Using Autonomous Drone Swarms and Deep Neural Networks for Searching Dynamic Simulation Environments

Design Document: v2.0
Fall 2021 - Spring 2022

Group 23:

Keegan Kerns, Jakob Germann, Akash Samlal, Caleb McCown, Jeremy Chan
Github Repository: https://github.com/kkerns2/swarmDrone

Sponsorship: Lockheed Martin Missiles and Fire Control Applied Research
Scott Nelson, Joseph Rivera

Table of Contents

1. Project Overview

1.1 Problem Statement

The Lockheed Martin Missiles and Fire Control Applied Research group requires an implementation of an urban simulation in the Gazebo software and an autonomous drone swarm capable of traversing said environment while searching for a predetermined target. The development team has been directed to implement all software systems using the Robot Operating System and Gazebo simulation

environment. This project is to be solved by senior design group 23 no later than March 2022, and will aid in reconnaissance, security, and search & rescue missions.

The drone swarm is expected to conduct path planning through the environment and detect the target within a 3 minute time period. The object detection model is expected to achieve a minimum accuracy rating of 70% and IoU minimum of 0.5. The sponsor has also specified that the number of drones within the swarm must be between 4-5.

1.2 Statements of Interest

1.2.1 Keegan Kerns

There were many projects offered to us as students for Senior Design, the swarm simulation project drew my eye because of its many aspects of Computer Science applied to achieve the end goal. In this project, there is computer vision, operating systems, as well as artificial intelligence. Artificial intelligence is something I have been introduced to and used for small assignments as well as one project, however it's not something I had used often, therefore getting deeper into the topic's intricacies is exciting. When it comes to computer vision itself, I am a novice and am looking forward to doing the research and learning from my team members on the topic. Also dealing with a new operating system known as ROS is something that I am intrigued by and from current knowledge is something that I know will come in handy in the future. I am motivated by this project because there are many opportunities to learn and grow as a computer scientist.

My previous experience working for the DOD, there is high demand for engineers and scientists who are able to conduct projects that blend A.I. and computer vision to improve and broaden mission capability. I plan to contribute to the defense of our country by learning and applying these subjects to projects in the future. Not to mention I enjoy working on a team and building something that I can be proud of. That means that even if it's a project that explores the cutting edge I know that going forward in this career path I am content. In this case, I know that our sponsor Lockheed Martin will put our hard work to good use for the right reasons. Which motivates me to push my hardest, as well as contribute and learn all I can.

1.2.2 Jakob Germann

While difficult to choose from all the proposed projects, Lockheed's swarm project interested me the most for a variety of reasons. Along with a personal interest in artificial intelligence, the project allows me to have a more tangible product come its

completion. I have found that building something abstract, as is the case with almost all class assignments, can be incredibly unsatisfying. This is why I have always been interested in the robotic, AI, and biological applications of computer science since they offer projects that are more concrete in nature. While still in a simulation, the Swarm project has the potential to be used in physical applications in various industries which is what motivated me towards it. It also helps that the skills I would gain from partaking in it would be used in similar projects with other companies and research groups.

Despite still being new to artificial intelligence and modeling/simulation applications, these fields have always been of significant interest to me. Thus, being able to develop an application that can have a major impact on the lives of individuals on and off modern battlefields is an exciting prospect. It will also be interesting to see how our application is used in the future once the next generation of senior design students begin to develop a physical version of the drone swarm. It is my hope that our work these next two semesters will help pave the way for a new generation of automation and AI technology.

1.2.3 Akash Samlal

My sole decision to major in Computer Science was mainly competing at the FIRST TECH Challenge Robotics Competition at my local robotics club. Our high school club was created when my friends and I pitched to our high school teacher and later got approved by the principal. We had roles assigned to each other. I was the only programmer on the team since everyone else wanted to work on hardware. With the amount of pressure on me, I had to force myself to learn Java and understand how to use the FTC API for the robot. After our first year of the club, we were more experienced than ever, we competed at two qualifying matches, in our 1st qualifying match we reached 3rd place and in our 2nd qualifying match, we scored 1st place, which was a big deal to all of us. Late nights at the lab, learning how to program the encoders, navigate the robot autonomously, and reach the States for the first time fueled my passion for entering the Computer Science domain.

I worked on multiple projects outside of my academic coursework at college, similar to what my colleagues have. I ended up creating a car chase game on unity and publishing the app on android for the first time, competed at a hackathon, and demoed a project that calculates a user form based on a workout and determines if the user is in the proper form or not. As well as solving a personal problem for myself, which was a snipping tool similar to in windows but keeps a record of previous snips and has a print function. The creation of the tool helped a lot in my homework and in studying for exams since I will have a snip of a question and right after a solution which is much quicker than copying and pasting to MS Paint.

Spring 2021 was when I took Robot Vision, and I immediately fell in love with the class since we had to compute image preprocessing from scratch without using any libraries at the time. Other classes were allowed, but my professor was new to teaching undergrad, and he was gauging the level of difficulty of assignments to assign for undergrad. Needless to say, despite many all-nighters to solve the homework, I have learned so much from that one class compared to my other CS courses that I know what I want to study in the CS Field, and that would be Computer Vision.

Shortly after Spring Semester, I was also able to get into the CWEP Program with Lockheed Martin and UCF, and the realm I worked under was in Computer Vision which was a blessing in disguise. I learned the state of the art deep learning technologies and computer vision applications; I was drowned with so much knowledge that I was excited to work with my peers and tackle new challenges. After seeing the Swarm Project Proposal, it has covered mainly all the aspects that I'm interested in, which is Computer Vision, aspects such as object detection and classification, and simulating searching mechanisms that are relatable and useful in the future. This is my motivation to apply and work for the Simulated Swarm project for Lockheed Martin.

1.2.4 Jeremy Chan

My motivation for the project is because of its technicality and complexity. I believe it's one of few projects that embodies the definition of "senior design". I really wanted to do a project that would force me to be a better programmer and designer. I find myself favoring science and engineering related programming over web development. This project is a perfect challenge for me because of robotics and AI. Since this project is sponsored by Lockheed Martin, an industry leader, I would get a view of what it's like working for a large project.

In the past, I took an introduction course to AI, and this semester, I am taking robot vision as one of my technical electives. This project is a strong mix of those two disciplines, and it will absolutely be a challenge to apply my knowledge. I'm always excited to see how my classes relate to real world ideas, applications, and projects.

1.2.5 Caleb McCown

In my time learning Computer Science I have been very interested in Artificial Intelligence and the hardware aspects of the computer, especially when it comes to robotics. This Simulated Swarm project seemed to fit my interests quite closely. There were other projects that I felt seemed more interesting, but this project would equally

broaden my horizons to other areas of Computer Science, while learning about the topics I have been interested in learning about.

My interests coincide with the project due to my upbringing. Ever since middle school I had been interested in robotics, dealing with the sensors digitizing real-world information and the robot then having some action based on that information. Artificial Intelligence has been a relatively new interest to me while I have been at college, such as participating in clubs and writing papers on it. I have never done much with simulation environments, but I am looking forward to learning more about it to add to my knowledge of Computer Science and programming. I am also happy to be doing this project for Lockheed Martin, as they are a respected company in the industry and due to my respect for the military.

Ultimately my motivation in doing this project comes from an amalgamation of what was previously described. I look forward to working with and understanding new topics in Computer Science. I enjoy the challenge and its complexity, and it seems to be a project that matches much of what I am interested in.

1.3 Broader Impacts

Creating a more adaptable, mobile support unit that allows for friendly combattants to better manage where, who and what is around them. This is most likely for Marines that are engaged in operations that require high resolution coordination. The greatest innovation in warfare will always be communications, and if you can supply accurate data from drones and navigate team members based on live intel then that means less casualties and more successful operations. As the technology becomes more open source, this could mean local law enforcement may be able to benefit from this innovation. However, there is risk in doing this since that means the more open source this means adversaries can adapt and overcome if they learn the system as well. No matter the case this is a technology that can change how tactical engagement is conducted.

Off the battlefield, there are a variety of applications that the drones can be used in. It is important to note the alarming amount of natural disasters that occur across the United States every year. As such, conducting search and rescue operations of those lost in the damage from these events is becoming ever more paramount. Our drone swarms would be able to play a vital role in assisting operators when locating civilians in the wreckage of urban environments after an earthquake has struck, finding individuals in the forested areas of the northern U.S., and even finding missing persons at sea. As an example, hurricanes in the southeastern United States are a constant problem faced by members of these communities. People have the potential to be trapped within the

wreckage of their homes or even be swept away due to flooding. While search and rescue operators are efficient, there are always those that are missed or that cannot be reached in time. Mass deployment of drone swarms across afflicted cities would allow for operators to better locate missing civilians in time. The drones would also be able to carry small packages containing food rations, drinkable water, basic medical supplies, and even communication units to those who first responders cannot reach immediately. As technology develops and more powerful propulsion systems are made, the drones might even have the potential to extract missing people without the need for human assistance. This would allow first responders to focus on more heavily afflicted regions and save them from being overwhelmed in the face of such disasters.

1.4 Legal, Ethical, & Privacy Issues

1.4.1 Legal Issues

- There was no NDA that was provided, due to this we have no limitations on the secrecy or disclosure of the project with anyone.
- According to the sponsor, Black Team keeps all IP rights relating to the project. They are mainly concerned on how we can implement search algorithms and object detection and classification on targets, regardless if it's drone related.
    - UCF and Lockheed Martin assign IP rights exclusively to the Simulated Swarm Black Team Members.

- All drone related activities are conducted within the confines of the Gazebo simulation environment. As such, our project acts in accordance with all FAA guidelines.

- All drone related activities do not have any physical substance, since they are being simulated and will not be intended as weapons. Therefore, we would abide by all of ITAR's guidelines.
- Since our host operating system uses Ubuntu, one of the distributions of Linux, the license that is provided is GNU GPL License. The GNU GPL License also known as GNU General Public License, allows for any work conducted such as commercial, research, or personal use, can be used for free, however any work that was modified under it has to be under the license of GPL.

    - For instance if we modify the Linux architecture and distribute it, we have to put out a GNU GPL License under the distribution.

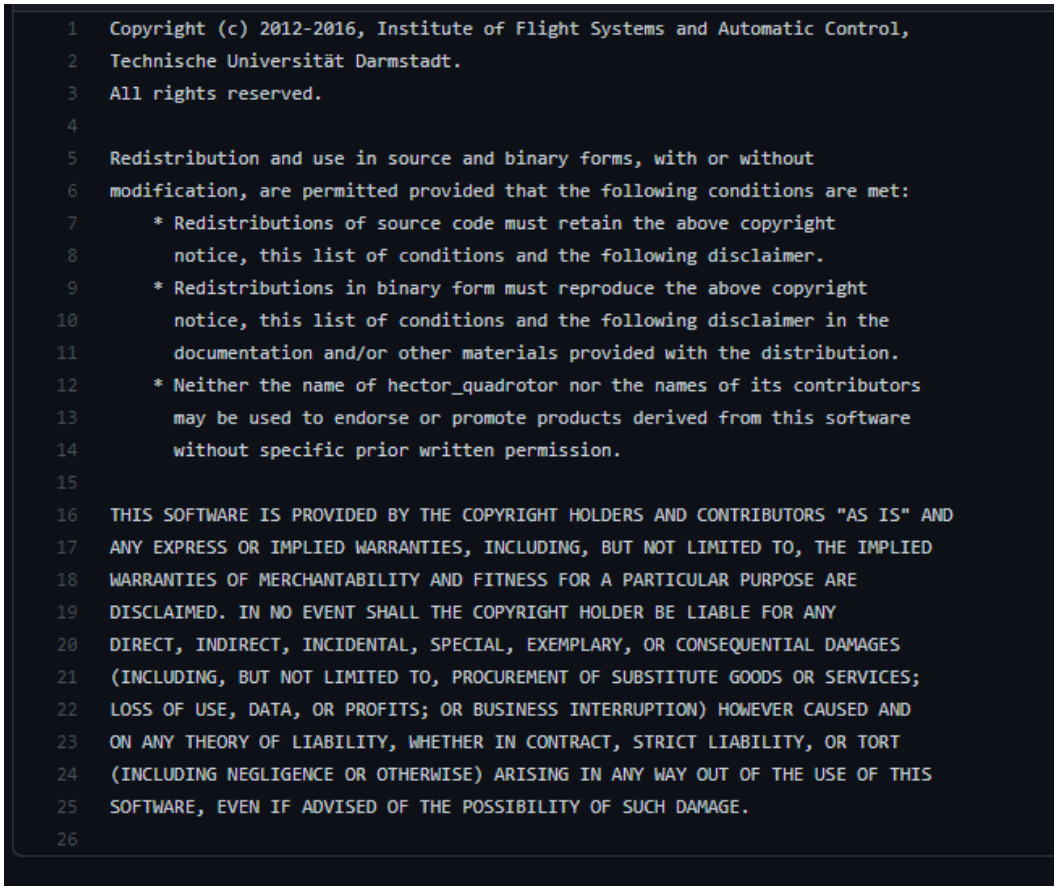
- Gazebo, a 3D simulation platform, is licensed under the Apache 2.0 license. The Apache 2.0 license is similar to the M.I.T. License in which we are allowed to

redistribute any of the product for commercial use, however we must specify that Apache doesn't sponsor or endorse our product.

