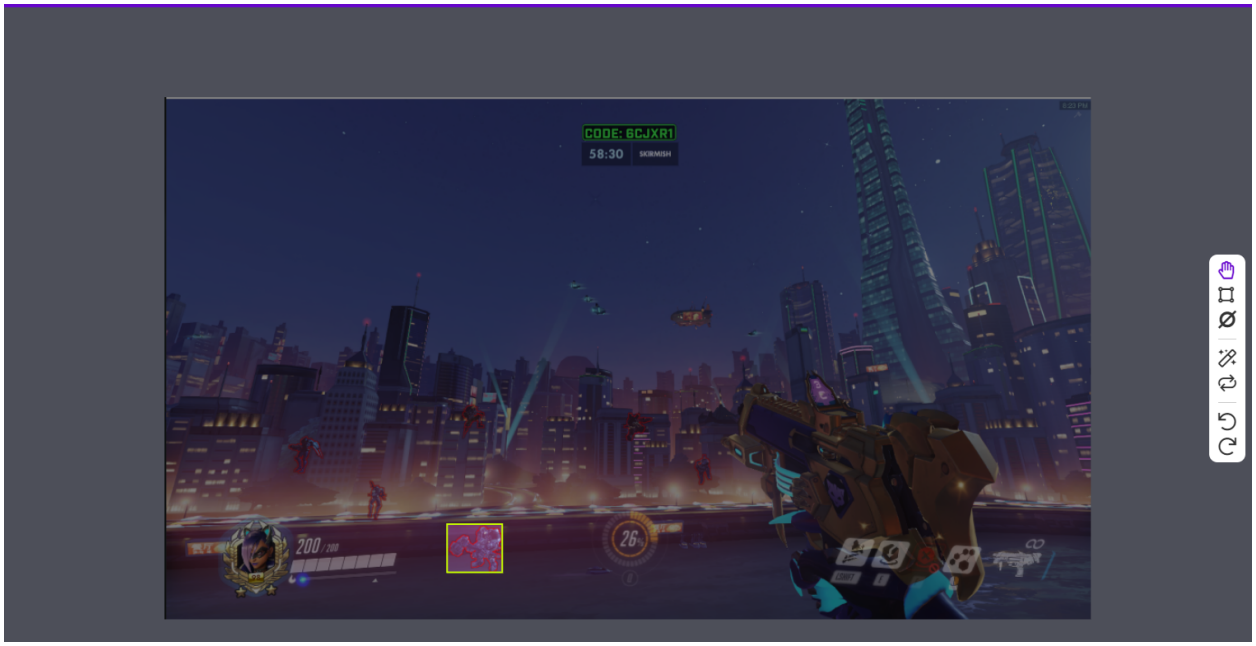


Figure 17: Annotation creation

Using the bar GUI on the right, you select the square icon and simply place one corner on the image and then the opposite corner and that creates your box. I would highly recommend this option for anyone trying to do Image labeling with ease. It is a tedious process due to being a manual approach but it allows for very precise annotations.

Once you finish annotating your images, you can pick your percentages for creating your Train and Test folders. Roboflow will randomly and automatically do this for you after selecting your percentages. For this test, the data was divided as such: a 70% Training Set, a 20% Validation Set, and a 10% Test Set. Then all you do is simply export the correct image labeling format for your algorithm and it will give you a train and test set folder with your images AND annotations for you. For the YOLOv5 Train/Test model we need to export the annotations in “YOLO v5 PyTorch”. This will give you options on how to unzip depending on the use case. For our sake, copy the

code block for the Jupyter Notebook section. As Roboflow already will state, do not share your api key as it points to credential information as well as your data set.

Now for the actual training part, Roboflow has created their own Google Collab that gives you a boilerplate for training and testing your custom data set with their generated code.

The first thing you should do immediately is to make sure all training is utilizing your GPU instead of your CPU. At the top of the Notebook, follow this path to change to GPU.

"*Runtime → Change Runtime Type → Hardware accelerator → GPU"*

After that, we begin to follow along with their Google Collab, the second step is we import the dependencies. So first we will import the YOLOv5 repository into the Notebook by running the code block. After that completes, we will import the PyTorch, the library for displaying images "IPython.display" and googles library for downloading models and datasets. When successfully done the outputs should look like this.

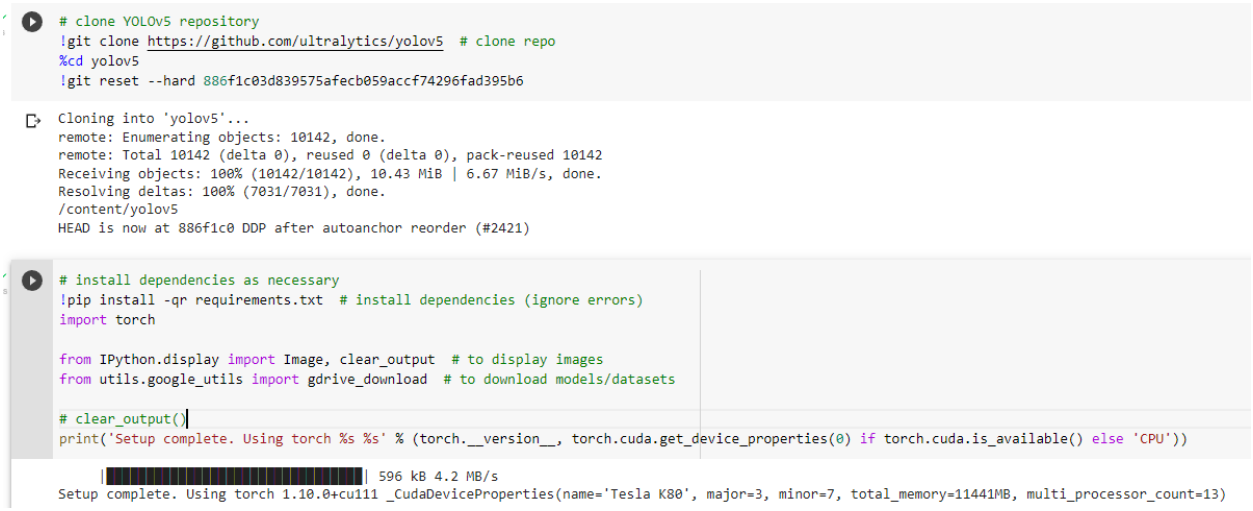


Figure 18: Correct model output

This next part is where that code block that we copied earlier comes in. Underneath the image that should look very familiar as this was how we exported our labels from Roboflow. In code block below the image, replace the entire code block but verify that the import statement is

“!pip install -q roboflow”

Run this and it will import the dataset from Roboflow. After these have been completed I recommend verifying that your data successfully imported into drive by clicking the folder on the left side of the Notebook. After that, click the folder with an arrow on it. Follow this path:

“content → yolov5”

You should see whatever you named your project on Roboflow be a folder. If this holds true, everything so far has worked properly.The proceeding line just moves the Notebooks scope into the yolov5 folder in google drive. The code block after that will run the YAML file provided by Roboflow to load in your data. You can see this file by opening that folder named after your project.

The next section is where you are able to change the parameters of your model for testing such as classes, anchors, etc. You don't need to change these unless you need to for your model. After that section is the fun part, we finally are training our YOLOv5 model using our custom data set. This is where we will change some arguments to change the performance of our model. Using statistics that are generated later, these arguments will be what you tweak to perfect your model for a final production version.

The arguments are shown below:

- **img:** define input image size
- **batch:** determine batch size
- **epochs:** define the number of training epochs. (Note: often, 3000+ are common here!)
- **data:** set the path to our yaml file
- **cfg:** specify our model configuration
- **weights:** specify a custom path to weights. (Note: you can download weights from the Ultralytics Google Drive folder)
- **name:** result names
- **nosave:** only save the final checkpoint
- **cache:** cache images for faster training

Figure 19: YOLOv5 argument list

Image Size: Remember that the images were formatted to a square because of how YOLO works as described earlier, so do not change that.

Batch: This is the total amount of images that will present in a single training run.(Should be the same number as however many images in

Epoch: An epoch is how many times the algorithm has passed through the entire dataset. (Should be nothing less than 1000)

Cfg: this is the configuration of YOLOv5 you plan to run. The github for YOLOv5 shows a graph of all the stats for each model, refer to the folder on the left side, go to the directory “hub” within the yolov5 directory for the options and compare the names to the graph on the YOLOv5 github. (See below)

Model	size (pixels)	mAP^{val} 0.5:0.95	mAP^{val} 0.5	Speed CPU b1 (ms)	Speed V100 b1 (ms)	Speed V100 b32 (ms)	params (M)	FLOPs @640 (B)
YOLOv5n	640	28.4	46.0	**45**	**6.3**	**0.6**	**1.9**	**4.5**
YOLOv5s	640	37.2	56.0	98	6.4	0.9	7.2	16.5
YOLOv5m	640	45.2	63.9	224	8.2	1.7	21.2	49.0
YOLOv5l	640	48.8	67.2	430	10.1	2.7	46.5	109.1
YOLOv5x	640	50.7	68.9	766	12.1	4.8	86.7	205.7
YOLOv5n6	1280	34.0	50.7	153	8.1	2.1	3.2	4.6
YOLOv5s6	1280	44.5	63.0	385	8.2	3.6	16.8	12.6
YOLOv5m6	1280	51.0	69.0	887	11.1	6.8	35.7	50.0
YOLOv5l6	1280	53.6	71.6	1784	15.8	10.5	76.8	111.4
YOLOv5x6 + TTA	1280 1536	54.7 **55.4**	**72.4** 72.3	3136 -	26.2 -	19.4 -	140.7 -	209.8 -

Figure 20: Comparison of YOLOv5 versions

Weights: This is usually used to strengthen the connection between two nodes in the CNN but since we are only classifying one class, we will opt to leave it as is by default which is blank

Evolve: An important parameter that allows the user to optimize their hyperparameters specific to their dataset. This was an extremely useful tool that we took advantage of to find the success we had in the v4 iteration of our model. Our specific use-case was to run ten epochs three hundred times in order to increase our fitness value, which is a weighted combination of metrics: mAP@0.5 contributes 10% of the weight and mAP@0.5:0.95 contributes the remaining 90%. In doing so you're allowing your model to train more precisely towards your data set. Each run of the model is what's referred to as a "generation". The main genetic operators are crossover and mutation. Evolve running with default settings will work with a 90% probability and a 0.04 variance to create new offspring based on a combination of the best parents from all previous generations. Results are tracked in yolov5/evolve.txt, and the highest fitness offspring is saved every generation as yolov5/runs/evolve/hyp_evolved.yaml. In our use case, evolve ended up significantly reducing the amount of False Positives we were running into which was the main drawback of our v4 iteration. Although do be wary, make sure to dial back some of the augmentations done through Roboflow so that you are not double dipping too much in the image augmentation. Given that the hyperparameter optimization does take into account the augmentations done beforehand on the image, it might oversaturate each image with augmentations and not fulfill its intended purpose as a True Positive. This can cause more problems than it would if we didn't use evolve at all.

These are the main things to change when altering our model's efficiency. As described earlier, it is usually a give and take when it comes to speed and precision. Once you run that line, it will begin training your model based on the given arguments. If everything goes as it should, you will start seeing all of your epochs generated in the output as shown below:

```
           Class      Images     Targets           P           R      mAP@.5  mAP@.5:.95: 100% 1/1 [00:00<00:00,  2.80it/s]
             all          10           0           0           0           0           0

    Epoch   gpu_mem       box       obj       cls     total   targets  img_size
     5/99     1.88G   0.09753   0.01657         0    0.1141         3       416: 100% 3/3 [00:01<00:00,  2.11it/s]
           Class      Images     Targets           P           R      mAP@.5  mAP@.5:.95: 100% 1/1 [00:00<00:00,  2.78it/s]
             all          10           0           0           0           0           0

    Epoch   gpu_mem       box       obj       cls     total   targets  img_size
     6/99     1.88G   0.08514   0.01822         0    0.1034         7       416: 100% 3/3 [00:01<00:00,  2.10it/s]
           Class      Images     Targets           P           R      mAP@.5  mAP@.5:.95: 100% 1/1 [00:00<00:00,  2.84it/s]
             all          10           0           0           0           0           0

    Epoch   gpu_mem       box       obj       cls     total   targets  img_size
     7/99     1.88G    0.1019   0.01845         0    0.1204         7       416: 100% 3/3 [00:01<00:00,  2.08it/s]
           Class      Images     Targets           P           R      mAP@.5  mAP@.5:.95: 100% 1/1 [00:00<00:00,  2.88it/s]
             all          10           0           0           0           0           0

    Epoch   gpu_mem       box       obj       cls     total   targets  img_size
     8/99     1.88G    0.1115   0.01691         0    0.1284         5       416: 100% 3/3 [00:01<00:00,  2.11it/s]
           Class      Images     Targets           P           R      mAP@.5  mAP@.5:.95: 100% 1/1 [00:00<00:00,  2.92it/s]
             all          10           0           0           0           0           0

    Epoch   gpu_mem       box       obj       cls     total   targets  img_size
     9/99     1.88G   0.09643   0.01572         0    0.1121         5       416: 100% 3/3 [00:01<00:00,  2.13it/s]
           Class      Images     Targets           P           R      mAP@.5  mAP@.5:.95: 100% 1/1 [00:00<00:00,  2.99it/s]
             all          10          10    0.000341         0.1     0.00107    0.000107

    Epoch   gpu_mem       box       obj       cls     total   targets  img_size
    10/99     1.88G   0.09931   0.01529         0    0.1146         4       416: 100% 3/3 [00:01<00:00,  2.11it/s]
           Class      Images     Targets           P           R      mAP@.5  mAP@.5:.95: 100% 1/1 [00:00<00:00,  2.91it/s]
             all          10          10    0.000348         0.1    0.000136    1.36e-05

    Epoch   gpu_mem       box       obj       cls     total   targets  img_size
  0% 0/3 [00:00<?, ?it/s]
```

Figure 21: Epoch output

This will take a while, but utilizing GPU is much faster than CPU, so be sure that it was enabled as described earlier. You can estimate the amount of time it will take based on each epoch's average execution time multiplied by the amount of epochs that you have predetermined. Do not count the first three epochs run time as those are what's called "warm-up epochs". Hopefully, while running your epochs, you will see an increase in values like mAP@.5 and mAP@.5:.95. Noticing these constantly going in an upward trend is a sign of a well trained model. Any time that you finish with a lower value than the value described from a previous epoch, you will know that your model has been overtrained and to reduce the amount of total epochs. Thankfully though, when pulling your .pt file that holds the model weights, Yolov5 will automatically give you weights from right before your model overtrains which allows for longer trained models to not have such downside.

Google Collab WARNING!!!!

Google Collab will have a runtime error if the notebook is left idle for 30 minutes. What does this mean? Well it means everything in the collab is reset to the beginning. No model, no dataset, no import! Either manually ensure you are working on the page or open DevTools and navigate to the Console. Then, paste this script below and it will create a new code block to prevent a disconnect from the Notebook.

—----------

function ClickConnect(){

console.log("Working");

document.querySelector("colab-toolbar-button").click()

}setInterval(ClickConnect,60000)

—----------

This is crucial for production when the epochs will be very large and to lose all that time and data is very inefficient. When testing the process, one run was left running and went faster than expected. The person working on it had gone to eat while running epoch 1000 on a batch of 50 and came back to the disconnect. Needless to say, they were a tad frustrated so make sure to insert the script above whenever running a test on a model. If you leave it running, the pop up saying there was a disconnect will still show but it didn't actually disconnect. If you just close the popup, you can verify this by checking the folder on the left and looking for the previously described paths. You will see that they are still there.