- ROS, which is our main program to communicate between the drone and with the simulation uses the BSD License. The BSD License is considered a low restriction type license, which doesn't put any barriers or restrictions to the end user, and allows any modification and free use of the program. Only requirement is that we have to distribute the BSD License and copyright.

- Our drone's AI will not cause a robot uprising and become a secondary Terminator. As our simulated drones should not pass the Turing Test, the AI's would not advance to this level and abide by all the SKYNET.

- Since we are using Hector Drone from the Institute of Flight Systems and Automatic Control, we have to reshare the M.I.T. License once we make our repository open to the public. There should be no infringement of the license and as well if any source code has been modified we have to share everything to the public (which it will be). Part of the license requests us to not sell or distribute the content for commercial use, which if Lockheed Martin uses the Hector Drone, they have to find another alternative use since they will use it for commercial markers. As well we can not state that Hector Drone nor Institute of Flight Systems and Automatic Control is affiliated with this project.



```
Copyright (c) 2012-2016, Institute of Flight Systems and Automatic Control,
Technische Universität Darmstadt.
All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions are met:
    * Redistributions of source code must retain the above copyright
      notice, this list of conditions and the following disclaimer.
    * Redistributions in binary form must reproduce the above copyright
      notice, this list of conditions and the following disclaimer in the
      documentation and/or other materials provided with the distribution.
    * Neither the name of hector_quadrotor nor the names of its contributors
      may be used to endorse or promote products derived from this software
      without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND
ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER BE LIABLE FOR ANY
DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
(INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND
ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
```

Courtesty of Institue of Flight Systems and Automatic Control, Technische Universitat Darmstadt: Hector Drone License

https://github.com/tu-darmstadt-ros-pkg/hector_quadrotor/blob/kinetic-devel/LICENSE.txt

1.4.2 Ethical Issues

- The original objective of the project was to identify and destroy the target using the aerial swarm robots. However, the sponsor clarified and stated that there will be no malicious intent on any of the targets, such as identifying and destroying marked targets. Instead, the intent of the simulation project is mainly for academic purposes and how well can the team integrate today's state of the art solutions into ROS and Gazebo. Future application from the results of the project will be applied towards search and rescue missions. With a high accurate speed and reliability with the drones, they will be able to find and save more lives than before.

- With the current climate of the use of drones in the military, it is much to be expected that the use of algorithms we apply will be used towards marked living targets. The potential of mistaking a civilian as a marked target and costing their lives raises a lot of ethical concerns over who should use it. Fortunately, our algorithms will be trained with a BB8 target, so there will be no potential of transfer learning to carry over to human features from a robotic animation figure. If in the future, the Lockheed Martin Team wants to incorporate drone striking with the algorithms we implemented, the team needs to rebuild and train the project as a whole again.

1.4.3 Privacy Issues

- There will be no collection of sensitive data as well as saving and storing images of any person on a remote server for training. The only data that will be collected will be our target which is BB8. Multiple perspective views will be generated due to image augmentation on the BB8 dataset. Augmentations such as mosaic blur, rotation, cropping, translation, and random blooming are a few techniques that will be generated on the dataset. With object detection and classification, no data that is collected will be saved or stored to a server or locally from the drones itself. As well, the data that the drones will collect for the SLAM algorithms, will be hosted on a 3D virtual environment that is licensed for open source. There is no personal or sensitive data hosted in the Gazebo environment.

2. Team Dynamics and Division of Labor

2.1 Akash Samlal

2.1.1 Setting up the Ubuntu environment.

a. Setup ROS filesystem with ROS Noetic.
b. Setup Gazebo environment with Gazebo 11.
c. Create workstations to perform testing and development of new code.

2.1.2 Setting up Packages.

a. Implement Hector SLAM.
b. URDF files need to be found and implemented in Rviz and Gazebo.
c. Implement LIDAR Sensor.
d. Implement Odometry and Geometry messages.

2.1.3 Setting up swarm node

a. Setup inbound/outbound nodes to broadcast signals for detecting targets.
b. Integrating nav point publisher with YOLO node.

2.1.4 Propagate ROS to 5 drones in the Gazebo and begin testing against the target.There are two main things to consider when using multiple drones in the same simulation. We need to make sure that any topics that are pre-built that are needed can be duplicated otherwise there can be conflicting traffic that could confuse the swarm. The amount of overhead in rendering and simulating that many drones in a fully rendered environment.

2.2 Caleb McCown

2.2.1 Setting up Ubuntu environment
a. Setup ROS filesystem with ROS Noetic.
b. Setup Gazebo environment with Gazebo 11.
c. Create workstations to perform testing and development of new code.

2.2.2 Setting up object detection node
a. Pytorch implementation.
b. Integrating camera of a drone to feed images to ODS.
c. Develop a YOLOv5 solution.
d. Research Integration and multithreading

2.2.3 Propagate ROS to 5 drones in the Gazebo and begin testing against the target.There are two main things to consider when using multiple drones in the same simulation. We need to make sure that any topics that are pre-built that are needed can be duplicated otherwise there can be conflicting traffic that could confuse the swarm. The amount of overhead in rendering and simulating that many drones in a fully rendered environment.

2.2.4 Training object detection model
a. Collect pictures.
b. Utilize Roboflow.
c. Train the imageset.

2.3 Keegan Kerns

2.3.1 Setting up the Ubuntu environment.
a. Setup ROS filesystem with ROS Noetic.
b. Setup Gazebo environment with Gazebo 11.
c. Create workstations to perform testing and development of new code.

2.3.2 Setting up Path Planning Packages.

a. ROS Navigation Stack packages.
b. URDF files need to be found and implemented in Rviz and Gazebo.
c. Implement the AMCL node with correct parameters.
d. Implement the move_base node.

2.3.3 Setting up navigation point publisher

a. Setup inbound/outbound nodes to broadcast signals for detecting targets.
b. Publish points from map segmentation.

2.4 JJ Chan

2.4.1 Setting up Ubuntu environment

a. Setup ROS filesystem with ROS Noetic.
b. Setup Gazebo environment with Gazebo 11
c. Create workstations to perform testing and development of new code.

2.4.2 Training YOLOv5 model

a. Setup Roboflow
b. Gather images
c. Annotate
d. Create Google Colab training code
e. Train

2.4.3 Implement object detection node

a. Download model from PyTorch Hub.
b. Integrating camera of a drone to feed images to ODS.
c. Create Inference and drawing bounding boxes functions.
d. Parsing inference results as *pandas.DataFrame*.
e. Implement N-drones for N-instances of YOLO flexibility.
f. Multithread handling for N-drones and *CV2.imshow* function.

2.5 Jakob Germann

2.5.1 Setting up Ubuntu environment

a. Setup ROS filesystem with ROS Noetic.
b. Setup Gazebo environment with Gazebo 11
c. Create workstations to perform testing and development of new code.

2.5.2 Project Manager

a. Responsible for tracking all tasks, updating task documentation as needed, maintaining paper trail of all meetings, and continually schedule appointments with project sponsors, Teaching Assistants, and Professors as needed.

2.5.3 Design & implement modified Boid separation property for final swarm congregation.

a. Included development of distance calculation and final formation management.

2.5.4 Design & implement phase 2 navigation for extended swarm search.

a. Included map segmentation, waypoint extraction, waypoint division, drone assignment, and waypoint piping.

2.6 Project Management Methodologies

2.6.1 Agile

Throughout our research on the nature of the project, we decided that the Agile project management methodology would allow us the most flexibility, organization, and time-management efficiency. Since we were unfamiliar with the ROS system, we anticipated continuous change in certain project requirements and concluded that this management method would be best. At the recommendation of our sponsor, we turned to JIRA as our management platform.

2.6.2 Paired Programming

Early in development, we felt it necessary to incorporate the Extreme Programming method's paired programming approach. Thus far, we have found an effective increase in the success of development. Part of the exercises we take during these programming sessions is to have one person continuously planning the system architecture and pseudocode for each component while the other implements these ideas into Python or C++ code.

3. Project Requirements and Technical Information

3.1 Project Requirements

Project Requirements	
ID	**Description**
1	The drone swarm shall consist of 4-5 aerial based drones.
2	The drone swarm shall have a path planned through the environment using some Path Planning algorithm and LiDAR.
3	The Object Detection model will detect a custom target.
4	The object detection model shall have a minimum accuracy rating of 70%
5	The object detection model shall have an intersection of union metric of 0.5 or greater.
6	The drone swarm shall take no longer than 3 minutes to find the BB8 target within the environment.
7	The software shall be implemented using some version and distribution of ROS and Gazebo

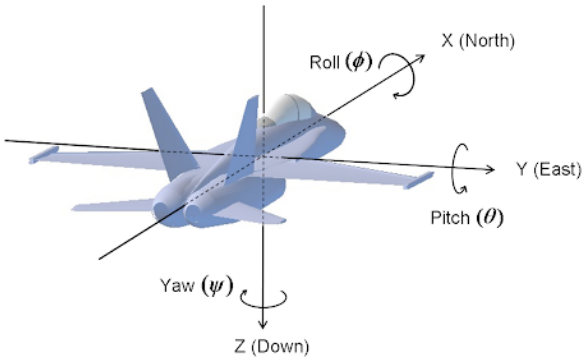
3.2 Drone Specifications

We will be using the Hector quadrotor drone model developed by a group from the Technische Universität Darmstadt in Germany. In order for the drone to function as one might expect real drones to, the appropriate physical properties must be coded into the Gazebo environment. These physics include the aerodynamic properties of the drone, propulsion methods, and rotational controllers and are programmed using C++. From there, they are integrated into Gazebo as custom plugins.

The movement properties of aircraft use Quaternions to allow for proper simulation of expected flight movements. Quaternions are mathematical models that are used in aircraft and rocketry control systems. They encode for the standard x, y, z axes in a 3-d

Cartesian coordinate system, but also add the axes of rotation to each previous axis: pitch, roll, and yaw.

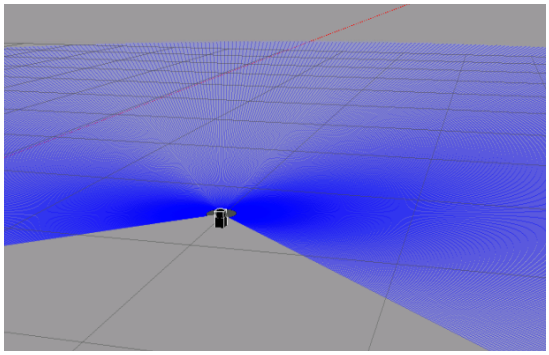
In the case of the Hector drone, the creators of the aerodynamics physics and aerial control systems simulate flight mechanics using the properties of quaternions. The method of which those properties are implemented differs from conventional aircraft in that there is more emphasis on using the lift and yaw axes when controlling the drone.