Another solution is to pay for the premium Google Colab service, Google Colab Pro, or to set up your own runtime for Google Colab to connect to. Using Google Colab Pro,

you are able to use Google's resources more frequently and for longer periods of time. You are also able to use an increased amount of ram. The latter of the two options is a bit more difficult to set up but depending on your personal hardware, you could see some dramatic decreases in time to execute each epoch as well as not getting timed out using Google's runtimes.

Statistics and Testing the Model

After the training is finished, the next code blocks show some graphs that show the performance statistics of your model. The first code block there uses TensorBoard. We want to see a positive trend and are looking to see if it ever capped out. Here is one of the prototype runs results on TensorBoard:

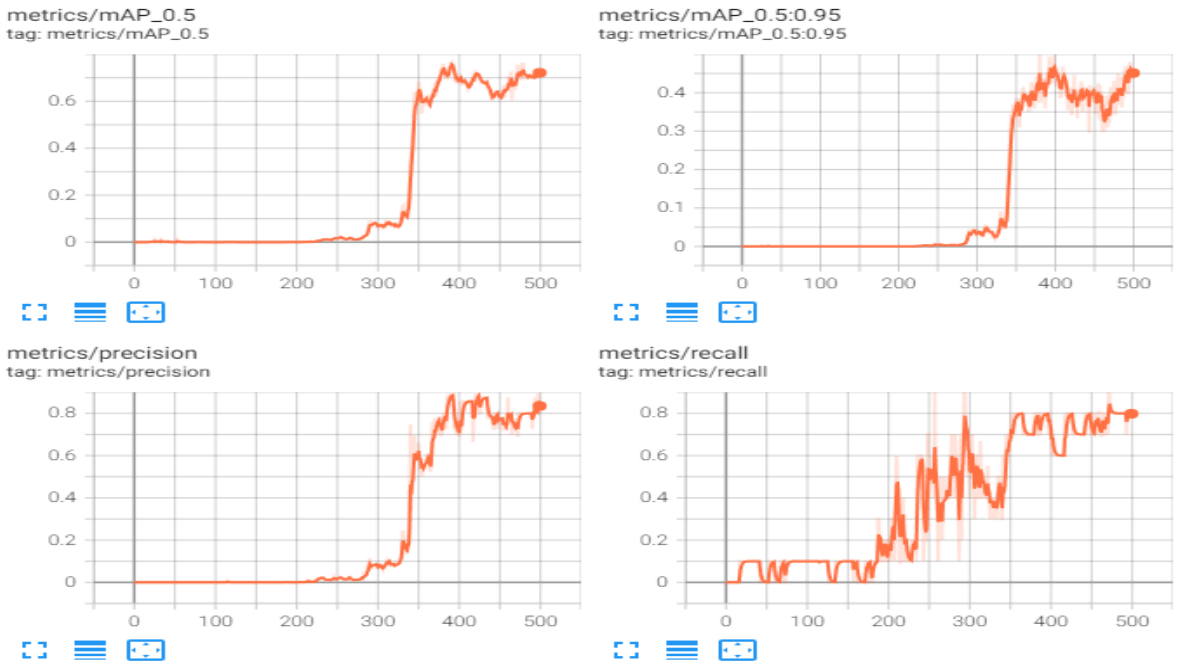


Figure 22: Prototype runs on TensorBoard

As you can see, the epochs used on this test run was 500. Around 350 everything began to make vast improvements. It appears that it capped out in precision around a .90 confidence and the mAP capped at about 0.5. What to do with these stats is consider what more epochs would have done for the model. It appears that around epoch 400, our best model was designed and then we overworked the model and the performance began to dip. So for the batch size used, 400 would result be our best model and running the test again with this knowledge would result in a better model.

There are also more vanilla graphs if the Tensorboard fails to work in the code block below Tensorboard.

Below this is our testing code blocks. As great as statistics are, it is important for us to identify that it was successfully finding the correct target with your own eyes. The code block below will show the ground truth test batch while the one below it will show the prediction test batch. Use this to verify if everything worked as expected. If it didn't refer to the arguments and statistics to generate a better weight.

This will show the test run of your weight after training. Here were four of the predictions from the best weight on the prototype dataset.

Object Detection Prototype Results:

Result 1:

Result 2:

Result 3:

Result 4:

After this, the following code blocks test the inference speed of your model, so run those to see how fast your best weight detects the target. The code blocks are there for you to save your model. If for whatever reason the last code block to move your generated weights into your google drive, using the folder on the left, just open the “gdrive” in content and drag “yolov5” into “My Drive”. Now the weights for your classification are usable.

That being said, this is the process planned for implementing on our custom data set, “BB-8,” when we are implementing it for real. We will add our weight in using detect.py, the source of input (the simulation using a drone camera), and the model. Figuring out how to use the generated text file to halt the simulation when the target is successfully found will be the starting point for Senior Design 2 in this subject. We will also verify which configuration we aim to use for production and test which we prefer for our final version.

YOLOv5 Object Detection Implementation:

When it came to actual implementation, we utilized opencv in Python to gather our images. We edited our drones models to create new nodes for us to connect to our Python ROS files that were responsible for reading in the video feed from the cameras, feeding the images through our active YOLOv5 model, and using confidence thresholds, decide if it was a detection or not. Our threshold was set to 0.6 confidence. The drones publish the image feed to a ROS node that we then take in using a subscription in our python file. The data type is that of a Img_msg in which we used pillow to convert it to a PIL image. This is the image that we then utilized to run through the model. If the output was a detection, we used that image and converted it to BGR for opencv to publish the image with the proper channels. This image that is shown to the user has a blue bounding box around the target.

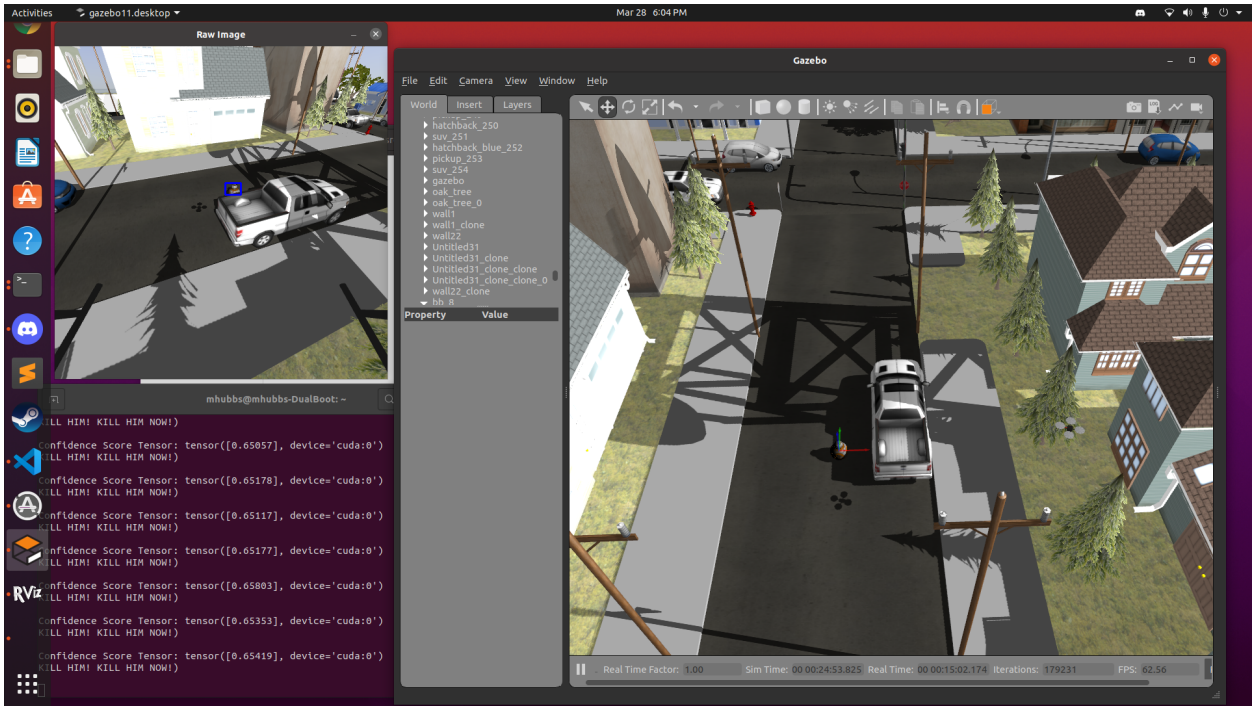


Figure 23: Object Detection in Gazebo

When gathering our dataset, we utilized the same camera and manually flew the drone around to gather images of True Positives and True Negatives. A True Positive is when you tell your model that you are training that bb-8 is absolutely in view. This is done through the annotation software inside of Robolow. We decide an image is a True Negative when you tell the model that there is absolutely no bb-8 inside of the image, also one using Roboflows annotation software. We wrote a different ROS python file that also subscribed to the camera feed of the drone and we put a listener on the spacebar for us to take the pictures with. When we press the spacebar, whatever frame of the video feed is there, an image file is created and saved into a folder for us to annotate as described earlier in the prototype. Using this base data set, we were able to increase our total data significantly by using Roboflow's built in augmentation tool.

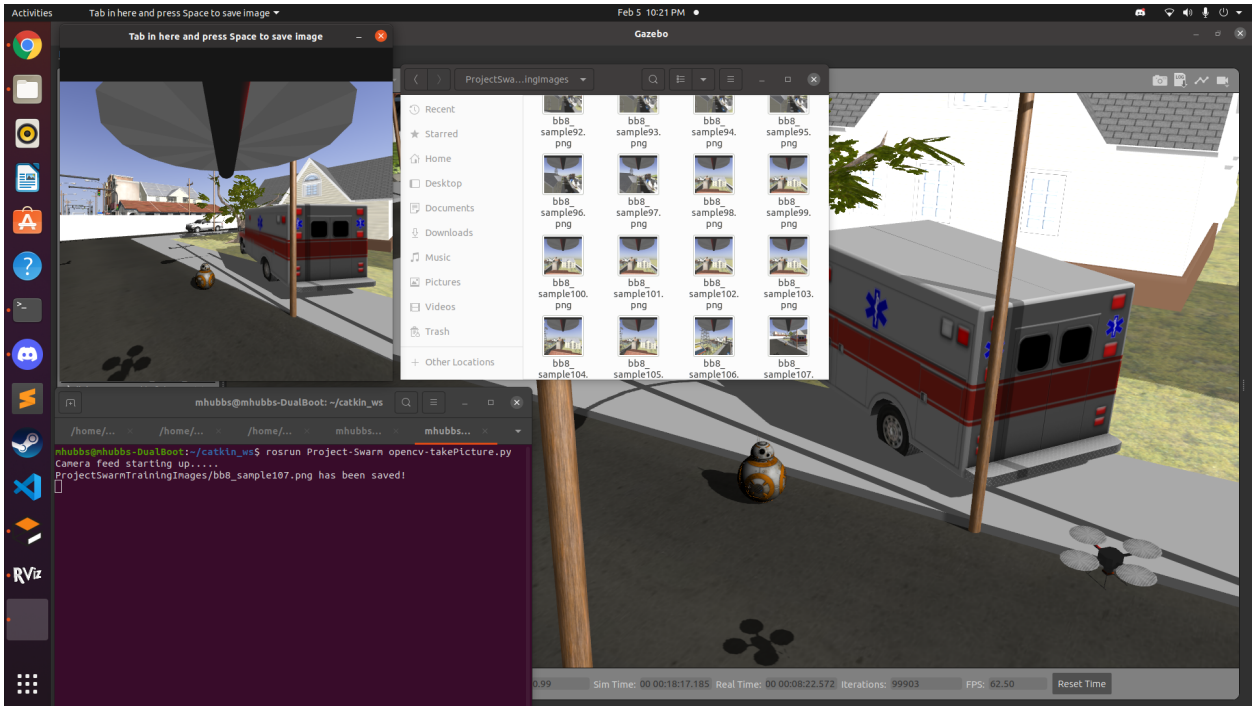


Figure 24: take_picture.py in action

We trained 6 versions of our YOLOv5 model and the best model resulted in our 4th version, dubbed the V4 model. We used this model in our final implementation for the project due to its success. It utilized the data augmentation tools in Roboflow along with the altering of hyperparameters using the evolve hyperparameter optimization tool. This paired with our steadily increasing data set allowed for a fantastic model with few faults. We also decided to lessen the variance of each of the augmentations that Roboflow was using to expand the data set. For instance, changing the noise for 0% - 15%, to only allowing it to get up to 10%. Given we only have a single class to find, relying on smaller thresholds allowed the model to be able to distinctly find bb-8 in almost all scenarios. The only problem that we had with this model was a few pesky false positives but the problem was solved by increasing our confidence interval threshold when actually searching for bb-8. Our metrics for the V4 model are shown below.

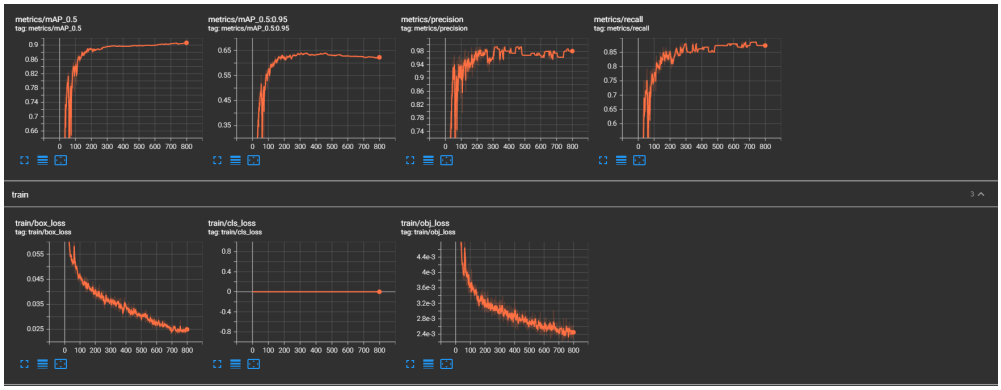


Figure 25: v4 Model TensorBoard

Pathfinding and Environment Exploration

As we continue to believe that, were we not to need to run an instance for each of 4-5 drones on a single laptop computer, that working in the world of three dimensions rather than two for the mapping and path planning of the drones, most of the discussion in this section will be in terms of 3D structures. All of these 3d structures have an analogous 2d structure that can be intuitively drawn from them, because there is a clean mapping between these structures, we've chosen to refrain from repeating the information here twice when this parallelism suffices.

SLAM

In order for a robot to effectively explore an environment it needs to know several things, namely: where it is, where it should go, and how to get there. The first piece is localization, the second requires having a map to make an informed decision, and the third requires a map as well in order to path around any obstacles. Both localization

and mapping functionality will be needed; because we are meant to explore an unknown environment, there is no pre-existing map for reference, and because the environment is gps denied, we need some way to keep track of our current position based on solely our robots previous positions, the commands it has been given, and the readings its sensors can provide from the environment. Trying to do both of these steps separately is nearly impossible, since without a map, there is certainly going to be drift in the inertial sensors, and without proper localization, there is no reference frame to properly place lidar scans where they belong in a map. Somehow these tasks must be performed in parallel such that they can both inform one another at every step along the way, a process referred to as SLAM.