Example of a Quaternion in relation to an F-18 Hornet

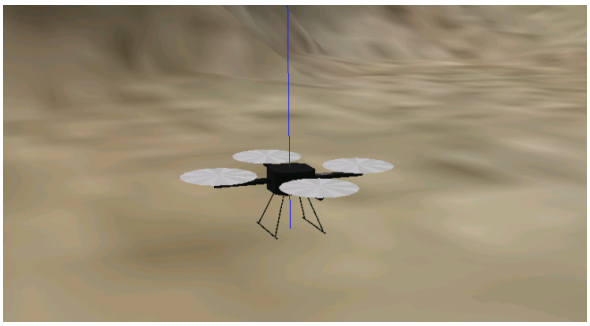
Courtesy of reference: http://edge.rit.edu/edge/P18043/public/Preliminary%20Detailed%20Design

The physical model relies on several URDF files that encode for different components including the drone body, rotors, LiDar sensors, and camera system. The model utilizes a Hokuyo UTM-30LX Lidar sensor that projects approximately 1081 sensor beams. The arc of projection ranges from -135° to +135° and provides continuous updates to the system at a rate of 40 updates per second.



Example of the Hokuyo UTM-30LX simulated sensor projecting its arc.

The drone itself is built off the premade gazebo model and was designed in blender. The mesh is then imported via its .dae file and used in the model's URDF simulation file. The URDF file specifies the physical components of the drone. Within this file, we specify collision, inertial, and visual properties using the six axes of a quaternion.



Example of a hector drone being spawned into the Gazebo environment.

The base model file sets the aforementioned properties to the following for its base link:

- **Inertial Properties:**
  - Mass: 1.477 kilograms
  - Inertial Origin: (0, 0, 0)
  - Inertia: (ixx = 0.01152, ixy = 0.0 ixz = 0.0, iyy = 0.01152, iyz = 0.0, izz = 0.0218)
- **Collision Properties:**
  - XYZ: (0, 0, 0)
  - RPY: (0, 0, 0)
    - Roll, Pitch, & Yaw
  - Geometry:
    - Mesh .dae file

Note that the geometry of the collision properties is set to the mesh file to most accurately reflect what a collision with the object would look like. The model's collision box does not fully morph to the shape of the drone. Instead a box is rendered around the outermost bounds of the model which are specified in the mesh file.

The Hokuyo lidar is mounted upside down, -0.097 meters below the drone body's origin point. As shown in Figure 3.1.2, the lidar maps out the environment across the 270° forward arc, shown by the multicolored lines.

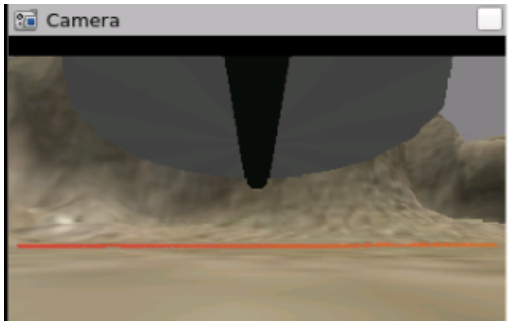


Figure 3.1.1: Example of Drone Camera in action. Displayed through ROS Debugging tool.

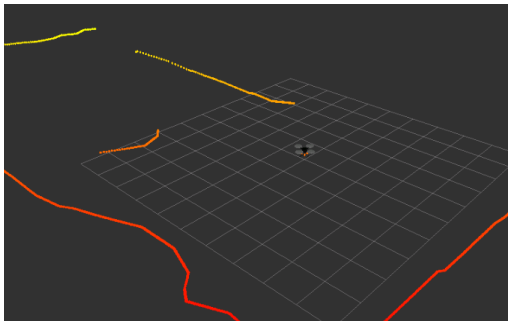


Figure 3.1.2: Example of Hokuyo Lidar mapping an environment at the current altitude.

Figure 3.1.1 shows the gazebo camera in action. The camera lens faces the same direction as the lidar sensor and is mounted just below it at approximately (0.05, 0, -0.06) meters in relation to the drone's origin. The camera generates images with a

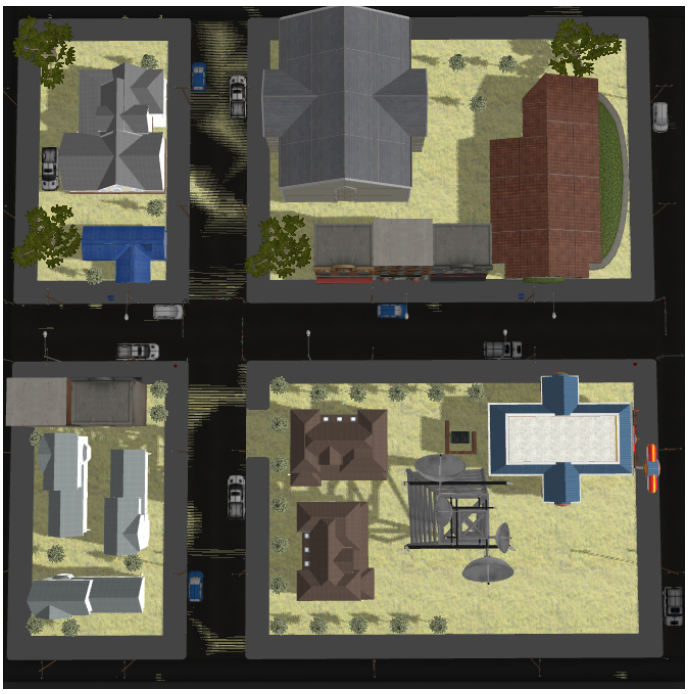
resolution of 320 x 240 pixels at a rate of 10 images per second. Increasing image resolution is possible, but we anticipate this may have negative effects on the efficiency of the overall system. Currently, the camera has no zoom functionality, so drones will need to be relatively close to the target in order for it to detect it.

The simulated lidar sensor shall provide continuous updates to the path planning system. This is dependent on the altitude the drone is currently at as the lidar does not allow for 3-d scanning by itself. We anticipate certain swarm algorithms may be able to counteract this limitation.

The camera will also feed its visual data to the object recognition system. If the BB8 target is detected by the YOLOv5 model, all drones in the swarm will be pinged that the target has been found.

3.3 World Specifications

There are two simulation sections available in the urban.world file.The first one consists of an urban environment approximately 89.41m x 91.12m and contains a variety of objects that emulate a small rural city block. We anticipate the drones to be spawned on either of the intersecting streets.



Overhead view of the Urban world.

The target model will be randomly placed within the environment, provided the following limitations:

- The object will not be placed within buildings
- The object will not be placed atop or within a tree or lamppost
- The object will maintain its texture across all positions
- The object will not be placed atop any office buildings or houses.

Overview of the Urban Environment. Note the Radio tower as this may interfere during the searching process.

The second world is built around a suburban environment. This environment contains no office buildings, but does have a single water tower. Depending on the efficiency of the path planning and swarm algorithms, the drones may be able to navigate through the open structure of the water tower.

3.4 Project Deliverables

Project Deliverables	
Title	**Description**
Design Document	The document shall detail the specific methods used to implement the software, how to reproduce it effectively, and reports on the effectiveness of the software. This document should number approximately 150 pages.
Single Drone	A single drone will be implemented within the ROS-Gazebo environment capable of executing basic teleoperated commands.
Drone Swarm	A collection of drones will be implemented with the ROS-Gazebo environment, all of which are capable of executing basic teleoperated commands. The swarm is about 4-5 drones in size and maintains a constant formation as it traverses the environment.
Drone Swarm Obstacle Avoidance	Each drone in the swarm shall be outfitted with a sensor that detects what objects the drone is approaching, if any. If it detects something, the drone shall adjust its trajectory, while maintaining awareness of its sister drones, to avoid colliding with the object. Once the object is successfully traversed, the drone will reposition itself within the swarm formation.
Object Detection	A machine learning model shall be implemented and trained off of a custom dataset consisting of pictures of the BB8 target in various areas, lighting, positions, etc. This model will be trained and tuned until a testing accuracy of 70% or greater is achieved consistently.
Path Planning	Algorithms that allow the drone swarm to plan a path through its environment shall be implemented. The path should take no more

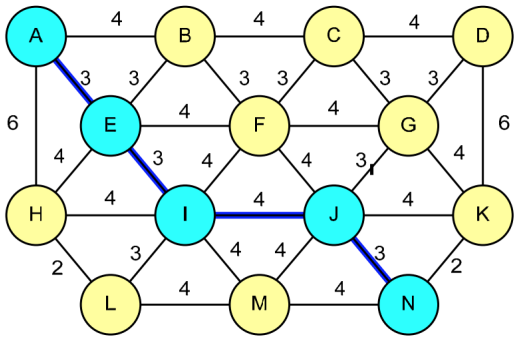
	than 3 minutes to traverse for the swarm in order to meet maximum time requirement of finding the target
Object Detection Swarm Integration	The object detection model will be integrated into the drone and fed real-time data from each drone sequentially. Once the target has been found, the swarm will cease following the path and move to the target's location.

4. Conceptual Discussions

4.1 Path Planning

4.1.2 A* Path Planning Algorithm:

A* Path Planning Algorithm is another path planning algorithm that calculates the shortest distance between the origin of where the robot is starting from to the goal destination state of where the robot will stop in an unknown environment. The A* Path Planning is an heuristic algorithm, which is an algorithm that doesn't sacrifice accuracy, speed and precision from traditional methods, and focuses on solving the problem in the fastest state, as well as a graph search algorithm. It will map out all the possible pathways for the goal state for the robot and select the shortest possible path out of all that is generated, and continuously generated while the robot navigates.



Example of A* Path Algorithm, represented as a Graph Node. [53]

Since the path planning algorithm is a graph search algorithm, each node in the graph are possible spots for the robot to navigate, and to add penalty for each pathway, is by calculating a cost to each line in the graph. The algorithm will calculate possible pathways by totaling the entire cost of each path and then backpropagating back and comparing with the current path of the robot.

Despite A* Path Planning is widely viewed for its strong advantages such as speed, precision, and accuracy from calculating the pathways; there have been better alternative search path planning algorithms such as Dijkstra's algorithm, since it can process the graph prior to moving the robot.

$$f(n) = g(n) + h(n)$$

Cost Function of the A* Algorithm

This is the cost function for the A* path planning algorithm. The following variables meaning:

- n means the next node on graph/path
- g means cost from origin node to next node (n)
- h means is a heuristic function that calculates the cost of the shortest path to goal node.

A* Implementation in ROS Package:
git clone
https://github.com/FernandoDorado/A-STAR-search-algorithm-implemented-in-ROS.git

4.1.2 Dijkstra's Algorithm

What is Dijkstra's Algorithm?

One of the most commonly used pathfinding methods in computer science was Dijkstra's Algorithm. Originally created back in 1956 by the Dutch Scientist Edsger Wybe Dijkstra, the notoriously greedy algorithm is used to find the shortest path between two points. In essence you can apply this method to a map, a graph data structure and everything in between when it comes to navigation. The reason this algorithm is considered greedy is because of the fact that greedy algorithms solve their problems by making local choices with the hopes that it will solve the larger problem. In this case Dijkstra will always pick the closest node in its path at the moment, and it will restart the process from there.

How does it work?

For explaining the mechanics of how Dijkstra's Algorithm works we will talk in terms of a graph with edges and vertices.

1. We start with an empty set where we will store the vertices that are a part of what the algorithm will consider members of the shortest path set. The condition that these vertices must have is that they must have a minimum distance from the source vertex.
2. Now to distinguish these distances there needs to be some distance associated with each vertex. At the beginning the start vertex will have distance 0 and every vertex from there will be infinite just for the start.
3. Now we run a while statement that breaks when the shortest path set includes all vertices in our graph.
    a. Pick a vertex V that is not already in the set and has a minimum distance value.
    b. Add V to the shortest path set.
    c. Update the minimum distance value of all the adjacent vertices to V. Iterate through all these vertices and if the sum of the distance is V and the adjacent vertex as well as the weight edge of V minus the adjacent edge is less than the distance value of V, then update the distance value of the adjacent vertex.

Once the graph has been processed the output will be a subgraph or set of vertices that belong to the shortest path set.

How to apply it in ROS?