SLAM stands for “Simultaneous Localization and Mapping” and can be used to create a map of a searched area as it is searched. This will prove an essential component of both efficient collaboration between drones when shared and of pathfinding even for an individual drone. The prograssive formation of a map serves multiple purposes. First, by generating a map, information about the environment is not lost once the robot navigates away from the area, and this information can be shared with both other robots and any other party which could benefit from the map information, such as a human team working alongside the robots which could be provided with a tactical map display including terrain, buildings, and any important coordinates. Another great benefit to SLAM is that by performing both localization and mapping, each improves the usefulness and accuracy of the other, resulting in a whole that is greater than the sum of its parts. A robot can correct for accumulated errors in its inertial sensors by comparing new laser readings with existing maps, and by having more accurate location data, the map building will also be more accurate.

Once robots are able to locate one another in each other’s maps, they can perform map merging. It is this map merging that will help with collaboration specifically, because once they know what one another has seen, they can avoid as many redundant scans as possible in pursuit of mapping out and searching the entire environment. There are several methods for performing map merging, but

fundamentally, the problem of multi-robot map building boils down to determining the initial position and orientation of each single robot. If we can just provide this to the robots on initialization the problem becomes much simpler, otherwise it becomes important to explore other options. Regardless of how they find each other, once robots have merged maps they can continue updating the merged map the same way their local map is updated without any of the extra costs of merging; the map merging only needs to happen once before maintaining it costs the same as a local map update.

The most promising algorithm I have found for merging maps of robots with unknown starting positions in real time is one which considers a robot exploring in the local map of another. As robots explore the environment, they are very likely to eventually explore some of the same areas, especially before actively coordinating. As robots explore, in order to perform their own local slam they build point clouds from the Lidar sensor on each scan to localize that scan inside their local map and ultimately continue expanding it. In addition, robots within range for communication can send messages containing the scan data as they fly along, and other robots can read that message and try to localize it within their own map. Two things can happen next, either the localization goes well, and now the second robot is confident that the first is exploring somewhere in a shared map space, or alternatively the localization does not go well, and no further action is taken for now. Assuming the first case with suspected merge opportunity, either it can be assumed valid and merged immediately, or it can be verified by the second robot pausing what it was doing to physically approach the first and only perform the merge if they can both observe one another directly.

The trade-off between speed and certainty weighs differently depending on the environment, for instance in an environment that has many common features, like a series of hallways or perhaps even a bunch of streets with similarly tall buildings around them, this method would be prone to false positives, the verification step would frequently be necessary, and so skipping it would lead to false shared maps. Conversely in a more diverse environment such as an apartment, the different rooms

are very distinguishable from one another, and there would be a low risk of false positives, and the verification would be unnecessary. For our application, although having known starting configurations would be ideal, if we do not have known starting positions I am in favor of taking this approach with verification before merges for the sake of robustness and transferability to other environments.

SLAM - How it Works

From the raw sensor data, the robot receives either a scan object or a point cloud object, depending on the type of sensor. A 3d Lidar sensor provides point clouds and a planar lidar sensor provides scans. Just like we worked with two different types of lidar sensors before coming to a final decision, we also had two different implementations of SLAM that work on related but distinct principles. In particular, a scan matching based approach and a graph SLAM approach were used in our 2D and 3D implementations respectively.

The approach we ultimately took by the end of the project was 2D scan matching SLAM, due to our computers not having enough power to properly run 4 independent drones with the 3d pathfinding, thus making it unnecessary to maintain a 3D map. Scan matching SLAM is less robust to dramatic changes in robot pose, but it is a relatively simple approach. This approach, as the name may suggest, attempts to align new scans to the map in place of classic odometry for localization. By assuming a relatively small change from one scan to the next, one can quickly iterate on estimates of the position and subsequently add the new scan information to the map based on that transform. Although the assumption of small transformations between scans helps keep the algorithm running quickly, it also causes a failure state if the robot translates or rotates faster than it is configured to expect. Similarly,. If there is a featureless hallway longer than the robot’s sensor range, or there are no obstacles within the range of the sensor, it will not detect any movement and the map could become corrupted by bad data as it becomes misaligned. In our scenario, this did not prove problematic because the outdoor urban environment is often full of many features which can be

used to help this method of localization. In figure 26 below we can see an image representing this process of creating odometry from a series of scans.

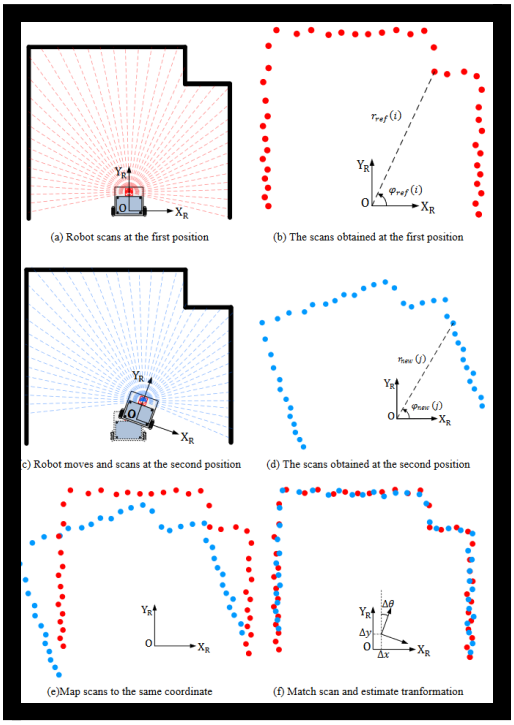


Figure 26: The process of creating odometry information through scan matching

The other approach, and the method we would have preferred to be able to use, is graph SLAM, more specifically the hdl_graph_slam implementation from koide3 and other contributors on github. This accepts 3D point clouds, robot odometry, IMU data, and GPS, though we did not make use of GPS ourselves, and combines many constraints on position formed by these pieces of information for a much more consistent localization. Graph SLAM, as the name implies, creates a graph with nodes and edges representing at each keyframe the robot's relative measurements to observed features, estimates on relative motion from odometry, the robot's estimated orientation, the angle of the floor, and loop closures. As is the case with most things, these estimates are noisy and imperfect, as such we consider the truth to be normally distributed with these estimates as the center points of those distributions. To make the world consistent, the graph SLAM algorithm then takes as its localization transform, that which has the highest probability when combining each of the probabilistic constraints on the graph. Figure 27 illustrates these ideas on a smaller scale.

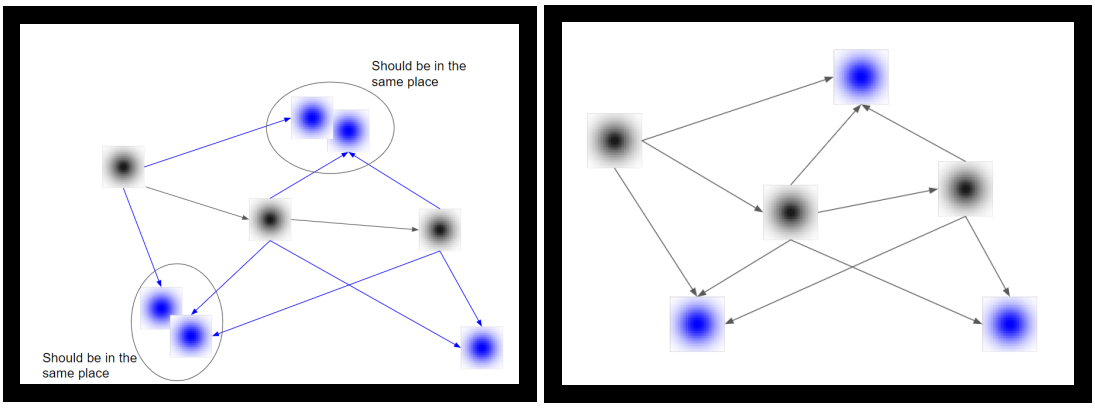


Figure 27: A simplified example graph with relative motion and relative measurement constraints in which there is an error between the first two scans (left) but the graph gets corrected to the most probable truth (right).

Occupancy Map

The native formats most suited to performing SLAM are not very good for performing other tasks on them, such as path planning. Path finding algorithms generally function over a graph of nodes representing locations, and edges representing which other nodes can be reached directly from a node along with weights which represent the cost of moving along a particular edge. This is a model that fits perfectly in two dimensions with a grid of squares that are either occupied or not, such as the classic maze. It also fits equally well in three dimensions with a grid of cubes, and instead of 8 adjacency edges each, there are 26 adjacency edges. It is necessary then to find a way to convert the point cloud map data from the SLAM process into an occupancy map of cubes. The first approach to this is a simple array of all locations in three dimensional space with an occupancy value. If the granularity is small enough and/or the map is large enough, this method runs into the issue of storing a very large amount of memory and if it turns out that a robot needs to explore somewhere outside of the initial map, it does not grow very flexibly. There is some advantage in its simplicity, but that can be outweighed by greater performance.

An alternative exists in the form of an octree, a tree data structure in which each node has either zero or eight children. This hierarchically represents 3d space, starting from some cube, cutting it in half each way into 8 smaller cubes, and repeating on each subsequent cube until some maximum depth. This poses several quite nice advantages over a basic grid. The map can be dynamically expanded, if the robot needs to explore some place outside of the initial region, the existing octree can become one of 8 nodes of a larger octree that includes the new space.

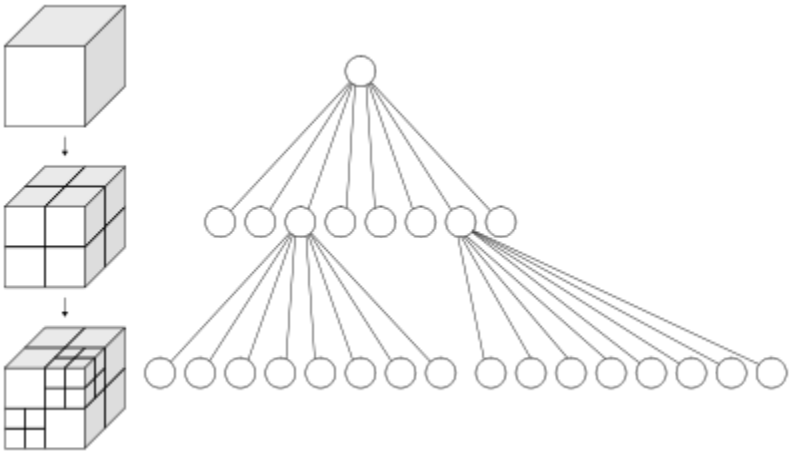


Figure 28: Recursive subdivision of a cube into octants and the corresponding octree.

Because it is hierarchical, Less information can be stored about areas that are homogenous; at the start of the search almost the entire map has not been observed yet, and so it can be represented by very large cubes across most of the map, and as the search continues, the free space which exists in large corridors between buildings and in even larger wide open spaces above the tops of buildings can likewise be represented by cubes much larger than the small granularity used to represent obstructions. This makes the map much more efficient in memory, which is a precious resource on a drone.

The granularity can also be an advantage when trying to speed up path finding. There should be no need to individually check thousands of small cubes within a corridor when all of them are known to be empty, a basic grid would need to still check them all, while an octree representation can indeed make far fewer checks by simply operating at a larger cube size. I'll expand on why this is important later, but tracing line of sight checks also takes advantage of the hierarchy to perform fewer checks much faster in the same way as path planning does.

Determining the Best Next Destination - Candidates

Based on the above, the robot knows where it exists in a map of the environment, consisting of obstacles it has observed, open spaces it has observed, and unknown spaces which it has not observed; it can answer the question, "where am I?". The next question it needs to answer is "where should I go?". Intuitively, the best place to go is where the robot can optimize its learning, to get the most new information to see as much of the space that has not yet been seen as possible. This just needs to now be put into more precise terms and then an implementation can be made.

There are already three classifications of cubes on the map: occupied, free, and unknown. The robot cannot exist in an occupied space, and so any unknown spaces may not be able to contain the robot either, while the robot certainly can be contained by free spaces. We don't know how far the robot will be able to see into the unknown space from wherever it goes, but let's assume for now that we have some constant estimation. Then we can calculate an estimated new observation volume based on how much of the field of view of the sensor is occupied by unobserved regions, as the volume of the frustum formed by cutting the already seen space off of the field of view looking into the unknown space. This volume is optimized if inside of a large unknown region, and minimized in a large observed region. The odds of actually being able to exist in a space are optimized at certainty in a large observed space, and quite low in

an unobserved space that both may be occupied and may have walls in the way of reaching it. Thus, on the constraint of being able to reach a location for sure, the most information can be gathered by being on the boundary between unknown and free spaces. Any further into unknown space might be inaccessible, and any further away from unknown space is shrinking the frustum of new information for no benefit.

The free volumes on that boundary are what I'll from here on be referring to as frontiers. Any best destination is necessarily restricted to one of these frontiers, but there are multiple frontiers to choose from. Finding and selecting from frontiers to make the next goal, especially in an amount of time befitting a drone, is not trivial, and that will be the subject of this section.

Detecting frontiers can be as simple as looking through all cubes in the map, and for each cube if it fits the definition of a frontier recording it. Doing this whole thing every time the drone needs to detect a frontier would however also be dreadfully inefficient. Not only is it unlikely that a robot will care about a frontier all the way on the other end of the map from where it is now, it also doesn't need to update its record of frontiers in areas where it isn't getting new information. A frontier the robot saw 10 seconds ago that it chose not to explore is still in the same place, it would be redundant to re-detect it. Surely there are ways to make this better.

A potential solution to redundancy in searching for frontiers in the same places repeatedly and on the parts of the map nowhere nearby is to keep a list of the frontiers across time. To maintain the local list, keep track of the new observations at each cycle, and only check the union of those new observations and the previous frontiers adjacent to them for updates to the list of frontiers. As the local frontiers are found or explored beyond, that is communicated to others to form a global list of frontiers much the same way as the map is shared. The global list of frontiers alongside the shared map of the environment allow a sort of implicit collaboration between robots, where if

another is exploring an area, there is a greater cost and lesser benefit to chasing down the same path as it inherent to selection of a goal, but also the benefit of using the knowledge picked up by others in the history of their frontier discoveries not yet explored allows a robot to already have goals once its local frontiers run out.

The detection can also be sped up by searching at a different cube size. Because the map representation is hierarchical, we can choose to look at a different granularity a few steps up from the lowest to have exponentially fewer cubes to examine. Even if frontiers are desired at a lower level, we could choose to only perform the deeper search for frontiers within the large cube frontiers, still saving time. There are actually advantages to using larger cubes for frontier detection aside from speed of detection.

Simply detecting frontiers and comparing them directly in that form has its own problems too, notably heavy redundancy in evaluation and selection of which to ultimately path to. Frontiers will tend to appear in large clusters almost always, and the path to each frontier in a cluster will be almost the same, but all will be calculated fully regardless if nothing is done about it. If those clusters can be detected even partially then the following step of evaluation can be performed as if each cluster were only a single frontier space, drastically reducing the time it takes. I've already hinted at clustering by larger cube sizes, which comes for free with another performance increase. From within those larger cubes we can just take the center and go there; there is no need for small granularity precision since by the time we get there, we'll have enough new information to want a new target anyway, it does a good job of approximating the frontier.

To get more clustering after that, frontiers that are within some prescribed distance, no greater than how far the robot can expect to see through unknown space, of one another can be merged to further reduce the frontiers to evaluate into a very tractable number, as small as just 2 if isolated in a long hallway for instance.

The number of clusters can in most cases be reduced even further for no cost by only looking at local frontiers if there are any. This would encourage robots to focus on continuing to follow a path of exploration they are already taking, which is likely what they will want to do anyway. Thus, they often will not even need to compare their nearby goal to the far away frontiers. Once there are no local frontiers left, then the robot will have to poll the global list of frontiers and start searching at those, but because the list of local frontiers was empty to get there, it does not double search anything.

Determining the Best Next Destination - Choosing

Once there is a list of several frontier candidates, it is time to choose between them, and assign robots to them. First, let's examine the case of a single robot, and then show how it can grow to multiple robots and encourage spreading out.

Fundamentally this choice comes to a ranking of prospects - costs. The prospects can either be considered as some constant everywhere, or guessed by environmental factors, taking advantage of common features in a specific type of environment such as turning a corner in a street revealing another long street, or going through a doorway probably revealing a large room. I believe that would be at least a small improvement, but also likely an unnecessary complication, and this could be a stretch goal if we get through all of our planned tasks and still want a way to improve the search by a little bit. The other piece, cost, can be found by running pathfinding from the current position to the frontier candidate, and using the distance. This results in simply going towards the nearest frontier.

Once there are multiple collaborating robots, let's assume they are given destinations iteratively. Then they can be encouraged to look in different places by first choosing as in the single robot case, and then all others choosing a frontier after putting a discount on the prospects of frontiers near destinations claimed by other robots according to the distance away from them. Then, as robots conclude a target path and begin looking for a new trajectory, they remember the locations of all other robots' destinations and perform the discounting as though they are the last to choose, because at that moment they are.

Collision avoidance

Methods for avoiding collisions can be split into two primary categories, which are probably best applied together in real world situations. When there is a map of the environment being maneuvered through, the path generated between the present location and the destination can be formed in such a way as to guarantee a certain distance between the line of the robot's center's path and any known obstructions. In an ideal world this proactive form of avoidance would be sufficient by itself, however the real world is noisy, sensors are imperfect, and mechanical parts don't move exactly the way they are meant to. A path that just barely misses every obstacle in theory, is likely to run into at least some of them if the simulation accounts for those imperfections with artificial noise. Other moving actors in the scenario need to also be avoided, though these cannot be mapped.

There are several types of reactive collision avoidance, differing from one another in how they detect possible collisions and in how they avoid the possible collisions after detecting them. What was discussed by a previous senior design group on this project

for their ground based robot was to use what I'll call whiskers. They check distances in three different directions generally in front of the robot: straight, slightly left, and slightly right. If one of these whiskers would detect something in the way of the robot along the path they sent it to follow, it would turn in the direction away from it and keep moving that way until it can turn back toward the goal without re-triggering the whisker. Inspired by animals with whiskers, this method doesn't actually have whiskers, but seeks to imitate them virtually with lidar sensor readings, with the advantage of being a very fast way to react to possible collisions and provides a very simple method of turning away from the obstacles without needing to really slow down. On the other hand, this has the limitations of only noticing possible collisions with things in front of the robot, while if something would hit the robot from the side or from behind, the robot would not see it and try to protect itself from the collision. As I understand it from their design documentation, this also feels that it might be somewhat rigid in the reactions taken by just turning away. If two drones were to be on a collision course, but one's flying next to a wall, that drone would not be able to turn more towards the wall without also crashing into it.

We prefer a slightly modified but similar concept, based on the lidar sensor we plan to include on the robot, as well as some ideas found in physics. Whereas we referred to the previous method as whiskers, this one could be thought of as a bubble. Rather than checking only in front of the robot, checking in all directions for obstacles within some relatively close distance. Then the resulting readings constrained to at most that radius r, form a bubble of radius r squished a bit from the directions with obstacles. Then, the direction the robot intended to go to reach the next waypoint on its path is modified by applying an imagined force on the robot proportional to the distance that the sphere is compressed in that direction. If an object is just barely inside the bubble, but not immediately at risk of hitting the bounding sphere, then the drone will only take a small reaction on its path away from it, but as something gets extremely close to hitting the robot, the robot experiences a repulsive force of incredibly large magnitude. Where the whiskers can detect and inform movement, the bubble is like surrounding the robot with a bunch of springs which passively act to push it away from anything it approaches too closely.

The advantage of the bubble over the whiskers method is that it makes reactions that are very tame in most cases assuming the initial path finding tries to give enough space but can make an extreme reaction if it becomes needed, this can also react well to the supposed pinching against a wall type of collision by forcing the robot further from the wall to take a bigger turn to avoid the crash than the one near the wall. On the other hand, it needs to check a greater number of directions to form its bubble, which comes at some computational cost. Thankfully, those sensor readings are already necessary regardless, and the cost of comparing a set of them to a constant and performing some vector addition is quite small and of little concern.

Ultimately the solution implemented is effectively the bubble, but rather than inflating the robot's outline, we inflate the environment, which is already in a format which represents costs of travel and doesn't need to be constantly changed while the robot moves. At its core, this gives the same increasing costs to locations as the robot gets closer to hitting obstacles as the bubble method we came up with, with the benefit of already having an implementation in the existing ROS navigation stack.

Pathfinding

After determining where it wants to go, the robot must figure out how to get there. To start, an overview of pathfinding algorithms, and then those that were explored as options. Fundamentally, pathfinding algorithms function over a graph of some defined space, defined by a series of locations as nodes and connections between those locations as edges, often weighted with some distance or other cost. A pathfinding algorithm must somehow explore that region in order to find some connection between two points, and furthermore do so while incurring the least cost of travel.

The solution we have settled on for our final design is A* for global planning with the smoothing happening naturally as this gets translated into physical movement of the drones. During Senior Design I, much thought was put into making sure paths were especially tight, and not constrained to the grid, but as it turns out the relatively fine grain grid, imperfect flight of a drone, and especially the separation of global and local path planning make that level of concern unnecessary. Despite not continuing along this path, we believe it is still beneficial to keep the discussion of Theta* nearer the end of this section as it was a notable chunk of research and may prove beneficial to future iteration of the project.

Now, let's begin an overview of path finding algorithms on a small journey to rediscover A* and iterations upon A*, and understand the benefits provided by these. A well known algorithm is Dijkstra's, which when run over a graph does the following:

1. Label all nodes as unvisited, making a set of them.

2. Setting the cost to reach the starting node to zero, and all others to some infinity. Mark the initial node as current.

3. Consider the current node. For each connected node, compare the present cost to reach with the cost to reach the current node, plus the edge cost from the current node to that connected node, set the cost of the other node equal to the lesser of these.

4. After considering all neighbors, mark the current node as visited, remove it from the unvisited set.

5. If the destination has been visited or if there is no way to reach the destination, terminate.

6. Select the unvisited node with the smallest cost, mark it as the current node, and then return to step 3.

Another algorithm, and a naive one, is a greedy algorithm which explores comparably very few nodes. It ignores the accumulated cost and greedily seeks to minimize some heuristic that represents the distance from the goal. It ultimately does a significant amount less work, in exchange for giving up the promise of finding a best path. This algorithm could look like this:

1. Mark the start as visited, placing it into a queue.

2. Consider the node in the queue with the least distance to the goal as the current node, remove it from the queue.

3. For each unvisited neighbor of the current node, mark it as visited and add it to the queue.

4. If the goal has been visited, or cannot be visited, terminate.

5. Repeat from step 2.

Both of these looked at so far have some flaw that holds them back, search time and path length respectively, and there is an alternative. A* is the standard go to pathfinding algorithm, and the first we looked at more seriously. A* Combines the best of Dijkstra's algorithm and a greedy best first search approach, to find something in the

middle which can guarantee finding a shortest path, and do so by searching fewer nodes than Dijsktra's. Rather than focussing completely on either the distance from the start, or the distance from the goal, A* is informed by both, favoring vertices which approach the goal via an estimated cost, but also keeping in mind the actual cost of the path so far assembled.

A* in its basic form is an improvement to be sure, but it remains restricted to a grid, which leaves zig-zaggy paths along diagonal directions, which are unpleasant to look at and not ideal for a flight path in the real world. While seeking a solution to this issue I came across Theta*, which is an any-angle pathfinding algorithm based on A*, and which looks quite promising as the solution to be taken for the final product. In a graph that allows calculation of line of sight, such as the one we will be using for this application, Theta* is able to take advantage of that to smooth out a path as it is generated. The difference comes during the step of adjusting the cost value as a new node is updated; while A* only considers the cost from the current node to the new one plus the existing cost of the current node, Theta* considers that and in addition considers the path from the parent of the current node to the new node directly, without routing through the current node, but only if it passes a line of sight check with the parent.

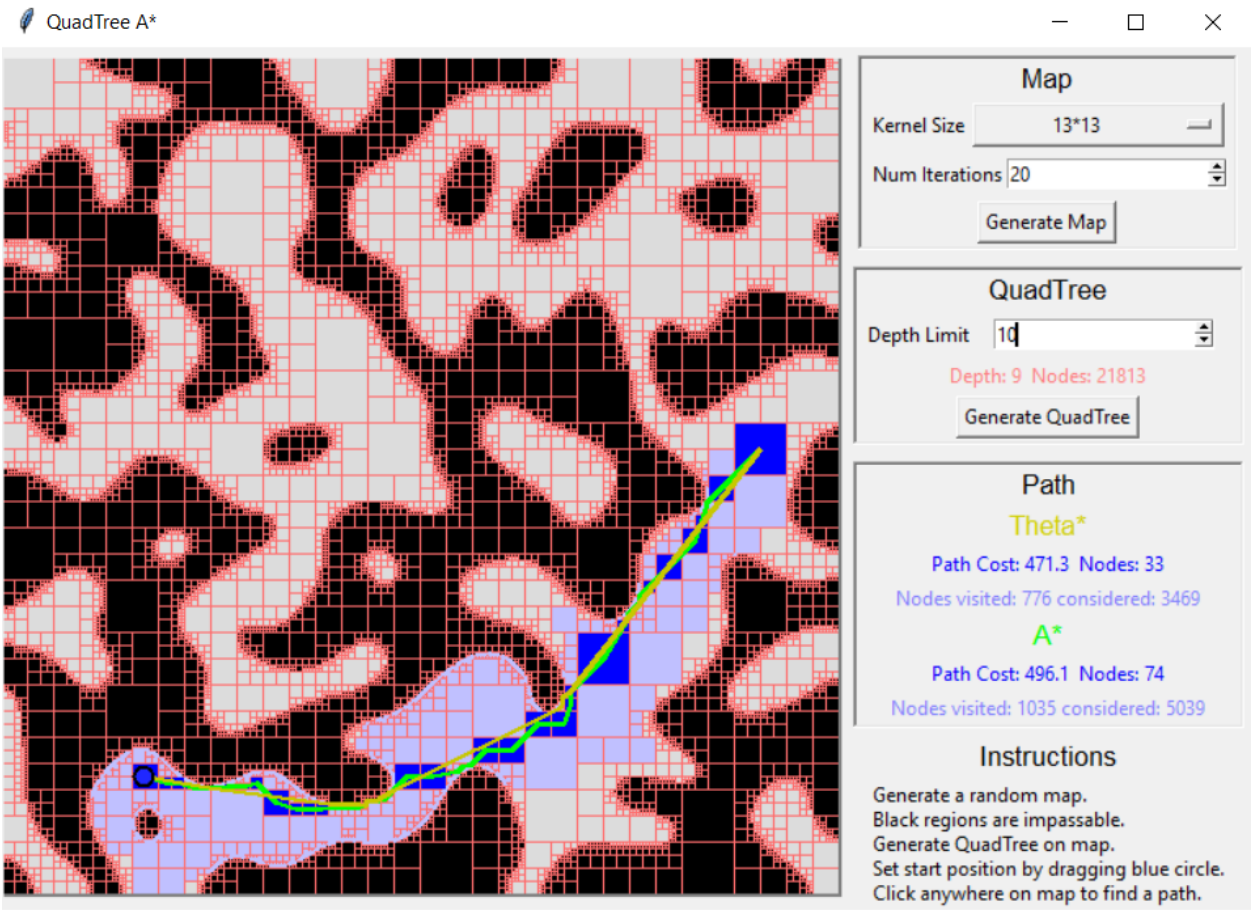


Figure 29: Interactive comparison of A and Theta**

In this figure we see a comparison of the paths found via A*(green) and Theta*(yellow) on a two dimensional graph. Note how the true shortest path is obtained by breaking away from the strict grid based approach. Theta* visits and considers fewer nodes than A*, and results in an ultimately shorter path found.

A different alteration of A* I came across was D*, a variation that seeks to optimize for adapting to unexplored environments or environments which are prone to change. Although this seems like something that we should look at deeper at a glance, due to the other details of our exploration strategy this would be redundant. More specifically,

all locations to which we will command the robot to path are locations which have already been observed by the onboard lidar sensor at some point, and so we can be confident in setting and following a deterministic course to the goal. Although I found looking into this algorithm intriguing, I don't believe it will be very applicable to our particular solution.

Up to this point, the discussion of pathfinding algorithms has been focussed on two dimensional examples, but does this all expand to a three dimensional graph formed by cells and adjacencies in the octree map discussed in previous sections? Indeed it does, in fact the graph itself has no inherent notion of dimensionality, it is simply an abstraction of a map. In two dimensions with a grid of squares, adjacencies can be recorded by 8 edges, one for each cardinal direction, and one for each diagonal direction halfway between two cardinal directions. The only difference when adding a third dimension is that the graph's nodes will each have 26 edges, in the middle layer horizontally the same connections as in two dimensions, and then a full 3x3 square above and below for 8 + 9 + 9 = 26 connections. The exact same set of algorithms can be run on this more populated and more connected graph with no loss of generality, so there are no concerns there, aside from sufficiently large areas becoming potentially quite time consuming to search, since the number of nodes visited grows much more rapidly as a function of the distance from the goal than when there were only 8 edges per node, and by performing theta* instead of A* we also must perform line of sight checks throughout.

Lets look back through to the properties of the octree and how that might alleviate these troubles. The number of nodes exploding more quickly as more nodes get visited; Most nodes being visited would be in large open spaces, which an octree is quite good at reducing into a much smaller number of larger volumes. We must perform line of sign checks; Octrees are particularly good at calculating collisions efficiently, due to their structure, and this includes ray casts. This benefit serves as much to reinforce the decision to use an octree representation as it does to assuage

doubts about the tractability of three dimensional pathfinding, but accomplishes both nonetheless.

How Octomap Works

A standard point cloud structure contains information about where obstacles are, and this could be, and has been many times, directly translated into either a standard or hierarchical occupancy map. This is not an especially complicated task, but is also not able to distinguish between free and unobserved space, which for a robot seeking to explore only areas that it hasn't seen yet, is a very useful and important distinction to make. The best thing I have been able to find to solve this problem is Octomap. Octomap exists as a standalone package with very permissive licensing, and quite helpfully also provides packages to easily integrate with ROS.

A special property of the Octomap is that it takes both the point clouds and the positions they were scanned from, and explicitly models white space as a state of nodes in the octree internally for a total of three states by implicitly representing unknown space by the absence of a node. It is probabilistic, which enables it to be efficiently updatable with new readings which may have minor conflicts with the map so far produced. Octomap also enables multi-resolution querying by giving inner tree nodes a probability value equal to the highest probability of its children, which is especially useful in cases of needing to do anything quickly on the fly or in real time, but also whenever there is no need for high resolution and a higher speed of querying is useful. Lossless compression is inherent to the use of a hierarchical structure by pruning children if they are all identical in value, something useful for memory efficiency and for sharing of maps between robots over a network efficiently.