Inside ROS there is a set of packages that are known as the ROS Navigation Stack. Inside this stack there is the ability to create maps that can be used with a path planning algorithm to create a robot's path. There will be more on the specifics of the execution process in another section of this design document to help demonstrate the process. However the big takeaway is that the ROS navigation stack uses Dijkstra's Algorithm as one of its path planning algorithms.

4.1.3 SLAM Concepts

SLAM also known as Simultaneous Localization and Mapping is a set of algorithms that allows the robot to generate a 2D or 3D map of an unknown environment and determine its location based on the pose of the robot. The map that was generated from the robot will continuously update based on the data given by the sensors.

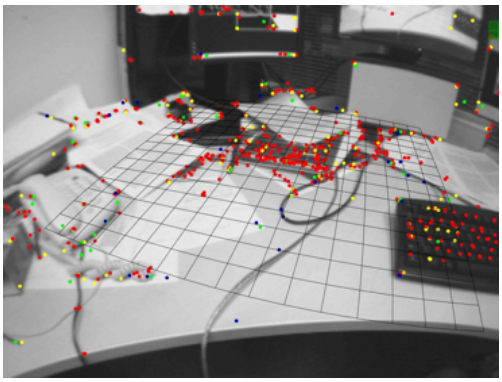
The LiDAR sensor that is used for SLAM, emits a laser that calculates the distance between the robot (origin) and the detection of the object. This is what will be used to generate the 3D or 2D map of the environment for the robot. Packages the team are interested in incorporating will be the gmapping package. The gmapping package is a ROS node that is a laser-based SLAM to generate a 2d mapping environment. Another package is incorporating the LiDAR SLAM for ROS that uses a cartographer algorithm from Google, this is useful since it can also generate the 3D and 2D mapping using OpenMP-boosted.

SLAM Algorithms

Visual SLAM

Visual SLAM also known as Visual Simultaneous Localization and Mapping, is a new approach in the computer vision industry in which it uses the input data from the camera and calculates the location and position of the camera while simultaneously mapping the unknown environment. The technology is still in its infancy, however it's quickly being adapted in the embedded vision market, but not ready for commercial use. A key benefit of adapting Visual SLAM in our drones is the only input data it needs is the camera, which keeps cost low as well as allowing greater flexibility for more sensors on the drone.

By tracking each frame with a set of points from the camera data, it can triangulate the 3D pose of the environment while generating the camera pose. Visual SLAM uses a solution called the bundle adjustment in which it minimizes the difference between the projected and the actual points.

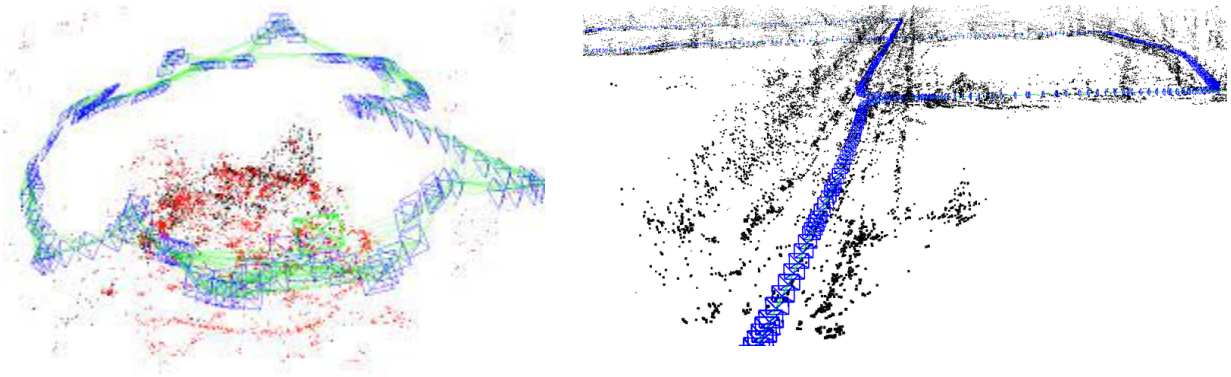


Example of Visual SLAM injecting 3D Camera Pose of each Object

Sparse SLAM

Sparse SLAM features of what encompasses SLAM such as the localization and mapping of an unknown environment however a caveat is that it does so with only sparse points from the features. Using a sparse point based SLAM, it allows a greater accuracy in the localization of the camera but is lacking semantic information from the

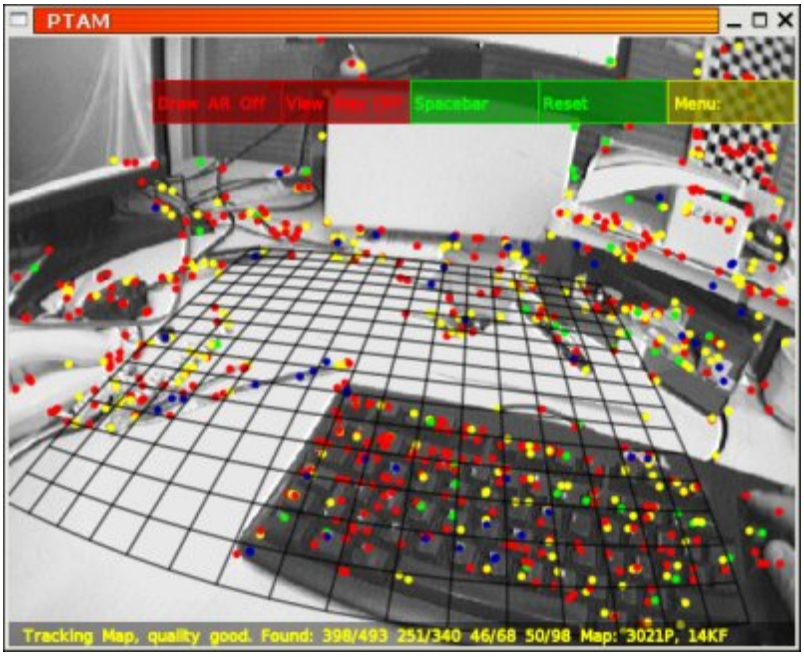
generated map of the unknown environment. Similarly as other SLAMs framework, it is also done in real time, and only needs a camera to compute such output.



Examples of Sparse SLAM Mapping of an Unknown Environment

Parallel Tracking and Mapping (PTAM) SLAM

Parallel Tracking and Mapping SLAM also known as PTAM SLAM, is another visual based SLAM that attempts to track the 3D position and localization of the camera from the drone and map the unknown environment in real time. It doesn't require any previous information such as prior weights of the unknown environment or any template to use. It can be used to figure out mostly any unknown environment out there by running through it multiple times. It has been commonly used in the Augmented Reality space but it is also used for drone navigation systems.

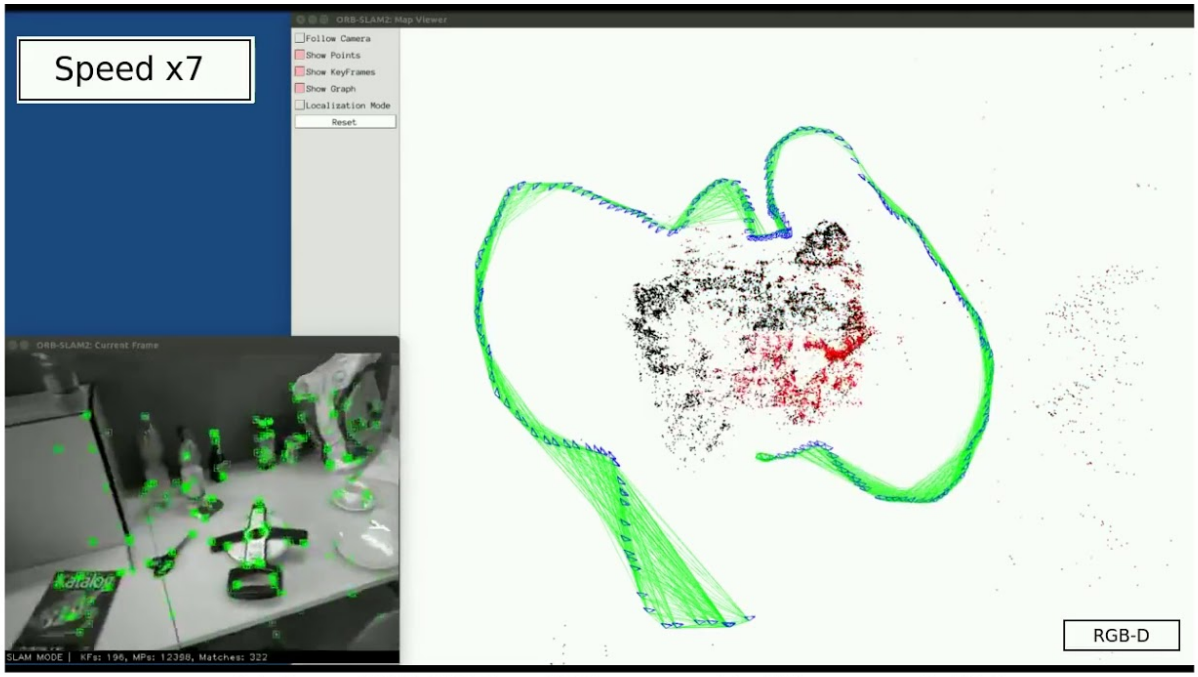


Example of PTAM, tracks the 3D Position and Localization from the Camera

ORB-SLAM

ORB SLAM is an open source repository for real time SLAM. ORB SLAM supports cameras that are monocular, stereo, and RGB-D, and it calculates the trajectory of the camera and creates a sparse 3D reconstruction in the unknown environment, which can support a variety of environments such as a driving around the city block or a drone navigating an unknown map.

ORB SLAM2 is the current open source repository that performs the same as v1 of ORB SLAM but also conducts a local Bundle Adjustment which minimizes the difference between the projected and the actual points. Both repositories conduct real time mapping solely of the input data of the camera.



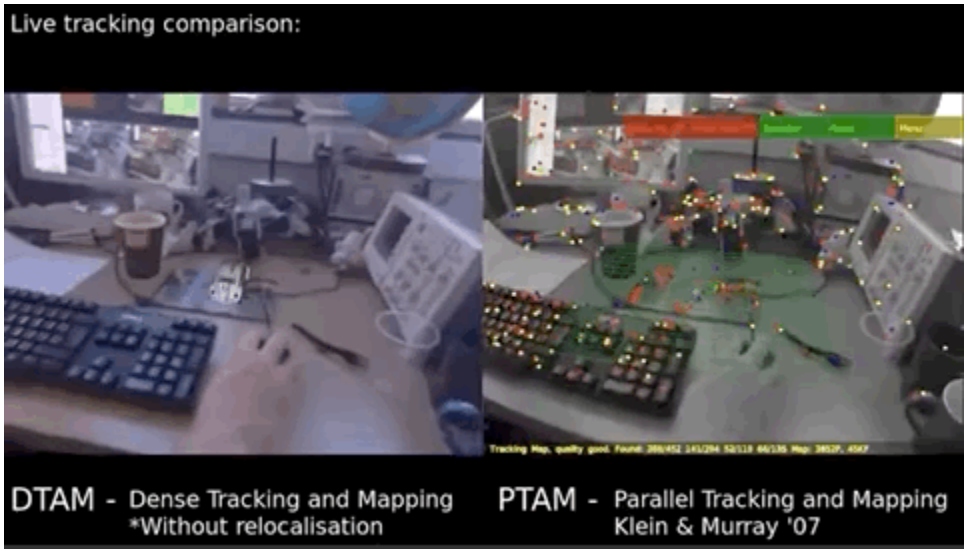
Screenshot of ORB SLAM generates a 3D sparse reconstruction map

Dense Tracking and Mapping SLAM (DTAM)

Dense Tracking and Mapping or DTAM for short, is one of the first real-time visual SLAM implementations in which it tracks and extracts every pixel in the frame. Rather than extracting only the relevant features of the frame to be less costly on the GPU, it is a more dense method, hence the method is called “Dense Tracking”. The method uses a lot of GPU computation, which might be one of the methods we will most likely not implement.

DTAM approach extracts every pixel of the frame until a keyframe of interest has been acquired, and then an initial depth map will then be generated from the initial values that were acquired. After the model generated based on the map, it will then be continuously compared to each successive frame, in which the map will be optimized and minimize the error from the original. After some time, the map will be dense in features and will be suitable for frames that are changing focus and motion blur.

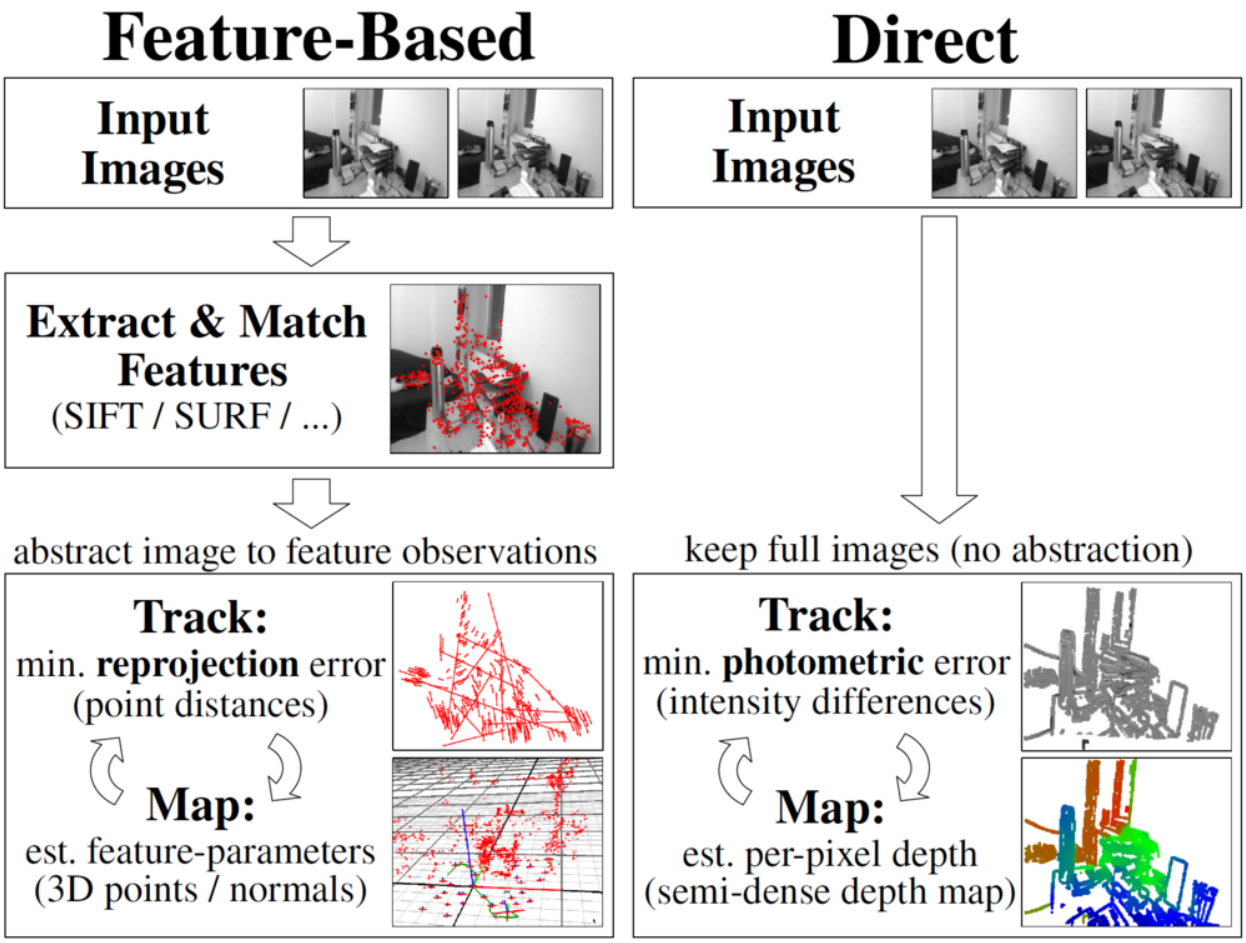
A comparison has been made with DTAM and PTAM, and it shows how DTAM has a much smoother and stable tracking of the object while PTAM continuously shifts and jitter with framerate issues. Also PTAM takes a sparse approach which only extracts features while DTAM extracts all pixels, which allows better tracking of the target even despite noise and motion blur.



DTAM vs PTAM algorithm approach

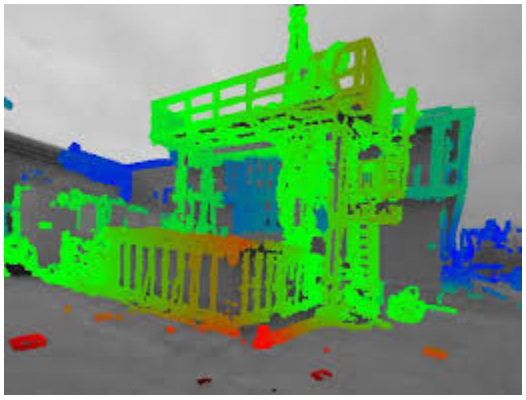
Large-Scale Direct Monocular SLAM

Large-Scale Direct Monocular SLAM or LSD-SLAM for short is a direct monocular SLAM approach. Rather than using features or keypoints such as DTAM, it will search for image intensity for both tracking and mapping. The RGB camera will track the environment with direct image alignment and the geometry will be calculated from semi-dense depth maps. The semi-dense depth map is generated by filtering over the pixel comparisons from the live feed camera. Afterwards, a Sim pose graph is then generated and will be used to correct the generated maps from the LSD-SLAM.



Feature-Based vs Direct SLAM

The Large-Scale Direct Monocular SLAM runs on a CPU over real time, which is one of the few approaches that doesn't rely heavily on the GPU similarly as DTAM. LSDM-SLAM uses a direct method that extracts all the information from the frames, which includes edges. This will allow a much higher accuracy and a stronger map with sparse environments and denser map.



Example of LSD-SLAM generating a high detail map

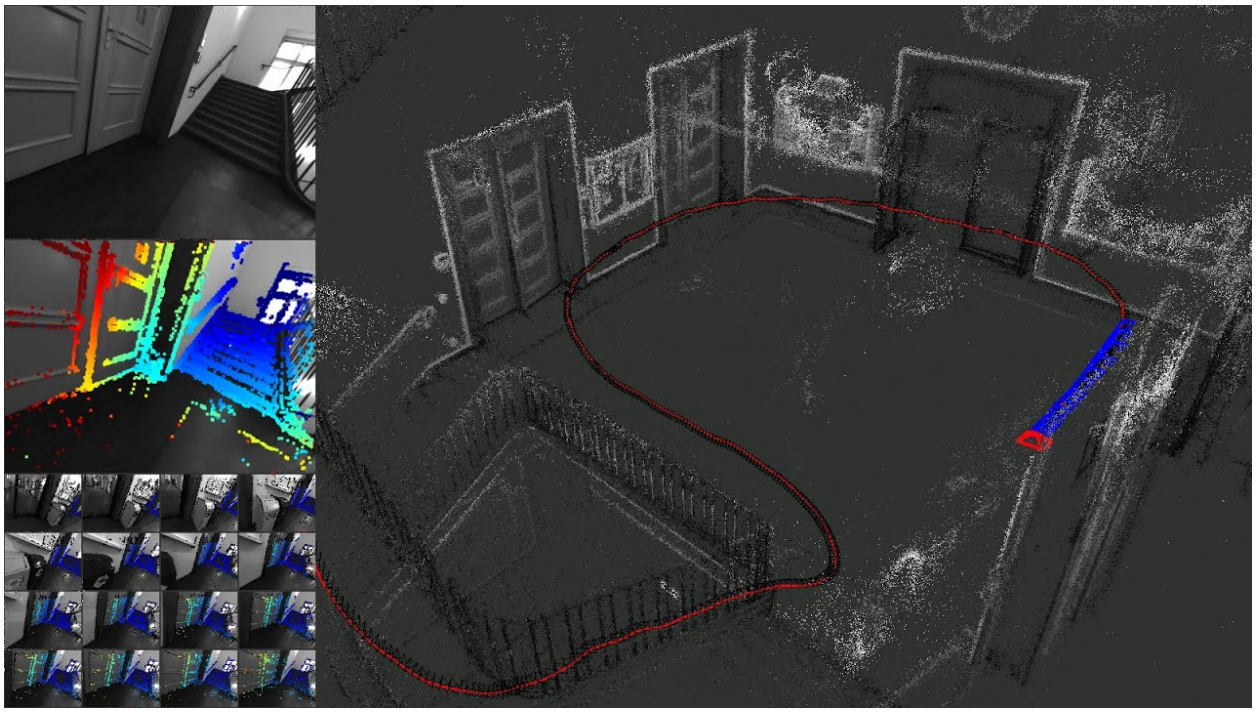
Direct Sparse Odometry

Direct Sparse Odometry also known as DSO combines two approaches in the SLAM model: Direct Probabilistic Model & Camera Motion. Unlike other SLAM approaches, DSO doesn't need any keypoint detectors or any descriptors, rather it gathers a sample of pixels from all the regions of the images that showed a great difference in intensity which will be used to compare with the frames and generate a map. The DSO benchmarks prove that it can outperform current state of the art direct and indirect methods in any real world environment while retaining high accuracy.

The following is used to install DSO:

Installation

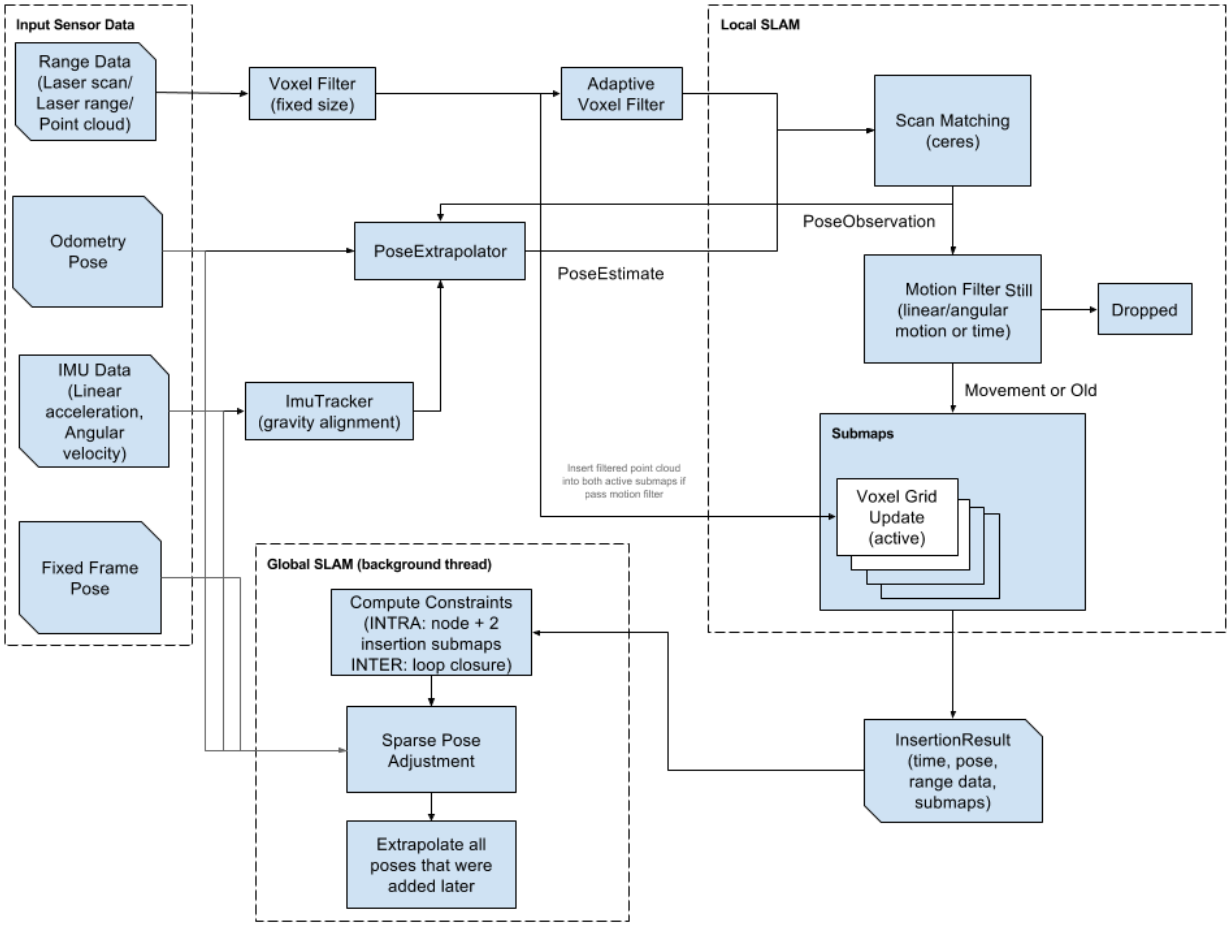
```
git clone https://github.com/JakobEngel/dso.git

sudo apt-get install libsuitesparse-dev libeigen3-dev libboost-all-dev
```



Example of Direct Sparse Odometry, generate map based on intensity of pixels

Cartographer for ROS

One of the main options for generating a SLAM both for local and global pathfinding in ROS. There is a specific build that is included with ros packages and can assist the robot in implementing SLAM with LiDAR. The current implementation is in 2D however it is possible to be adapted to 3D if needed, the user just needs to add the IMU sensor for this extent. Once the scans are complete there are two SLAM occurrences that contribute to the execution of the cartographer algorithm.