The probabilistic updating model of occupancy in Octomap is as a recursive bayes filter, efficiently updated via log-odds as a replacement for probability in the recursive bayes filter. In essence the true state of the robot in time, in this case the true position from which a provided point cloud was scanned, is modeled by a hidden or unobserved markov process, and the lidar scan point clouds are partial observations of that hidden state. The more confident the robot can be about its current position, the more of an influence that scan will have on the map, but also the more recent a scan is, the stronger its influence on the map. The map can't be completely certain of what the environment looks like, but it can certainly accept that it won't be perfect and model the likelihood of its guesses' correctness. A recursive bayes filter is actually made easier to compute and easier to update by using log odds instead of the natural probability estimates, and more stable with high uncertainty as a bonus on top of that. The formulas used, as per the Octomap presentation at ROS con in 2013 (Armin Hornung, 3D Mapping with OctoMap https://youtu.be/RRP29VnY8go?t=575), are these:

Occupancy modeled as binary bayes filter: 1+1-P(n | zt)P(n | zt)1-P(n | z1:t-1)P(n | z1:t-1)P(n)1-P(n)-1

Efficient update using log odds: L(n | z1:t) = L(n | z1:t-1) + L(n | zt)

To update according to readings, Octomap takes as input a point cloud and the origin of the sensor which produced the point cloud readings. For each point, a ray is cast from the sensor position to the point; the endpoint is marked as occupied, increasing the probability that the voxel is occupied, and other spaces along that ray are marked as free, decreasing the probability that the voxel is occupied. This is the most computationally expensive part of updating the octomap, nonetheless it is worth doing for the ability to explicitly model free space.

Octomap packages provide efficient methods to access as well as to create the map data. There are functions to iterate over leaf nodes, iterate over voxels in a bounding box, find all leaves which intersect a ray, find the leaf node containing a coordinate, create ROS messages, and much more available in the documentation for Octomap here: https://octomap.github.io/octomap/doc/.

As great as Octomap is for creating an efficient occupancy map, it cannot be used entirely by itself with a sensor for good results. Octomap is not a SLAM solution, and so it requires some implementation of SLAM and/or an incredibly robust odometry system, to perform localization and provide poses for octomap to use. Octomap is particularly useful for detecting collisions and is used for that purpose in the Moveit! Package for ROS, which integrates Octomap and is designed to plan motions for robotics manipulators, and which some have been able to set up to work for planning paths for a drone by having an “arm” without collisions and a drone at the end which does have collisions.

This abstracted information is all well and good, but it is useful to also understand how the volumes of space are actually separated into the nodes of a tree. It is easier to work in and view examples in two dimensions, so we have a program built in python which randomly produces a map of some region, and then creates a quadtree, the two dimensional equivalent to the three dimensional octree, in order to prototype pathfinding. This tool can also serve to help show the process of constructing the tree from the readings of an environment, assuming this random 2d environment is fully observed.

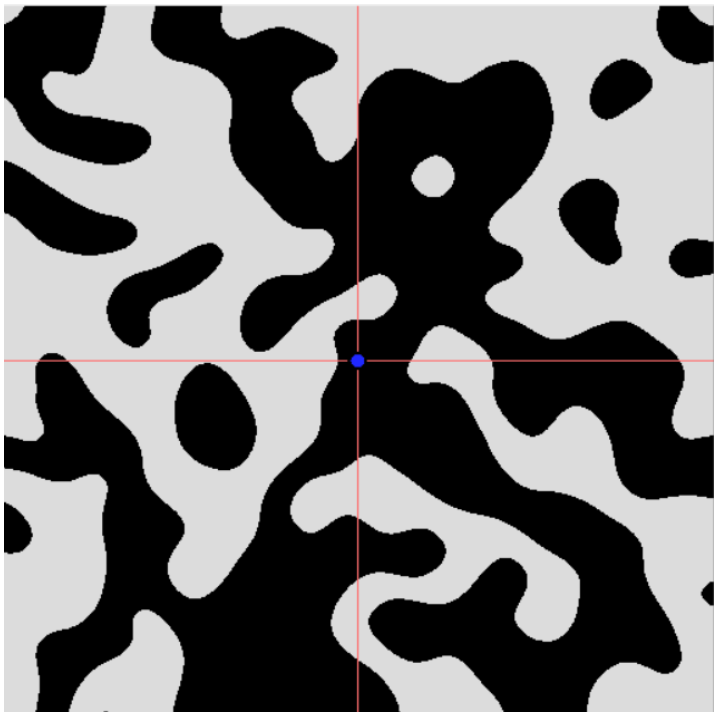


Figure 30: Step 1 of constructing a quadtree

In figure 30 we see the first step of dividing the region. We looked at the whole image, and concluded that it is not homogenous, and so it was split into 4 regions. We must then repeat this on each quadrant recursively, stopping once we reach a maximum depth or a region is homogenous, either all of a region's children are occupied or all of its children are unoccupied. In figure 32, we can see the result of working to a depth of 6 on the same region, and now we can start to see clearly that some homogeneous regions are saving memory compared to a full grid of equally small sized squares. And finally at a depth of 9 where the resolution of the quadtree is the same as the resolution of the image (figure 31), the quadtree lossless compression of the image can be seen by observing the size of the squares drawn in red.

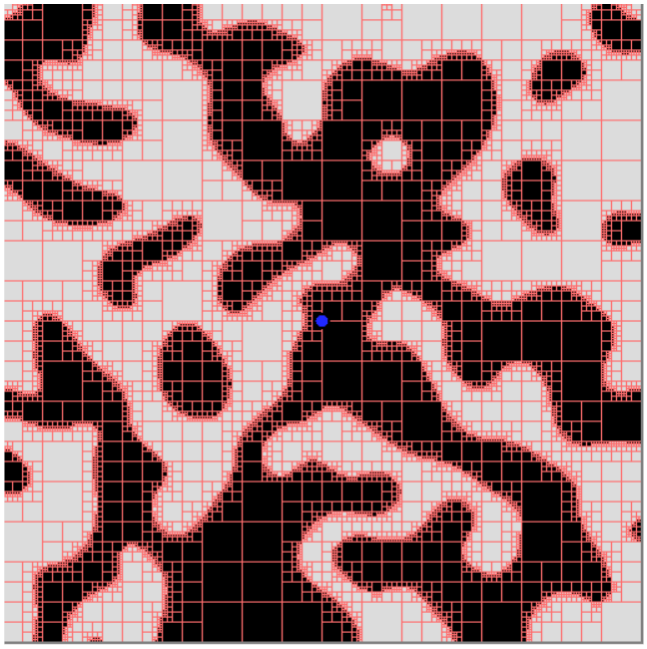


Figure 31: Quadtree 1

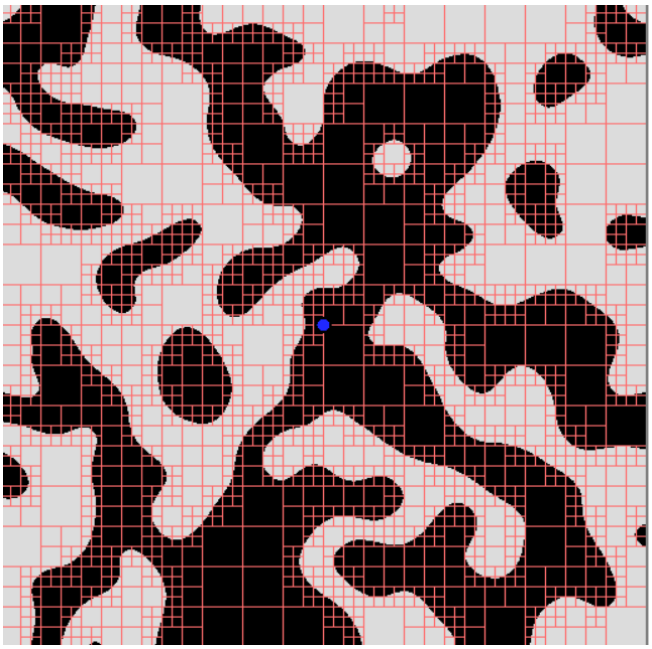


Figure 32: Quadtree 2

It has been mentioned several times above that octomap, and thus the octree data structure, allows efficient collision detection and raycasting; the same applies to the quadtree. Let's then explain what makes it so efficient by comparison to a basic grid. Before we can detect an intersection set with a quadtree, we must first detect a ray to line segment intersection. The equation for a ray can be defined by two vectors, one from the origin to the start point of the ray, and another representing the direction of the ray. If (x1, y1) is the ray origin, (x2, y2) is the ray direction, and (x3, y3) and (x4, y4) represent the endpoints of a line segment, algebraic manipulation can find where these would intersect if they went infinitely off in both directions as true lines, in terms of how far along the ray and how far along the line segment they would intersect. Then, if it hits on both the front side of the ray and between the endpoints of the line segment, then there is a collision. Collision with a square can be constructed by 4 line intersections for each side of the square. Finally intersecting a ray with a quadtree is performed recursively like so:

1. Check if the ray intersects the full quadtree node's bounding box
    a. If it hits, keep going, if it misses go back and stop looking.
2. Check whether or not the node has children
    a. If so, calculate intersection of the ray with each child subtree
        i. Record voxels intersected within each child subtree
    b. If not, record that this voxel intersects the ray
3. Return to the caller the list of voxels which intersect the ray.

To perform this in an octree follows that same algorithm as described for a quadtree, only with cubes instead of squares, constructed of simple ray to plane intersections instead of between rays and lines, really the process is quite closely related. This is a trend which extends to almost all operations on quadtrees and octrees. Perhaps it is also useful to understand how to find neighbors to a node, in order to apply pathfinding algorithms. Just like earlier, let's describe the algorithm on a quadtree and understand that it extends simply to octrees by replacing the 8 directions with 26 directions, adding up, down, and all the new intermediate directions.

1. For each cardinal direction {N, E, S, W}, and each direction directly between adjacent pairs, {NE, SE, SW, NW}, find the neighbor(s) in that direction and record them.
    a. Check for a neighbor in that direction as a sibling node

        i. If this is found, then it's the only one, so continue on to the next direction.

    b. Check higher levels up in the tree for a node with a sibling in the current direction, until eventually finding one.

        i. Descend the tree first down that sibling node and then in the opposite direction, until either reaching the same level of the tree as the initial node or reaching a leaf, whichever comes first.

            1. If a leaf was reached first, then the node found adjacent is of lower resolution, and is a larger square. Then it's the only one, so continue on to the next direction.

            2. If the node has no children, it is the same size, then it's the only one, so continue on to the next direction.

            3. Otherwise it does have children, so recursively follow each child further towards the node whose neighbors are being found until reaching leaves. Record those leaf nodes, which are numerous because they are smaller, in greater resolution, than the initial node.

2. Return the list of neighbors from all directions to the caller.

Development Considerations

Research will make up a large majority of the team's effort early in the project. A great starting point for our research is the senior design document found on the public Github repository of the previous senior design team.

The Ancestor's Project

The team has taken to referring to the previous senior design team as "The Ancestors" because it is much more concise than saying "the previous senior design team." We had initially intended to run their project on our local machines in order to learn from it before trying to expand on it, but this seems like it may be impossible, and might not even be completely necessary. While the Ancestors did provide a setup guide for their project on their public Github repository, ROS is an ever changing environment, and the project made use of turtlebot3, a popular low cost robot typically used by beginners to learn ROS. Turtlebot3 also has publicly available object recognition, pathfinding and mapping, and Gazebo simulation support, but ROS is in constant development and somewhere in the chaos, the dependencies for the specific package of turtlebot3 that the Ancestors used broke down. While it is likely possible to repair this, it is simply more work than it is worth, as the biggest thing we expect to use from the Ancestors is their urban environment to test our swarm in. Patrick Baur, the project lead for the project, even suggested that we learn Gazebo and ROS from a source besides their github repository.

Disabling the Target - Theoretical discussion

Lockheed Martin is the sponsor for this project, and they are a well known and respected defense contractor that, among other things, have made all kinds of world class weapons systems. The problem statement of this project mentions “disabling” the target of our search, and even though the physics of drone flight and weapons systems are outside the scope of the project because the main focus is on swarm search behavior, it is worthwhile to consider this sort of “flavor text” included in our problem statement. Ultimately, this project is so far removed from a real world combat scenario that all we can really consider at this stage is the pros and cons of several different

The first and most obvious option is a conventional firearm. The advantages are that firearms are a tried and true means of destroying things, and with a large enough caliber it won’t matter how much armor BB-8 has protecting his adorable sci-fi electronic insides. Suggesting mounting a gun on an autonomous robot meant to be used in an urban environment often alarms people on an emotional level due to the trope of murder-happy robots being common in dystopian science fiction settings. However, there are some valid, non-emotionally driven reasons why this may not be a good option. As anyone who has been to a range before knows, one of the four fundamental rules of gun safety is to be aware of your target and what is behind it, and currently, our object detection is only suited for detecting BB-8, so on a strict level, our system is incapable of adhering to this safety rule. This means that missing shots, shots ricocheting off things in the environment, and shots over penetrating BB-8 and continuing to travel through the evicorated droid’s hull could all be safety concerns to people and assets nearby. While it is true that our swarm is running in a simulated urban environment, it is possible that whatever environment is being simulated is abandoned and this isn’t an issue anymore. Other options that fit into the “conventional weaponry” category are mortars or other indirect fire weapons on a non-flying drone that hits a location described by the swarm’s mapping of the environment and human companions to the swarm that have their own means of disabling threats. Explosives

carry the same collateral damage issues that we had in the first place, and while US service members are the best marksmen the world has to offer, utilizing their skills exposes those soldiers to the hostile drone we are trying to eliminate.

Nets could be used in order to entangle the hostile target drone and prevent its movement. In Tokyo the police department has used drones with large nets to intercept smaller drones and bring them back to an officer for later inspection. This method works largely because the net gets tangled in the rotors of the malicious drone preventing them from spinning to generate thrust. The biggest benefit of a net is that if you accidentally miss and net a bystander, they may be minorly annoyed for a minute while they remove the net, but will be otherwise unharmed. A downside of the net is that depending on what the hostile drone was doing, simply immobiling it may not be enough to truly disable it. Additionally, BB-8 is ground based, and could likely keep rolling despite being tangled in a net, and he is the size of a small to medium sized dog and much heavier, meaning a drone likely cannot haul him back to a police officer the same way the police drone in Tokyo do.

Iterating on the idea of immobilization, and taking inspiration from *Home Alone* and *The Incredibles,* we could use a bucket of magnets to act as caltrops that will magnetically adhere to BB-8 as he rolls near them, or as they are dropped directly on him. Then, these magnetic blocks would make it difficult to roll when they made contact to the ground, much like how it requires more force to get a hand truck over a curb or step. If this alone is not enough to immobilize BB-8, then compressed gas with a stretchy membrane could be added to increase the size of the protrusion after the magnet sticks to the target, much like how Mr. Incredible is immobilized by the black expanding sticky balls in the scene where he learns the fate of the other Supers. This has similar pros and cons as the previous net method, and is more suited for a ground based target that is made of metal, like BB-8.

Radio frequency jamming is another option, and has already been implemented in real life. By sending a powerful wave of radio frequencies, you can disrupt a robot's communications which would take down any instruction the robot receives remotely as well as any global navigation satellite communications. This method is nice because it cannot hurt humans directly, and has been proven effective already. However, the downside to this method is that if our robots are autonomous and immune to radio bursts, our target likely is too. Additionally, the FCC closely monitors and regulates radio jamming technology, so if you do choose to go with this in a future project, be sure to follow rules regarding this.