The algorithm layout in context of ROS for Cartographer. [28]

Local SLAM

After a scan has been performed and filtered with other range data, it is ready for the local SLAM. Now to insert the new scan into the submap that is generated from the local SLAM, the system uses pose extrapolation and scan matching. This is used to

make a best guess on where the next local scan should be performed and then inserted back into this generating submap. There are two methods of scan matching available in the cartographer ROS package.

- *CeresScanMatcher* - the method takes an initial guess to see where to scan next and how it fits into the submap. It is the faster of the two, but if there are errors it cannot fix them. If the sensors are well set up and the timing is fine, then using this method is a perfect solution.
- *RealTimeCorrelativeScanMatcher* - can be stacked on top of the CeresScanMatcher, however the context of use is when the sensors cannot be trusted. The best match generated here is passed back to the CeresScanMatcher as the prior scan. It is expensive to use but it pays off in diverse environments with a lot of features.

Global SLAM

While the local SLAM creates all the submaps, the global SLAM will act as a backend that takes these submaps as input. The submaps are rearranged so that it creates a comprehensive global map of the environment. Essentially the local SLAM generates puzzle pieces of a map and the global SLAM is the actor that puts the puzzle together. The global SLAM can be considered a “GraphSLAM” which is a pose graph that creates constraints between nodes and submaps then optimizes on those constraints. The resulting net is considered a pose graph according to the documentation provided by https://google-cartographer-ros.readthedocs.io/en/latest/algo_walkthrough.html. Constraints can be viewed in Rviz, which is helpful in tuning the global SLAM. There are two types of constraints:

- Non-global constraint - scope is of the submap not the global map and is responsible for local trajectory.
- Global constraint - ties submaps together by choosing the best fit that is close enough. This accommodates situations like a loop closure (where the last submap connects the free space into a loop).

Cartographer tries to limit the number of constraints when executing to reduce overhead. So when building, the algorithm will subsample based on a ratio constant that can be established in the parameters. Without this feature, global SLAM would run extremely slow.

4.2 Object Detection

4.2.1 What is YOLO?

YOLO is an object detection algorithm that stands for ‘You only look once’. It always builds off other specialized artificial intelligence frameworks. For example, YOLOv3 and v4 were built off DarkNet, while YOLOv5 is built off Pytorch. The algorithm divides images into a grid system and each cell in the grid is responsible for detecting objects, from a pre-trained dataset, within the grid.

YOLO is very famous as an object detection algorithm because it is fast and accurate in identifying objects. YOLOv5 was built mainly by Glenn Jocher from Ultralytics.Some other approaches to object detection are fast R-CNN, Retina-Net, and Single-Shot Multibox Detector. These all work but usually require many run-throughs of the algorithm to be accurate, YOLO beats these competitors out because it only requires one run through to be accurate. Hence YOLO being fast and accurate.

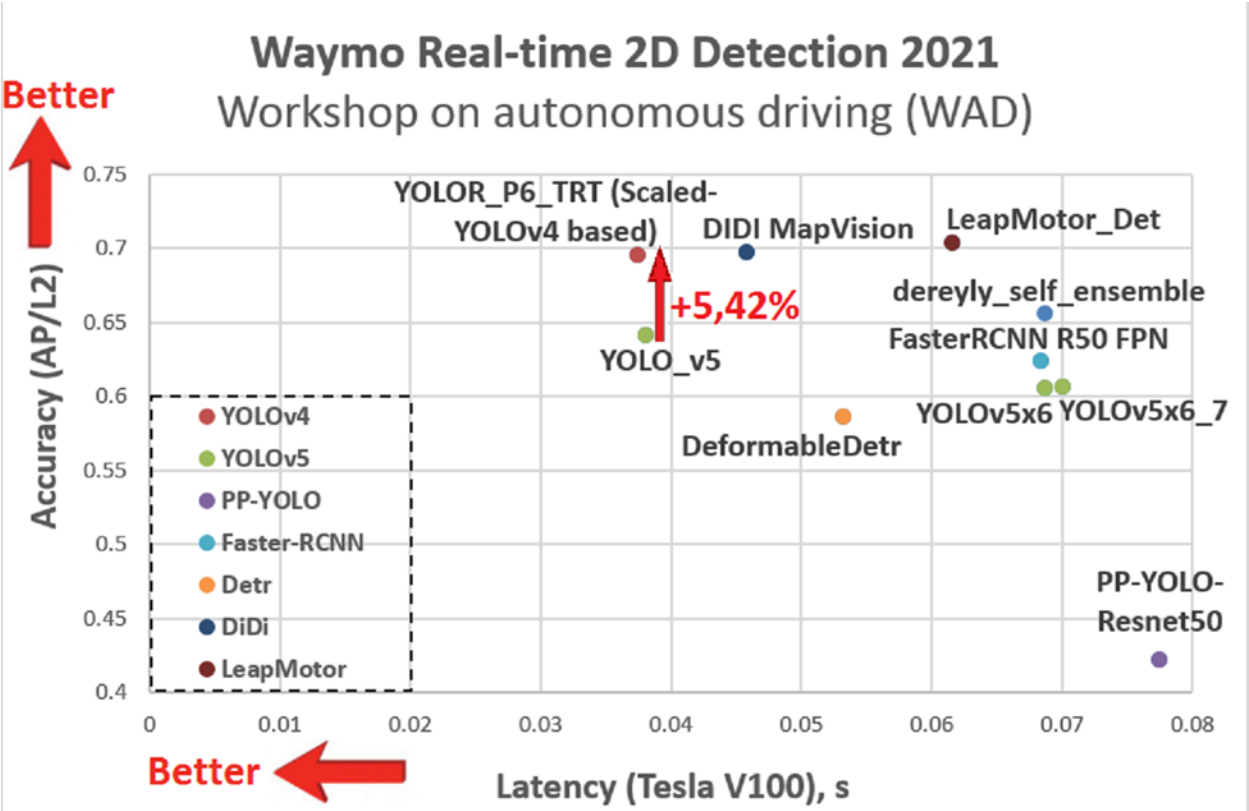
4.2.2 History of YOLO

YOLOv1 through YOLOv3 were all the brainchild of Joseph Redmon experimenting with the idea of object detection being done through the processes previously described. These versions of YOLO used DarkNet architecture, and they demonstrated a very fast and accurate process in object detection. After YOLOv3 was implemented Redmon no longer wanted to update YOLO because he did not like what his work was being used to accomplish. Thus, another person took on the mantle of this object detection algorithm. Alexey Bochoknovskiy took up Redmon’s work on YOLO and created YOLOv4. YOLOv4 was made by branching YOLOv3 on GitHub and continuously updating it to make it better all around. As it was branched from YOLOv3, YOLOv4 also used the DarkNet framework and was all around much better than YOLOv3.

While YOLOv4 was being made, Glenn Jocher was creating YOLOv5 using the same concept as YOLOv3 but using a different framework. YOLOv5 uses the Pytorch framework and is comparable to YOLOv4 but is generally faster. This version of YOLO is currently still being updated and made better than ever

4.2.3 Why YOLOv5?

Figure: Chart showing accuracy vs latency of different object detection algorithms in a real-time environment. [1]



To clear up any potential confusion about the legend in the diagram above, it is NOT saying YOLOv4 performs better than YOLOv5. However, it is saying “You Only Learn One Representation” (YOLOR) performs better than YOLOv5.

It would make sense to use the best performing object detector, however, as we have been doing research, we have grown to be more familiar with YOLO over YOLOR. Even though YOLOv5 has branched quite far away from the original YOLO coded in C, it keeps that brand recognition. As a result, users will prefer and gravitate to it due to its history. There is a benefit to that large user base as it means there will be more discussions related to YOLOv5 over YOLOR.

Documentation of YOLOR’s usage, unfortunately, appears to be lacking compared to YOLOv5’s. On the contrary, YOLOv5’s GitHub wiki contains a host of information detailing training tips to maximize precision.

The sheer amount of documentation and preference towards YOLOv5 makes it much easier to understand and work with, even though it performs slightly worse than YOLOR.

4.2.4 Justification of using YOLOv-5

In comparison to other networks, YOLOv5 is one of the fastest models out there, in accordance with speed and accuracy. YOLOv5 is an extremely fast model, it can run around 45 fps or even better when inference at real time for the larger model to over 155 fps on smaller model but of course sacrificing the accuracy. It is great at inference speed, detection of small or far away objects from a distance - which is extremely useful for our drone detecting and classifying objects in a far distance from targets - little to no overlapping of the bounding boxes, and as well really good at detection of over crowded objects.

Fast RCNN is another great network, however, in accordance to run speed and a very clean implementation of the bounding boxes, YOLOv5 has clearly shown from multiple tests conducting using the COCO dataset, that it 2.5 times faster than fast rcnn, as well having better performance than Fast RCNN. YOLO is also greatly used in ROS, with many plugins and packages that greatly support the model.

Not only is the YOLO model very fast, accurate, and widely supported in the ROS community, it is also very quick and easy to set up and get the model training and going. The only difficult part of training the model, as with all models, is gathering the custom dataset, creating bounding boxes, and then preprocessing the images. Fortunately, Roboflow handles the preprocessing for the training, which we just need to specify which preprocessing filter we want added and those filters will be added as new pictures to the dataset. Also, with bounding boxes, there have been multiple programs such as MATLAB automated bounded boxes generation, to Amazon Sagemaker bounding box personal support, to roboflow manual support. This is why generating and gathering the dataset should be a priority since it will take a long time to get everything set for training.

Overall, YOLOv5 and earlier models, proves it’s relatively fast, accurate, precise, cleaner - less noise on bounding boxes clutter - and very simple to implement as an object detection model than other object detection networks.

4.2.5 YOLOv5 Model Variants

There will be two important metrics used when comparing between models: speed, and mAP. Speed is in milliseconds and comes from an Nvidia Tesla V100 GPU with a batch size of 32. mAP is @ 0.5:0.95. This mAP value comes from an average of mAP values from a range of IoUs from 0.5 to 0.95 with steps of 0.05. Refer to the mAP section covered in object detection metrics for more information about its calculation.

A thing to note with these mAP values, by taking the mAP from a range of IoUs, this will lower the mean. Other models may appear to perform better, but it may be because their mAP value is taken at only 0.5.

Nano
- Speed – 0.6
- mAP – 28.4

Small
- Speed – 0.9
- mAP – 37.2

Medium
- Speed – 1.7
- mAP – 45.2

Large
- Speed – 2.7
- mAP – 48.8

Extra Large
- Speed – 4.8
- mAP – 50.7

These are the specs listed from the YOLOv5's GitHub repository. These metrics are useful when comparing models between themselves, but it is difficult to comprehend how it really performs in implementation. [4]

It would be best to start with the extra-large model and see how well it works. Since we have a minimum requirement of IoU > 0.5 and confidence > 70%, it would make sense to use a slower model and see how it performs in order to meet the minimum standards. If we are running into issues with processing speed and have accuracy to sacrifice, we can always switch to a smaller, but faster model.

4.2.6 Library/ Usage

Dependencies required for usage of YOLOv5:·

- Matplotlib
- Numpy
- Opencv-python
- Pillow
- pyYAML
- requests
- scipy
- torch
- torchvision
- tqdm
- wandb
- tensorboard
- pandas
- seaborn
- coremltools
- onnx

- onnx-simplifier
- tensorflow
- Tensorflow js
- albumentations
- cython
- pycocotools
- roboflow
- thop

To download YOLOv5 the following commands are required:

1. clone the git repository (github.com/ultralytics/yolov5.git)
2. go to the git repository's root directory
3. Install the dependencies YOLOv5 wants from the requirement.txt (this contains all the repositories mentioned above).

To change source of objects detected you would do the following command:
This picture is taken from the YOLOv5 docs to show all the accepted sources of input that it takes in.

```
$ python detect.py --source OPTION
```

Replace OPTION with your selection, to detect from:

- **Webcam** : (OPTION = 0) *For live object detection from your connected webcam*
- **Image** : (OPTION = filename.jpg) *Create a copy of the image with an object detection overlay*
- **Video** : (OPTION = filename.mp4) *Create a copy of the video with an object detection overlay*
- **Directory** : (OPTION = directory_name/) *Create a copy of all file with an object detection overlay*
- **Global File Type** (OPTION = directory_name/*.jpg) *Create a copy of all file with an object detection overlay*
- **RTSP stream** : (OPTION = rtsp://170.93.143.139/rtplive/470011e600ef003a004ee33696235daa) *For live object detection from a stream*
- **RTMP stream** : (OPTION = rtmp://192.168.1.105/live/test) *For live object detection from a stream*
- **HTTP stream** : (OPTION = http://112.50.243.8/PLTV/88888888/224/3221225900/1.m3u8) *For live object detection from a stream*

The following file formats are currently supported:

- **Images:** bmp, jpg, jpeg, png, tif, tiff, dng, webp, mpo
- **Videos:** mov, avi, mp4, mpg, mpeg, m4v, wmv, mkv

4.2.7 How does YOLO work?

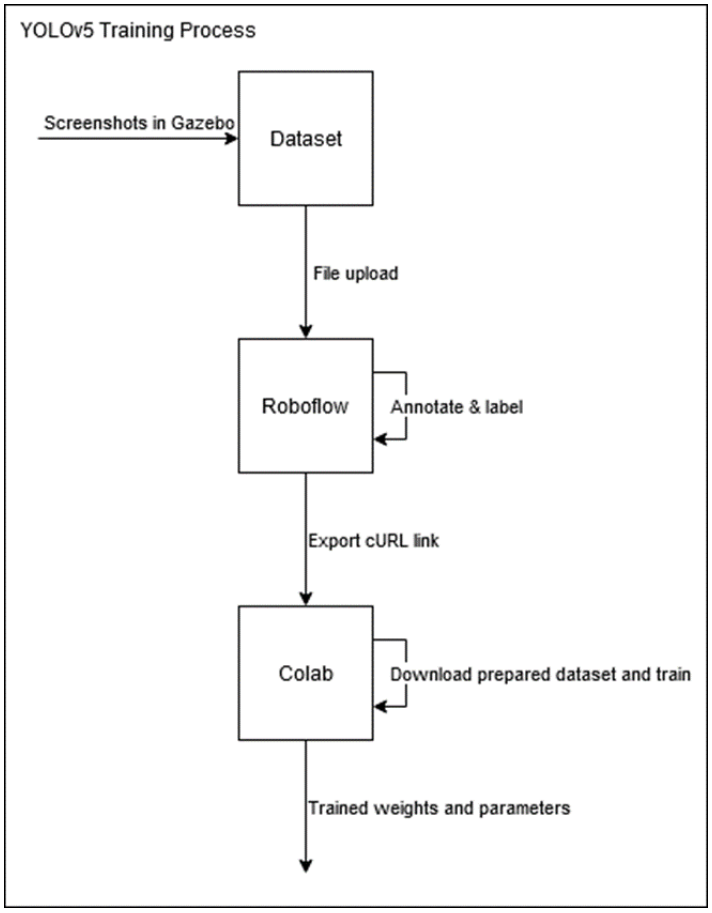
YOLO uses convolutional neural networks for real-time object detection. It tackles the problem using regression and provides the class probabilities of the detected images.

4.2.8 Training YOLOv5's Model

Overview

- Object detection target will be BB8
- Bare minimum > 50 images. Up to 500s would be excellent.
    - 70 : 30 training to test ratio
- Bounding boxes and labels need to be established
    - Roboflow
        - Complete API and pipeline for training in Colab
        - Comes with built-in annotation tools
        - Can export data to YOLO v5 PyTorch format
- BB8 model will probably need to be rendered in Gazebo and screenshots taken
    - Obstacles and backgrounds will need to vary because there is only one model we're using
    - Perspectives and distance also matter as the drones will be getting overhead shots
- Upload images to Roboflow
- Label images using Roboflow's native annotation tools
- Export in appropriate data format
- Train YOLO
- Evaluate model with TensorBoard

Figure: Diagram outlining the model training process.



Roboflow

Preparing our data for training will be a long and tedious process. Labeling is a mandatory process that every image needs to undergo. There are many different software solutions to accomplish that task. Local solutions would be something like “LabelIng” or “CVAT”. They would be excellent if we were planning on keeping everything local, though.

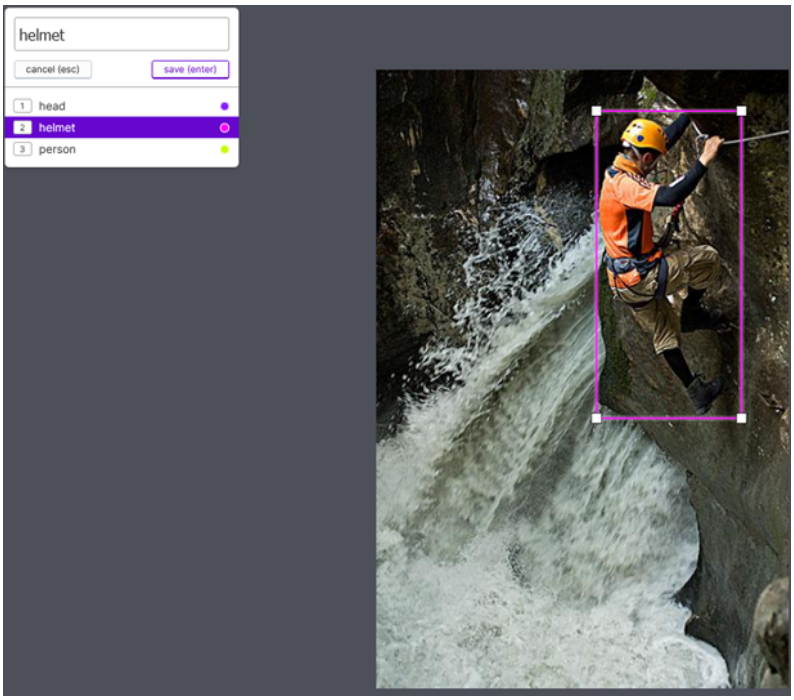
As for our workflow, we are more accustomed to working on the web. There is the additional bonus that multiple people could label if the data was on the web. Luckily, we are not dealing with any sensitive information, so there would be no privacy or security concerns of uploading our dataset online for labeling.

Roboflow is most likely our best solution for accomplishing the job. Image uploading can be done using their web user interface. It’ll be a nice and friendly interface to get our images on their servers. It’s as simple as dragging and dropping a folder straight into their interface. Alternatively, other team members can sign in and upload their own data into the collection.

Labeling on Roboflow

Labeling should be just as easy as uploading images to Roboflow. Once you are in the annotation tool, all you have to do is draw a bounding box around your object of interest and apply a class label to it. Labels apparently have hotkeys associated with them, so labeling is just that much faster.

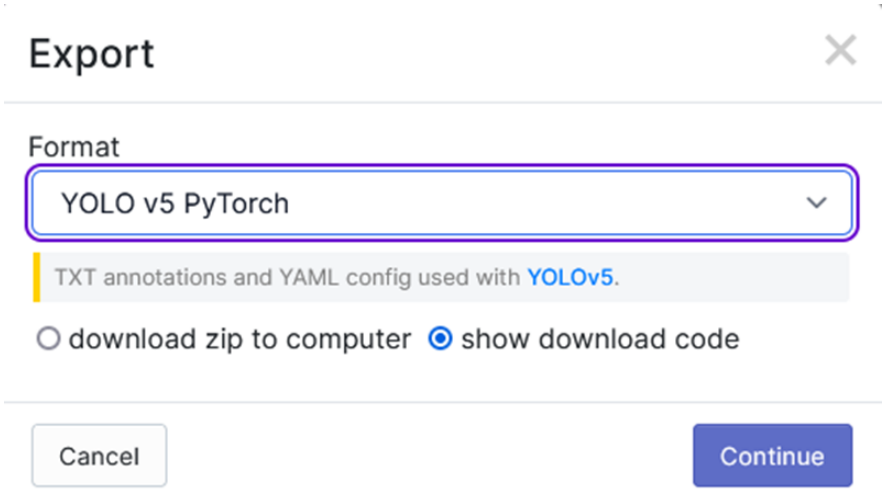
Figure: Screenshot obtained in Roboflow showing the labeling process. There is a menu of labels to the upper left and a drawn bounding box over the man in the right.



Exporting Roboflow Data

Roboflow has a nice export tool worth mentioning. It supports the common formats such as JSON, XML, TXT, and CSV, which can be used by a variety of machine learning frameworks. You can download your labeled dataset for local training or opt to keep it on their servers still. In the screenshot below, selecting “download code” will give you a curl link.

Figure: Screenshot obtained in Roboflow showing the export menu of Roboflow. It provides YOLOv5’s format and can export a download link.



The curl link can be used in Google Colab to quickly download the dataset off Roboflow and into your working notebook. The dataset should also be available if Roboflow's PIP package is installed into Colab. It appears that Roboflow's PIP does not really offer anything special over curl except through a slightly different workflow to access the data.

Training Environment Setup [5]

YOLOv5 itself will need to be installed into Google Colab's environment. This can be done with a simple "git clone". PyTorch will need to be imported, and all dependencies will need to be installed using PIP. The following command should handle the dependencies:

```
!pip install -qr requirements.txt
```

The dataset will need to be downloaded next. The option to use Roboflow's curl command or PIP package should be interchangeable. It will be up to implementation to decide which one to use, but it should not make a difference.

The YAML package might need to be imported to slightly modify the training YAML. At most, the number of classes will have to be modified, otherwise, the model can train with its default and have a lot of untrained classes.

Training

- img – input size
- batch – batch size
- epochs – training epochs
- data – path to imported .yaml
- cfg – model configuration

- weights – path to custom weights
- name – result names
- cache – cache images for faster processing
- nosave – save the final checkpoint

```
!python train.py --img x --batch x --epochs x --data data.yaml --cfg
./models/custom_yolov5.yaml --weights yolov5.pt --name results.txt  --cache --nosave
```

The training parameters are relatively straightforward, but there are a couple variables that can be tuned. Batch size and epochs will need attention during training, but that will be dependent on circumstances.