The last option is a higher energy beam, like a microwave burst. This is what the previous senior design team decided was best because it has been used in Tokyo before and was recommended by the sponsor. The higher energy not only disables communications as with radio jamming, but also is powerful enough to disrupt and damage the target's internal systems, circuitry, and sensors. Without their seniors, a hostile robot is likely made harmless.

In terms of this project, the simulation will simply end when the swarm has found BB-8, leaving how exactly BB-8 is exterminated to the imagination of the viewer. It will be up to whoever continues this project to determine which option best suits their needs.

Setting up the Environment for our simulation

The first thing we need to do is set up a Linux environment to run the Gazebo simulation. We have a few options for this, and pursued several of them. One option is setting up a virtual machine running Ubuntu 20.04 LTS because Gazebo's windows

application is still in development. After some short research into the popular hypervisor software options, the three frontrunners were VirtualBox, Hyper-V, and VMware Workstation Player. While VirtualBox is what the Ancestors used and has the most beginner friendly user interface, and Hyper-V comes default with windows only requiring enabling and is great for remote desktop hosting, it appears that VMware has the best hardware passthrough and allows the most video ram to be allocated to the guest machine of the three option, making it the best choice for our purposes.

While VMware's hardware passthrough tools are great and leads to a smooth experience in normal use cases, we found that after some testing it does get bogged down easily when rendering the Ancestor's urban environment, which could mean that the GPU passthrough is simply not good enough for GPU intensive tasks like AI, image recognition, and graphics rendering. Another option is to either dual boot Ubuntu with windows, live boot Ubuntu off of a USB, or simply wipe a team member's current operating system in favor of Ubuntu. All of these options will provide better performance than a virtual machine will.

Live booting is particularly appealing, as it would avoid the brutal sacrifice of a team member's entire operating system as well as the potential to take the environment with us to other computers that potentially have better hardware than the team's personal computers. In particular, the senior design lab has 1080ti's in their computers, and while all team members have competent computers for gaming and recreational purposes, most of the team's personal computers are less powerful than the senior design lab's computers. Due to these benefits, Live booting off a usb drive was the next option explored. I used a 64 gb usb 3.0 and installed linux on that drive. Doing so took an extraordinarily long time, which foreshadowed the four hours it took to install Gazebo and its dependencies. Additionally, everything took a long time to open and load. This is reflective of the lower data transfer rate of a usb 3.0 port as compared to a sata port. Due to these soul crushingly long load times, booting Linux off of an external drive is not appropriate for the needs of this project.

The last two options remaining are dual booting linux off an internal drive, or replacing a team member's current operating system in a brutal and tribal sacrifice of their previous data. Because all team members are hesitant to commit to this savage sacrifice, dual booting was the next option to explore. In general, the way you do this is to create a reasonably sized partition on one of your internal drives, download the .iso file of the operating system you wish to dual boot, flash that .iso file onto a usb or cd, go through your bios to boot to your installation medium, follow the installation wizard on your installation medium, then restart the computer, being sure to boot to the drive on which you installed the new operating system's boot loader. Video walkthroughs for this process are readily available on youtube, but there are a few pitfalls that I should address that I faced that are not covered by most such guides. The first is a consequence of having more than just one huge drive on your computer, as most people with enough disk space that they would be able to dual boot two operating systems likely have more than one drive. Partway through the installation wizard, you will be asked where to install Ubuntu and where to install the GRUB boot loader. You obviously make the partition you made earlier be your primary Ubuntu partition, but the boot loader will only be happy installing on whatever partition is named "/dev/sda" which also means that this is the drive to boot to when you want to boot Ubuntu, even if the main partition is on another drive. However, I also found that reinstalling Ubuntu automatically made an additional partition off of your main partition with the boot loader, making that drive the drive to boot to if you want to run Ubuntu, and windows clears the old boot loader the next time you boot to windows. Another major pitfall of dual booting, or more accurately of using Ubuntu at all, is the problem of graphics drivers. If you have a Nvidia graphics card and you are installing Ubuntu, in the wizard you want to leave the box for proprietary drivers unchecked, because the proprietary Nvidia graphics driver leads to a "purple/black screen of death" and is unable to render anything. Leaving that box unchecked leaves you with the open source Nvidia graphics driver which actually works.

With that, we have determined that running Ubuntu native off an internal drive connected to the motherboard by a sata or faster connection are the only feasible

options for our needs. However, while using all three of these options, I took the liberty of running the Ubuntu benchmark software “HardInfo” and compiling the results below. All tests were run on the same computer hardware, except for the manner in which Ubuntu was being run. One was in a virtual machine, another booted off a usb, and a third booted off an internal M.2 SSD. I have presented the benchmarking data in both table form and in graph form.

Linux Benchmarking

File Edit View Insert Format Data Tools Add-ons Help Last edit was seconds a

100% $ % .0 .00 123 Default (Ari... 10

H10 fx

	A	B	C	D	E	F
1		CPU Fibonacci	CPU Zlib	FPU FFT	FPU Raytracing	GPU Drawing
2	VM	0.47	0.4	1.28	1.83	5126.55
3	USB	0.45	1.58	0.61	1.75	5643.14
4	Dual Boot	0.44	1.6	0.6	1.71	5725.85
5						
6		lower better	higher better	lower better	lower better	higher better

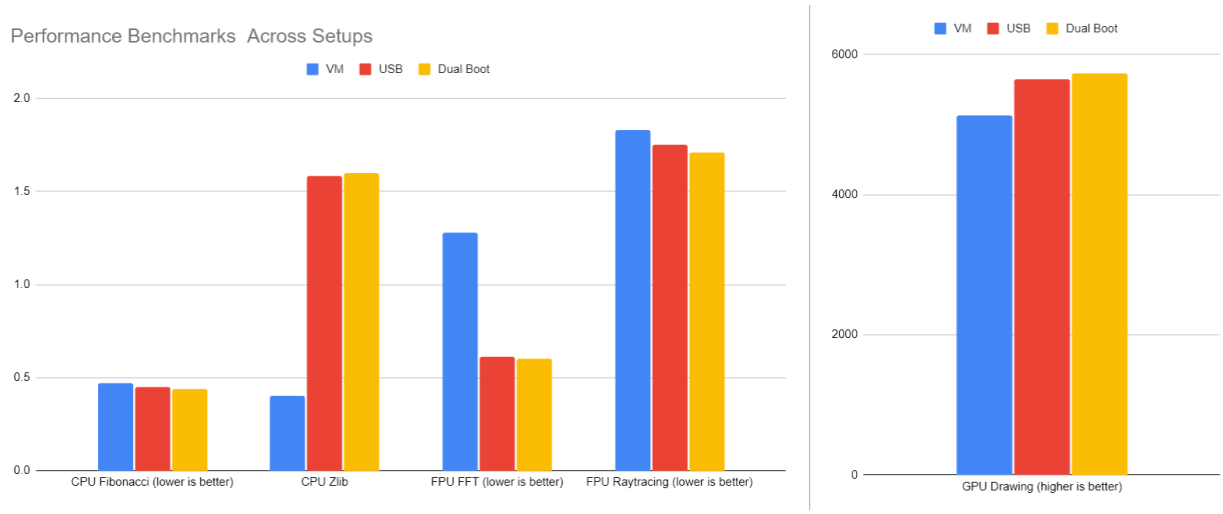


Figure 33: Screenshots from google sheets showing benchmarking data about virtual machine, USB boot, and dual boot in both table and graph form

Ultimately, dual booting gives better performance than running off a usb which gives better performance than running in a virtual machine, and which of these three options is sufficient largely depends on the computer you are using to run the simulation. GPU performance is likely the limiting factor, so while Matthew may be best served dual booting off his M.2 drive because he only has a 1070ti in his machine, Bobby may be able to use a virtual machine because he has a 3080 and plenty of GPU power despite the inefficient GPU passthrough in virtualization.

Now that we have settled on the best environment to run the operating system, we still have the task of getting Gazebo set up. For that, I have written a script. I gave this to the other team members via discord, but in the absence of the ability to attach a script to a text document, here is a screenshot of that script. It will install git and Gazebo, and get the .world file from The Ancestors and move it to the correct folder for Gazebo to access it. It does require “sudo” privileges, so you will have to run the command “sudo bash SDsetup.sh” from the command line in the folder in which this script is stored. If you prefer to type these commands into your terminal yourself and run them one at a time, you must add “sudo” before each of them. Either way, the first time you run a sudo elevated command in a terminal session, you will have to put in your machine’s password.

SDsetup.sh

```
#!/bin/sh
# This script is to be run on a fresh ubuntu install. if you already have things installed, there could be trouble.
apt update
apt install git
apt install ros-noetic-desktop-full

source /opt/ros/noetic/setup.bash
echo "source /opt/ros/noetic/setup.bash" >> ~/.bashrc
echo "source ~/catkin_ws/devel/setup.bash" >> ~/.bashrc
echo "export GAZEBO_MODEL_PATH=$GAZEBO_MODEL_PATH:~/catkin_ws/src/gazebo_models_worlds_collection/models" >> ~/.bashrc

source ~/.bashrc # source should do the same thing as closeing and reopening the terminal to make changes in .bashrc take effect.

rosdep init # possibly redundent, but won't hurt anything to do it again
rosdep update

mkdir -p ~/catkin_ws/src
cd ~/catkin_ws/
catkin_make
source devel/setup.bash

cd ~/catkin_ws/src
git clone https://github.com/patricklbauer/autonomous_search_with_ai.git
git clone https://github.com/chaolmu/gazebo_models_worlds_collection.git
cd ~/catkin_ws
catkin_make
source devel/setup.bash

cd ~/catkin_ws/src
apt install curl
curl -sSL http://get.gazebosim.org | sh
cp ./autonomous_search_with_ai/worlds/urban.world /usr/share/gazebo-11/worlds

```

Figure 34: Screenshot of the setup script, available on the team's github. If this fails, use the detailed gazebo installation instructions

Software and Tools Used

With a project this large, the team ended up using several different tools for development including Python, Gazebo, and ROS.

Gazebo and ROS

The simulation software we are using is Gazebo and ROS stands for "Robot Operating System" and is a popular tool for robot programming and simulation. While Gazebo

does not have very many different versions, they are simply numbered sequentially and their capabilities do not vary greatly from one version to the next, ROS is the complete opposite and requires careful consideration about what version to use.

Choosing Our ROS Version

First, there is a difference between ROS 1 and ROS 2. ROS 2 is still largely in development which makes ROS 1 is much more widely used, and is therefore more stable and documented. Every Version of ROS has a name associated with it, like Kinetic Kame, Noetic Ninjemys, or Melodic Morenia, and also is supported by only one distribution of Ubuntu, which has a name and number. Currently, the only two still supported versions are Noetic Ninjemys for Ubuntu 20.04 Focal Fossa and Melodic Morenia for Ubuntu 18.04 Bionic Beaver. While no longer supported, Kinetic Kame for Ubuntu 16.04 Xenial Xerus and Indigo Igloo for Ubuntu 14.04 Trusty Tahr are notable for their multi drone capacities. A drone package named Hector for Indigo stands out in particular. Regardless, the team is yet to make a decision on what branch of ROS to use, but we are leaning towards using Noetic on Ubuntu 20.04 since this is currently the most recent version of ROS and is the same version that the Ancestors used. In the end, Noetic on Ubuntu 20.04 was what we decided to use.

Opening the .world File

Gazebo has a folder named worlds which comes with a bunch of files in it with the .world extension. Loading a world is as simple as using the “gazebo” command with the file path starting from the worlds folder to the name of the .world file you wish to load as an argument. These .world files are simply XML files, and if you have prior

experience with XML, it will help you here. Additionally, these files are in the Simulation Description Format (SDF), whose structure is shown below in an image taken from Gazebo's website.

Figure 35: Diagram showing the SDF format

Model files also use XML and the SDF format. There is an open database that has models for all sorts of things like people, cars, and many robots like the turtlebot, parrot drone, and BB-8. If you open a world that uses a model from this database, it should automatically be downloaded at runtime.

Gazebo is actually two different executables, one is the server and the other is the client. The server is the part that is actually running the simulation. The client gives a

graphical user interface and lets you, a human, see what is happening in the simulation. It is possible to run only the server without the client, and is sometimes referred to in documentation and guides as running “headless.”

Another option discussed in a later section is launching a world file using a launch file. Launch files involve starting ROS nodes and require much more discussion, but all that is relevant in this section is that the “gazebo worlds/worldname.world” command is not the only way to open a world file

Gazebo Middleware

Gazebo has many plugins to help developers simulate what they need to. Currently, the team is looking into what plugins could be useful to the project, and if we find any we think are useful, we will list them here. GzUavChannel was considered for use by the team, but ultimately decided against as rVis is sufficient for debugging.

Challenges with Opening the World

Ultimately, we intend to use the previous senior design team’s world as the search environment for our swarm, and the first step of that is to open it on its own. We had some troubles with this, and after consulting Patrick Bauer, the project lead of the previous senior design team, we tried determined that the problem was that some non-standard models were used in the world, which meant that they could not be automatically fetched from the Gazebo model database, and the world never opened.

The non-standard models used are found in the github titled “Gazebo Models and Worlds Collection.” Additionally, we had to fix our path variable for Gazebo. After fixing these things, we were able to get the world to open and we could get a look around. This should be included in the setup script which was shown earlier.

The Environment from the Drone Swarm’s Perspective

It should come to no surprise that humans experience the world at ground level, whereas drones have the advantage of flight. Being higher gives a different perspective of the environment than being on the ground does. Consider these two pictures taken from the environment.

Figure 36: Comparison of line of sight based on altitude

Figure 37: Comparison of line of sight based on altitude

Take note of how much of the search area can be seen from ground level as opposed to from an aerial perspective. Both screenshots are taken from the far corner of the walled-in search area, but the elevated perspective allows seeing the majority of the uncovered road. While this is quite helpful, being higher in the air can also reduce the amount of area you can see as well. Consider the next two images of the search area.

Figures 38 and 39: Comparison showing the possible pitfalls of aerial imagery

As you can see in these two images, being elevated can also be a disadvantage in an urban environment because it can block line of sight to important things. Notice how the red car under the gas station weather awning is completely obstructed from view.

BB-8 is closer to the size of a dog than an entire car. Searching at several altitudes will be crucial to the success of the swarm in finding its quarry.

Spawning BB-8 with a Launch File

The next task is to place BB-8 in the world and get aerial pictures of him for object detection training purposes. In order to do this we must use launch files. A launch file is an XML file with the .launch extension that can be run from the command line in linux using the command “roslaunch package_name launch_file.” A package as referenced in the first parameter of the roslaunch command is related to a carkin workspace and deploying and building things in ROS and discussed in its own section. The main purpose of a launch file is to start ROS nodes. In order to understand why that is significant, let's discuss what ROS nodes are.

ROS Nodes

A ROS node is a process that performs a task. You can think of them as threads in Java or another parallel computing framework, if you have experience with that. Nodes can be combined into a graph to show how they are related and they communicate with each other through topics. Any robot can be made of a collection of nodes, perhaps one deals with the camera, another the LIDAR, and a third with pathfinding and locomotion. Splitting up control of a robot across many nodes can be beneficial in the same way splitting up code into several files or classes can be beneficial in that it allows you to avoid having one monolithic system. Additionally, it also means that a robot designed in this way will be more resilient to error, because if one node crashes,

it only loses the functionality of that node instead of the entire robot going out of commission.

Custom ROS Nodes

You can create your own ROS nodes in python by including “#!/usr/bin/env python” at the beginning of the file. Immediately after that line, be sure to include “Import rospy” in order to use that library. Rospy is a Python client library for ROS and is used similarly to other libraries in Python, or any other programming language.

ROS Topics

ROS Topics are how ROS nodes are able to communicate with each other. Topics typically have publish/subscribe functionality which allows nodes to put out information to the rest of the system asynchronously from when other nodes need to consume that information. Having this built in communication system is hugely useful, especially since so much of this project revolves around several individual robots working together as a swarm.

ROS Messages

ROS messages are how nodes actually put data on a topic and get data from a topic. Different messages for different topics may be formatted differently. For example, the ROS package named std_msgs has the type String, but other packages allow for sending pictures, navigation information, or data from robot sensors.

ROS Namespaces

All ROS nodes have a namespace which functions similarly to scope in a normal programming language. Namespaces are hierarchical, and nodes in one namespace cannot affect things in a different namespace. However, nodes do not need to be aware of what namespace they are located in as names are resolved relatively. When programming, nodes that are part of the same system and work together can be written as if they are located in the top level namespace, regardless of where they actually end up within the same namespace. Valid names must follow the following rules:

- The first character must be an uppercase or lowercase letter(A-Z, a-z), a tilde(~), or a forward slash
- Subsequent characters are alphanumeric, underscores, or forward slashes.
- Base names cannot have forward slashes or tildes in them

ROS Workspace

The build system of ROS is called Catkin. All packages built using ROS are built in a catkin workspace. Ultimately, a catkin workspace is just a directory with a folder in it named “src” where all the files for a project are kept, but the Catkin build system makes sure everything actually works when you try to run things. Generally, you will have one folder in your src folder for each package in your project, similar to using the javac or gcc commands to compile a file in a normal programming language. The command “catkin_make” will compile all packages in the workspace’s src folder. You may want to do the command “source devel/setup.sh” as well before trying to run your launch files.

ROS Packages

A ROS package is a way to organize your project. It does not do anything all that special, but you do need to be aware of what things are in what packages because the first parameter of the “roslaunch” command is the package you wish to run a launch file from, with the second being the name of the launch file itself.

Rviz

Rviz is short for ROS Visualization, and is a 3d visualization tool for robots and their sensors. Rvis allows you to view what information the robot is getting from its sensors, and is useful for visualizing the state of the robot. Morgan Quigley, an original

developer of ROS once famously said “rviz shows you what the robot thinks is happening, while Gazebo shows you what is really happening.”

Python

When using ROS, there are two options for programming languages to use, Python and C++. C++ is widely used for robotics due to its fast runtime and good interaction with low level software, but it is harder to write, especially for people who don't have experience with C++. Considering the team had experience with Python, and Python is generally more readable than other languages, we opted to use Python. Python also has many machine learning libraries like Tensorflow.

Tensorflow

Tensorflow is an open Source machine learning library for Python. It includes functions to design neural networks and implement deep learning. At the start of the semester, the team familiarized themselves with Tensorflow by doing some very basic exercises and reading up on some example code in preparation to use this library in Senior Design 2. The previous senior design team’s document mentions that the sponsor recommended Tensorflow for them, so we expect to make use of it.

Diagrams

This is the first diagram the team has regarding the project's operation. It is a drawing in Microsoft paint depicting Gazebo running in a virtual machine, with BB-8 hiding behind a tree and being spotted by a drone. While further investigations showed that dual booting may be a better option of achieving an Ubuntu environment for Gazebo to run in, a virtual machine may still be viable for those with more formidable computers. Additionally, the team has expressed some attachment to the art style, and has requested that I leave this in.

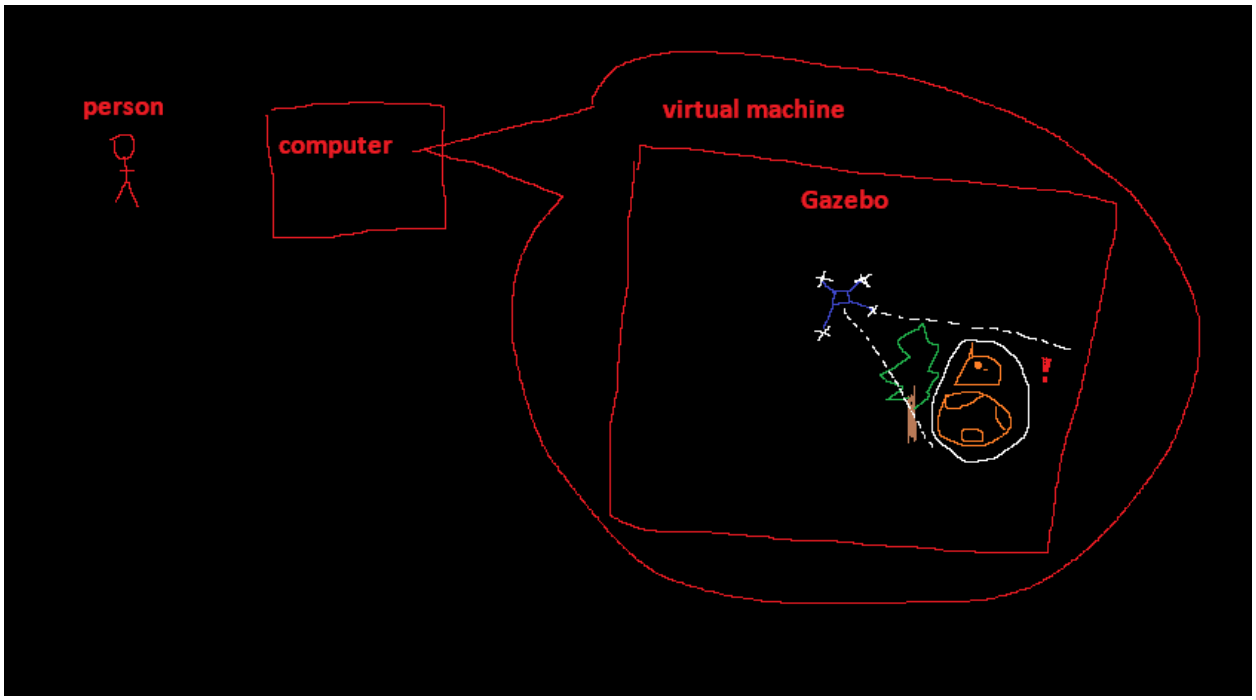


Figure 40: An incredibly early diagram hand drawn in microsoft paint. It has become an artifact of the team's shared culture and a testament to the team's interest in this project

This second diagram shows how ROS nodes can be used to separate the program into manageable portions. Each drone in the swarm can be made up of several ROS nodes, each node running the different function of the drone.

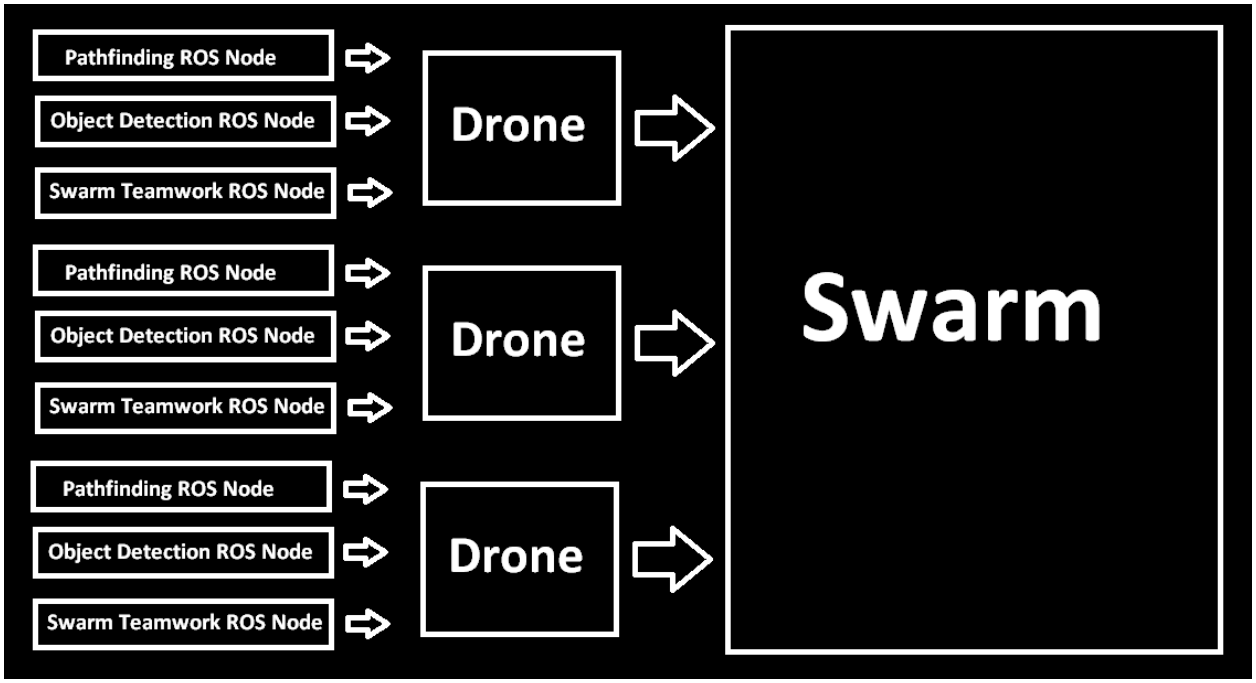


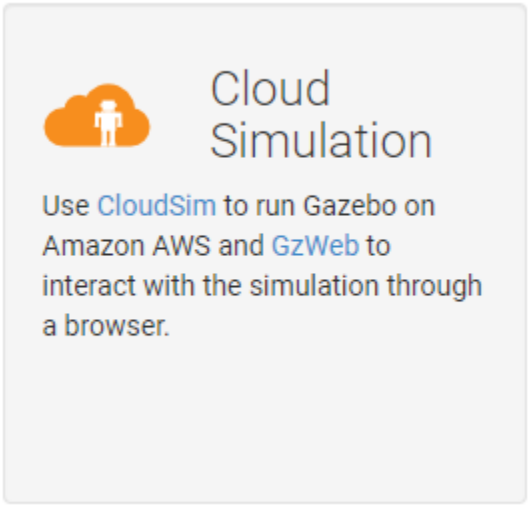
Figure 41: Block diagram showing how ROS nodes can break down a robot or swarm of robots into smaller building blocks.

Beyond the Document: For a V3

Cloud Solution:

One problem we ran into as we began to implement our paper solution for a swarm of drones were memory problems. Everything we ran created a new process which became a threading issue over time. We didn't have the time to solve the issue and had to resort to a 2D solution, but if there were to be improvements on our project beyond our V2 solution, this would be our answer.

Gazebo has a cloud option that allows you to deploy an environment on an AWS network, meaning all your processes are handled on a server and not on your computer. A Link to this site can be found here



https://classic.gazebosim.org/gzweb.html

This is the frontend instructions for how to communicate with your simulation using a frontend interface designed for gazebo simulations. This would be the workaround for dealing with multi-process problems and would be the team's recommendation if you had to move forward.

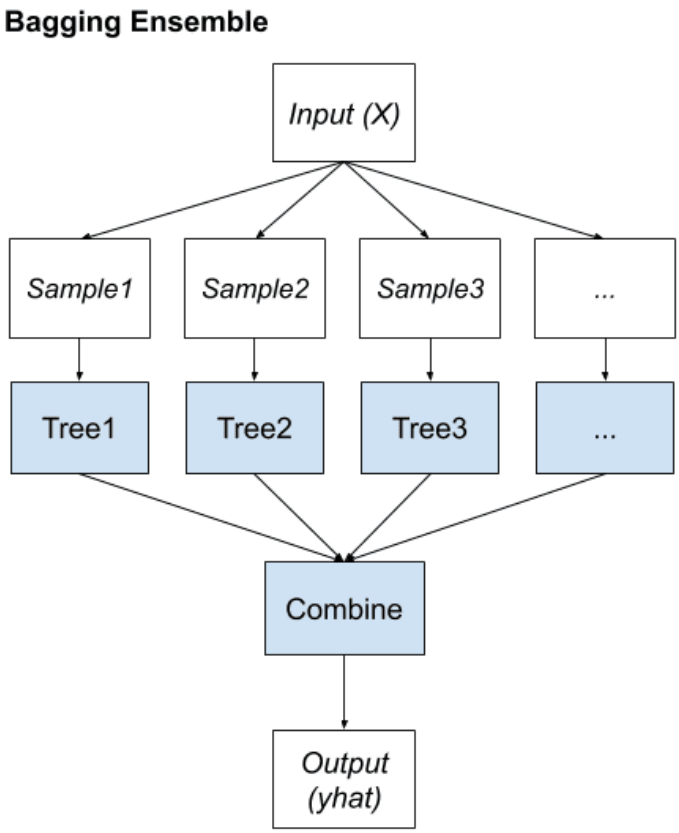
Potential Swarm Approaches:

As we worked through our project, got familiar with the technologies, we began to come up with more complex and arguably better approaches that we would recommend for future iterations. Not all of them were able to be implemented but we recommend using these as a way to think about the problem beyond researching existing solutions.

Ensemble/Cluster Swarm:

Based on the principles of Ensemble Machine Learning, the main idea behind this approach is to have a swarm of what we dubbed “Clusters” and each cluster has an odd number of drones. This approach would be good for a larger environment where there is more ground to cover and higher stakes on the line for dedicating the entire swarm to move to the target. Each cluster would be running their own Object Detection model which is trained on different data. For example, say there are 3 drones in a given cluster. Drone 1 should be running a different model from Drones 2 and 3. The datasets that were used to train these models should be unique. Each model shouldn’t have ever seen the data of the others. This is the same idea as the Bagging approach in Ensemble Machine Learning.

If a drone in a cluster identifies a target, the other drones in the cluster must verify and if the majority agrees that it is indeed the target, consider the target found. An image that helps explain Bagging Ensemble Learning is shown below.



Reinforced Learning Swarm:

Based on Reinforced Learning Principles, a basic design would need to be implemented with tunable parameters. Then the goal would be to run it through a reinforced model that will pick the best hyperparameters for the entire swarm implementation. These hyperparameters would be any tuneable numbers that the user finds to have an effect on the outcome of the swarm implementation. Examples might be speed, waypoint decisions, etc.

End of Senior Design 1 Retrospective

The first semester of Senior Design 1 focuses on researching the problem, the team introducing themselves to the new software and technology that will be used for the

project, and creating a design document in order to describe how to solve the problem the project presents. Considering this project is a continuation of a previous team's project and we have looked to the prior team's project and documentation for guidance, we expect the same to happen with our project. In this section, each team member will address what happened for them personally throughout the semester and what they wish they could tell their former selves about what was to come.

Matthew:

One thing that helped from the Senior Design boot camp is getting lunch with the team. Sharing a meal is a great way to bond together as a team, and in a project that spans the better part of a year, you are going to need to have some camaraderie to make it through the difficulties you will inevitably face. Being friendly with your teammates boosts your motivation to help them how you can, and adds social incentive to make sure you don't slack off. While cartoons and anime may have made you sick of hearing about the power of friendship, it is nonetheless a very real power, and you would be wise to harness it for yourself.

In general, you're going to be able to do less and less as the semester goes on. You will have the most free time at the start of the semester, and all your other classes will assign more and more stuff as the semester progresses, leading you to committing less and less time as final exam week approaches, until you are scrambling to get the last of your design doc pages written in the last week before they are due. Be sure to work on documentation and prototyping your project early, and try to get ahead as much as you can in anticipation of your other classes getting tough.

One thing our team did well that the other two teams seemed to have a bit more issue with was splitting effort between learning Gazebo and searching through research papers for what solutions already exist. Personally, I ended up doing much more Gazebo learning than the rest of the team, which allowed the others to make prototypes for the algorithms that we intend to implement next semester.

Your senior design document is the equivalent to your final exam for this class. It is a massive document spanning every topic your project touches, and with such a large scope it is possible that several people will write about the same thing without knowing during the stage of the class where each person keeps their pages separately. If this is the case, you will lose some pages when you merge your documents together. Be wary of this, and try to merge your pages together as many days in advance of the due date as possible in order to give time to fill in for the pages you may lose.

With regards to utilizing a budget provided to your team, you will need to request what you want purchased for you as soon as possible, and ask UCF if they will allow submission of receipts for reimbursement. The subscription for Gazebo and ROS training courses by The Construct we requested still has not come through at the time of writing, and it has been close to 5 weeks at this point.

Andrew:

This first semester of senior design my time on this project went heavily in the direction of researching pathfinding and path planning algorithms for drones, and the more I looked into those matters, the more I inevitably found myself also needing to find information regarding swarm cooperation as well. Something I would absolutely go

back and tell myself is to work more closely with Sebastian, who was focussing on algorithms for swarm optimizations, to take better advantage of the overlap in our respective research areas. The benefits from looking at things more closely together would have allowed each of us to more quickly get on the same page regarding how these aspects of managing the drone swarm's movements would relate to each other.

For the most part, I felt very good about how our team functioned as a group; we were able to very early on form a cohesive group dynamic and set up an effective distribution of efforts. James as the project manager set us up well for success at the beginning of the semester with clear goals, and everyone made a good effort towards accomplishing those goals, despite the array of obstacles which presented themselves. Personally, I felt that this went particularly well at the beginning of the semester, but as we approached thanksgiving and finals, I found myself with much less time and started to struggle to dedicate as much effort towards this research and prototyping. To have started out with an especially strong push early on in the semester even when things were fairly calm was a wise decision, one which I would certainly make again, but also one I may not have made were I to have not been pushed to do so, and so I am glad for the effectiveness of the group and its leadership.

All in all, I have learned a lot so far through this project, and what seemed at the start to be a massive undertaking certainly still seems the same way today, but I feel that we now have a much more clear way forward through it. I am excited for the coming semester, when we can all take what we've learned and gathered and finally start piecing it all together to proudly watch some drones fly through the city on their great search.

James:

-What went well

1. We stuck to the Gantt and maintained weekly meetings throughout all of Senior Design 1

2. We completed multiple prototypes and acquired knowledge for all required elements. Thus, implementation will be much smoother not having to learn the topics from scratch

3. The beginning of the semester was utilized well before other classes got busy, a lot of work being done in little time

4. Attendance was solid for the most part, a few fallouts but the schedule was held and communication maintained

5. Camaraderie was good throughout, no drama between the team

-What didn't go so well

A lot of version issues with Gazebo/ROS halted progress despite having two members of the team dedicated to the topic

1. Couldn't begin gathering our actual dataset due to Gazebo/ROS halting progress

2. When finals began to come closer, the team began to lose time for the project and achieving our deadlines began to become harder as the days went on

3. Creating a prototype for Swarm was not straightforward being dependent on everything else working properly

Tips For a Future Project Manager:

- Enforce mandatory meetings, make sure to have an agenda. If you don't show the team you are on top of things, they won't feel obligated to do so.

- Your dedication shows. If you slow down, the entire team does too. Doing what you say you will do is more crucial for you than anyone else on your team.

- Don't let time stress you. Make a solid plan, enforce it, and day by day a mountain of work and progress will slowly chip down to something more digestible

- Do not wait to do your documents! Set deadlines for the minimum required pages

- If you have funding, get that done immediately! Talk to the professors and get the process rolling as soon as possible. They won't be waiting for you and you will definitely be waiting for them

- Ask all the questions. The more you can get answered, the less unknowns there will be throughout the whole process.

- Take holidays, breaks, and finals season into account. Things will slow down, guaranteed. Make sure your plan takes these things into account.

- Good camaraderie is important. Meet in person as much as possible, don't make everything about the work. Have some lunch, hang out, you will be working together for a while so make sure to keep things positive and fun.

- Finally, enjoy the process. The ups and downs will only be that much more rewarding when you get to the end.

Sebastian:

This semester the targets of my research were robotics simulators with a focus on drones and the swarm algorithm. While the two areas differed greatly, I feel this gave me a very well rounded view of our project. Getting the simulation running and our algorithms programmed is very important for our success. A real point of challenge will come in linking the two so I believe that I will be well positioned to work on this particular problem.

Over the course of the semester, I think I have improved my approach to robotics problems, especially in simulations. I found watching guides and practicing implementing lots of examples to be very important so you can get comfortable with the simulators and all of their nuances. Beyond this, I think that classes in ROS would be very valuable in helping me understand the underlying code powering the robots and would provide us with the capability to outfit robot models with the sensors such as LIDAR and cameras as well as any other additions we would like to makel. Based on this I believe that courses would be a worthwhile investment going into Senior Design 2. Early on I was very concerned about just having something to show in the simulators, I have since changed focus to truly understanding the robotics systems since that is what will allow for flexible and comfortable implementations of our drone swarm, especially with all of the variables involved in with different algorithms, sensors, drones, and targets. Thus a firm grounding in robotics systems is the best solution to gaining the ability to solve this problem.

The other main focus of my research was the swarm algorithm. My approach for choosing an algorithm for a complex problem improved greatly in this research. Since this problem is relatively novel it was necessary to read several research papers to pick

an algorithm. Especially the particle swarm optimization algorithm which has many variants and applications for each variant. A word of advice here is that many articles cite lots of other similar algorithms that they have built their model off of. A lot of times these are worth reading since different authors vary a lot in how well they describe algorithms, implementations, and results. Beyond this, these articles will help you build a foundational knowledge in the general area you are looking at, sometimes they reference a model that is better for your situation than theirs is just based on use case alone.

Team dynamics wise, our group has been organized well in terms of weekly meetings and open communication through discord, largely thanks to our project manager James. He has done an excellent job in coordinating our efforts especially in the beginning when we were first getting started. We have a GANTT chart but with research heavy projects where there are many uncertain factors, it has not been as effective at organizing our efforts as it would be for more traditional software engineering projects.

Bobby:

I would say the most important thing is to be prepared to be bombarded with work mid semester. The due dates on the pages creep up fast and it was in my best interest to work on them as I was doing my research as opposed to gathering all my research and writing the paper after the fact. Other than that I would say these semesters went as smoothly as it possibly could have and that we made a lot of good headway. Also, when your group has momentum, run with it. We found that we had found the most success when all of us were all able to collaborate and help each other while not being slammed with school work.

Clear communication also allowed us to help each other's downfalls, which I thought our team handled spectacularly. There were weeks where some of us couldn’t attend meetings and also weren’t able to get certain things done for a slew of different reasons, but we were able to help each other when needed due to clear communication.

This was also my first project that I’ve done since COVID which ended up being a completely different experience than the others during COVID. I enjoyed meeting my group mates thoroughly and being able to spend time in person with them even when it wasn’t related to work. I think getting to know each other outside of work allowed us to work together more effectively and be a bit more understanding when certain things didn’t really work out.

I’d also like to note that I attribute a lot of our success in hitting the ground running at the start of the semester. Our project leader James paved the way to success for us by working very hard in the beginning to give us something tangible to work towards. Having attainable clear set goals early on allowed us to knock it out of the park in the first few months of the semester so that we were able to slow down pace a bit when things got wild.

If I were to go back and tell myself anything, I’d say try your best to get the budgeting portion worked out as soon as possible. Not being able to access our resources hindered us a decent amount this semester to the point where we probably will need to spend some extra time over the break learning what we didn’t get a chance to learn during the semester. Other than that I would say to try and do your best to keep the ball rolling throughout the semester and try your best not to get burnt out. These last few semesters are tough but it’s all doable with appropriate time management.

References:

19/12/2020, et al. “Guide to yolov5 for Real-Time Object Detection.” *Analytics India Magazine*, 18 Nov. 2021, https://analyticsindiamag.com/yolov5/.

“A 3D Frontier-Based Exploration Tool for Mavs.” *IEEE Xplore*, https://ieeexplore.ieee.org/document/7372156.

Alfeo, A. L., et al. “Swarm Coordination of Mini-Uavs for Target Search Using Imperfect Sensors.” *ArXiv.org*, 9 Jan. 2019, https://arxiv.org/abs/1901.02885.

Anirban, et al. “A Gentle Guide to Deep Learning Object Detection.” *PyImageSearch*, 17 Apr. 2021, https://www.pyimagesearch.com/2018/05/14/a-gentle-guide-to-deep-learning-object-detection/.

ArduPilot. “Ardupilot.” *ArduPilot.org*, https://ardupilot.org/.

Coordination for Multi-Robot Exploration and Mapping - AAAI. https://www.aaai.org/Papers/AAAI/2000/AAAI00-131.pdf.

D Lite - Idm-Lab.org*. http://idm-lab.org/bib/abstracts/papers/aaai02b.pdf.

Financial Inclusion in Egypt: Opportunities and Challenges. https://fount.aucegypt.edu/cgi/viewcontent.cgi?article=1798&context=etds.

"Frontier Exploration Technique for 3D Autonomous Slam Using K-Means Based Divisive Clustering." *IEEE Xplore*, https://ieeexplore.ieee.org/document/8424313.

Gad, Ahmed. "Introduction to Optimization with Genetic Algorithm." *Medium*, Towards Data Science, 3 July 2018, https://towardsdatascience.com/introduction-to-optimization-with-genetic-algorithm-2f5001d9964b.

Girshick, Ross, et al. "Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation." *ArXiv.org*, 22 Oct. 2014, https://arxiv.org/abs/1311.2524.

"Google Colaboratory." *Google Colab*, Google, https://colab.research.google.com/drive/1gDZ2xcTOgR39tGGs-EZ6i3RTs16wmzZQ.

"GZUAV." *GzUav*, https://gzuav.dmi.unict.it/.

"How the Compute Accuracy for Object Detection Tool Works." *How the Compute Accuracy For Object Detection Tool Works-ArcGIS Pro | Documentation*, https://pro.arcgis.com/en/pro-app/latest/tool-reference/image-analyst/how-compute-accuracy-for-object-detection-works.htm.

Hyperparameter Evolution - Yolov5 Documentation. https://docs.ultralytics.com/tutorials/hyperparameter-evolution/.

"An Improved Active Slam Algorithm for Multi-Robot Exploration." *IEEE Xplore*, https://ieeexplore.ieee.org/document/6060232.

"Introduction to Yolo Algorithm for Object Detection." *Section*, https://www.section.io/engineering-education/introduction-to-yolo-algorithm-for-object-detection/.

"Introduction." *Introduction · MAVLink Developer Guide*, https://mavlink.io/en/.

Liu, Ya, et al. "Moving Object Detection and Tracking Based on Background Subtraction." *SPIE Digital Library*, SPIE, 24 Sept. 2001, https://www.spiedigitallibrary.org/conference-proceedings-of-spie/4554/0000/Moving-object-detection-and-tracking-based-on-background-subtraction/10.1117/12.441618.short?SSO=1.

Liu, Yanli, et al. "A Fast Map Merging Algorithm in the Field of Multirobot Slam." *The Scientific World Journal*, Hindawi, 2 Nov. 2013, https://www.hindawi.com/journals/tswj/2013/169635/.

Narkhede, Sarang. "Understanding Confusion Matrix." *Medium*, Towards Data Science, 15 June 2021, https://towardsdatascience.com/understanding-confusion-matrix-a9ad42dcfd62.

OctoMap: An Efficient Probabilistic ... - Www.arminhornung.de.

http://www.arminhornung.de/Research/pub/hornung13auro.pdf.

Osrf. “Gazebo Components.” *Gazebo*,

http://gazebosim.org/tutorials?tut=components&cat=get_started.

Osrf. “Quick Start.” *Gazebo*, http://gazebosim.org/tutorials?tut=quick_start.

Osrf. “SDFORMAT Specification.” *SDFormat*,

http://sdformat.org/spec?ver=1.8&elem=sdf.

Osrf. “Ubuntu.” *Gazebo*, http://gazebosim.org/tutorials?tut=install_ubuntu&cat=install.

Paez, David, et al. “Distributed Particle Swarm Optimization for Multi-Robot System in Search and Rescue Operations.” *IFAC-PapersOnLine*, Elsevier, 29 Oct. 2021,

https://www.sciencedirect.com/science/article/pii/S2405896321014038-.

Pathfinding 3D Report - GitHub Pages.

https://ascane.github.io/assets/portfolio/pathfinding3d-report.pdf.

Price, Adam. “Optimising Boids Algorithm with Unsupervised Learning.” *Medium*, Towards Data Science, 22 June 2020,

https://towardsdatascience.com/optimising-boids-algorithm-with-unsupervised-learning-ba464891bdba.

Rawat, Shivam. “How to Prevent Google Colab from Disconnecting ?” *Medium*, Medium, 3 Sept. 2020, https://medium.com/@shivamrawat_756/how-to-prevent-google-colab-from-disconnecting-717b88a128c0.

Redmon, Joseph, et al. “You Only Look Once: Unified, Real-Time Object Detection.” *ArXiv.org*, 9 May 2016, https://arxiv.org/abs/1506.02640.

Search and Tracking Algorithms for Swarms of Robots: A Survey. https://www.researchgate.net/publication/283905903_Search_and_tracking_algorithms_for_swarms_of_robots_A_survey.

Solawetz, Jacob. “PP-Yolo Strikes Again - Record Object Detection at 68.9fps.” *Roboflow Blog*, Roboflow Blog, 11 May 2021, https://blog.roboflow.com/pp-yolo-strikes-again/.

Srinivas, Author. *Quest for AI*, 13 Aug. 2021, https://questforai.wordpress.com/.

StevieG47. “Stevieg47/Object-Detection-Classification: Image Classification, Object Detection in Tensorflow and Opencv.” *GitHub*, https://github.com/StevieG47/Object-Detection-Classification.

“Theta*.” *Wikipedia*, Wikimedia Foundation, 16 Dec. 2019, https://en.wikipedia.org/wiki/Theta*.

Ultralytics. “Ultralytics/yolov5: Yolov5 in PyTorch > ONNX > CoreML > TFLite.” *GitHub*, https://github.com/ultralytics/yolov5.

“Welcome to DroneKit-Python's Documentation!¶.” *Welcome to DroneKit-Python's Documentation!*, https://dronekit-python.readthedocs.io/en/latest/.

“Wiki.” *Ros.org*, http://wiki.ros.org/Distributions.

“Wiki.” *Ros.org*, http://wiki.ros.org/ROS/Tutorials.

“Yolo: Real-Time Object Detection Explained.” *V7*, https://www.v7labs.com/blog/yolo-object-detection.

“Zero to Hero: Guide to Object Detection Using Deep Learning: Faster R-CNN,Yolo,SSD.” *CV*, 28 Dec. 2017, https://cv-tricks.com/object-detection/faster-r-cnn-yolo-ssd/.

Zhiqiang, W., and Liu Jun. “A Review of Object Detection Based on Convolutional Neural Network: Semantic Scholar.” *Undefined*, 1 Jan. 1970, https://www.semanticscholar.org/paper/A-review-of-object-detection-based-on-convolutional-Zhiqiang-Jun/663a60841c3e703d7c18cf78f0657efee6aebb9d.