> According to YOLOv5's wiki, batch size is recommended to be the maximum size your GPU can handle, and epochs should start at 300. If the model does not overfit, epochs can double until it does. Then it can be backed-off to a safer value. [6]

Weights are also another variable that can be changed, but according to the wiki, leaving the training weight to be random is NOT recommended. Instead, just use the pretrained weights that they provide for you

TensorBoard

The TensorBoard extension can provide lots of information about the training such as mAP @ 0.5, mAP @ 0.5:0.95, precision, and recall. More information about these metrics can be found in their respective sections

Testing the Model

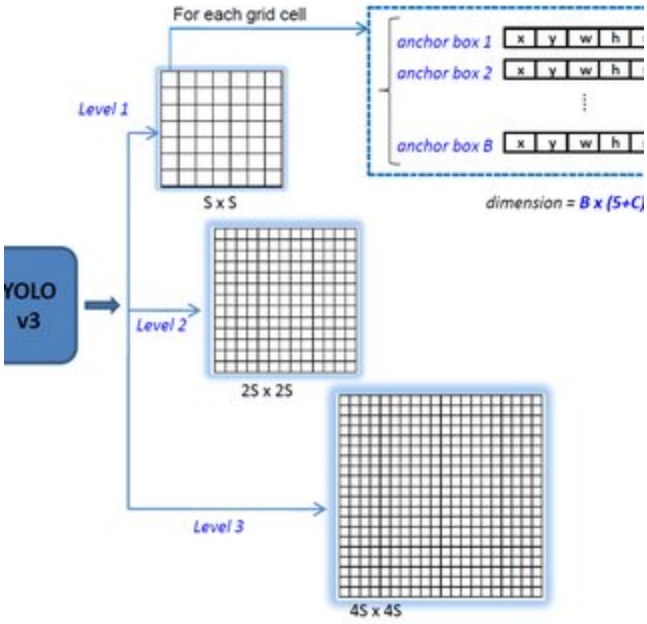
Once the model looks properly trained, it's time run it on the test dataset:

```
!python detect.py --weights best_weights.pt  --source /test/images
```

The images can finally be displayed for review. If everything appears correct, that means the model is properly trained and detecting well! The model is ready for export.

4.2.9 Residual Blocks

YOLO is given an image from some source and the image is divided into smaller grids. These grids are the same size as each other and every grid is responsible for detecting objects within it. This image[35] shows how YOLO breaks the image down into many boxes and will check each box for some object. YOLO will then go down further breaking the image into even smaller cells in order to have complete accuracy in finding objects.

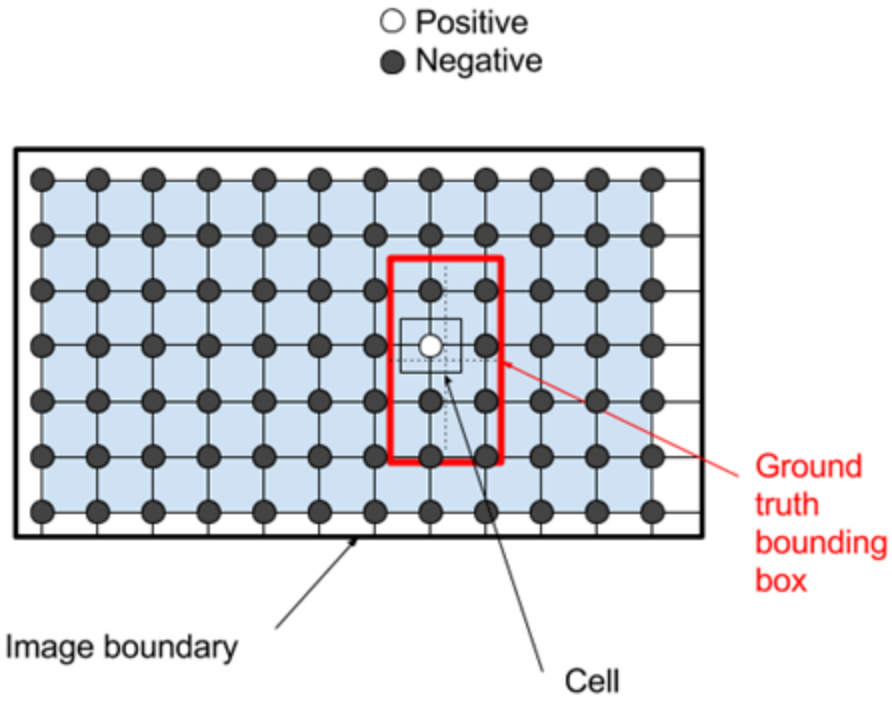


4.2.10 Bounding Box Regression

These are boxes that mark out an object from the given source image. Bounding boxes consist of a couple of important attributes:

- Width: Length of side on x-axis.
- Height: Length of side on y-axis.
- Class: What the object is supposed to be (person, mailbox, car, bb8, etc.).
- Center: The center of the box.

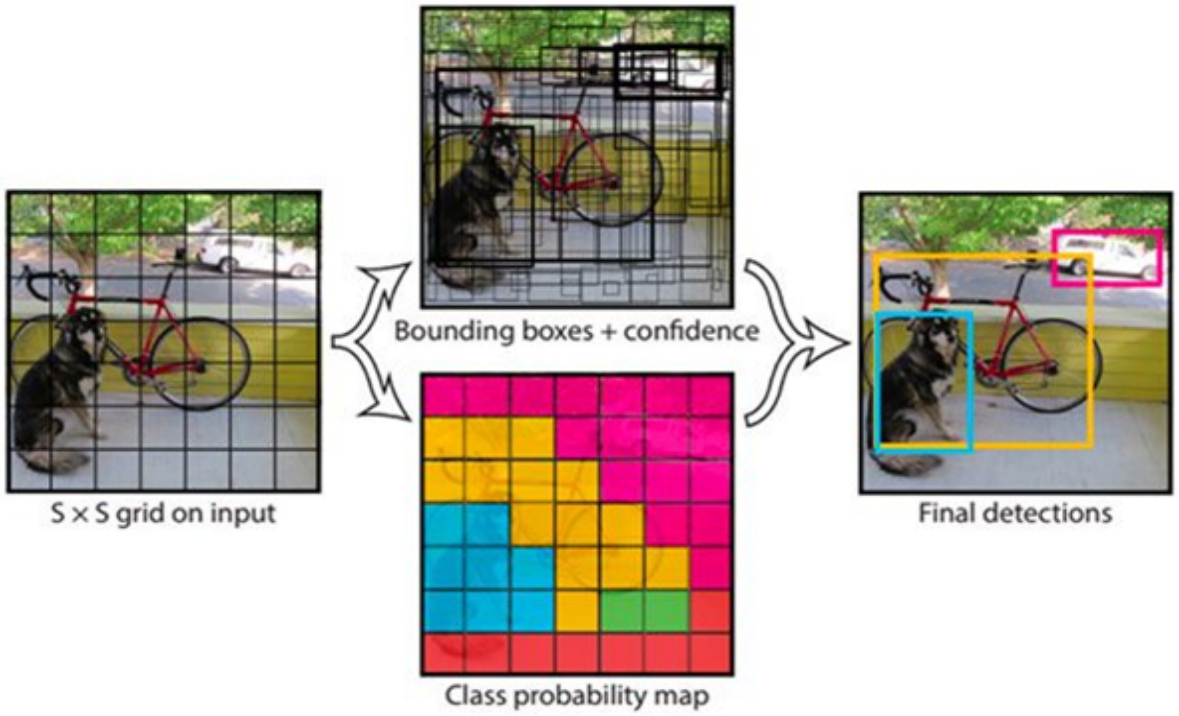
YOLO uses these boxes to predict the attributes given above and can then give a probability for how sure the algorithm is about this object. The following image[36] shows more about bounding boxes, where the cell in the grid that finds a hint of the object will put a box around the object, while cells that were negative will not react.



4.2.11 Intersection Over Union

IOU is an occurrence that happens in object detection algorithms. It describes how bounding boxes overlap. YOLO uses this to provide output boxes that surround objects very well. This is done by every grid cell in YOLO predicting the bounding box and confidence. If IOU is 1 then that means that the predicted bounding box should be the real bounding box. This is used to eliminate error in object detection because it will reduce false claims of what objects could be. Instead, it causes increased accuracy. For our specific project we need to have an IOU above .5 to be considered successful.

Here is an example[37] of this concept:



YOLO puts all these features together by the following steps.

1. Get an image and divide it into cells.
2. Each cell predicts bounding boxes and gives a confidence score.
3. The cells then predict class probabilities and guess the class of each object.
4. IOU will check the cell's bounding boxes and make sure that the boxes are equal to the actual boxes of the object.
5. Unique bounding boxes that fit the object perfectly are created.

4.2.12 Training

To train custom datasets:

1. Create a dataset.yaml file which contains:
    - Download command or URL for auto-downloading.
    - A path to a directory of training images or a path to a .txt file with a list of training images.
    - A path to a directory of validation images.
    - The number of classes
    - A list of class names

An example[38] of this that displays these aspects was displayed in the YOLOv5 docs which is shown below:

```
# train and val data as 1) directory: path/images/, 2) file: path/images.txt,
or 3) list: [path1/images/, path2/images/]
train: ../coco128/images/train2017/
val: ../coco128/images/train2017/

# number of classes
nc: 80

# class names
names: ['person', 'bicycle', 'car', 'motorcycle', 'airplane', 'bus', 'train',
'truck', 'boat', 'traffic light', 'fire hydrant', 'stop sign', 'parking
meter', 'bench', 'bird', 'cat', 'dog', 'horse', 'sheep', 'cow', 'elephant',
'bear', 'zebra', 'giraffe', 'backpack', 'umbrella', 'handbag', 'tie',
'suitcase', 'frisbee', 'skis', 'snowboard', 'sports ball', 'kite', 'baseball
bat', 'baseball glove', 'skateboard', 'surfboard', 'tennis racket', 'bottle',
'wine glass', 'cup', 'fork', 'knife', 'spoon', 'bowl', 'banana', 'apple',
'sandwich', 'orange', 'broccoli', 'carrot', 'hot dog', 'pizza', 'donut',
'cake', 'chair', 'couch', 'potted plant', 'bed', 'dining table', 'toilet',
'tv', 'laptop', 'mouse', 'remote', 'keyboard', 'cell phone', 'microwave',
'oven', 'toaster', 'sink', 'refrigerator', 'book', 'clock', 'vase',
'scissors', 'teddy bear', 'hair drier', 'toothbrush']
```

2. Create labels
    - Use a software to label all pictures that are being trained and make sure to export the labels to YOLO format (one .txt file per image).The YOLO format is that each row is an object. Each row is classified as ‘object_ID’ ‘x_center’ ‘y_center’ ‘width’ ‘height’.
    - The box coordinates are normalized xywh format from 0-1.
3. Organize directories

    For each image the path must be ‘/images/image.jpg’. It also must have a corresponding label of the path ‘/labels/image.txt’.This is because YOLOv5 goes into the directory and replaces images with labels in the pathing.
4. Select training model

    YOLOv5 has a couple different training models of differing speeds and accuracies, and one must be chosen to train the model on.
5. Train

    To train the following must be specified:
    - Dataset
    - Batch-size
    - Image size

- Pretrained weights or randomly initialized weights

An example of this is:
python train.py --img 640 --batch 16 --epochs 5 --data coco128.yaml --weights yolov5s.pt

This will then be saved to runs/train/exp1…2…3…

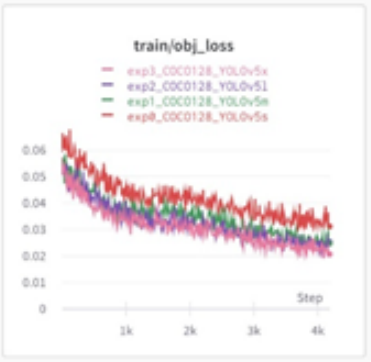
Visualization

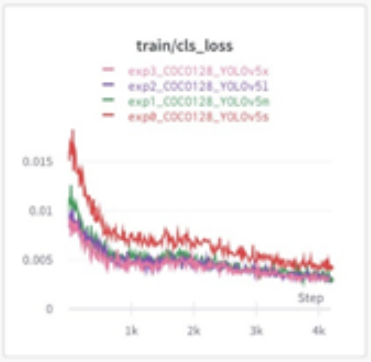
There are a couple of ways that can be used to visualize the training process. Local Logging: This is the normal way for visualizing the results given by the algorithm. It just logs all results to run/train with all its pictures and classifications on them. YOLOv5 will log the mosaics, labels, predictions, and augmentations of each picture. This provides for a very in-depth training progress.Metrics found during the training are logged to Tensorboard and will provide a custom results.txt which is plotted as a graph (results.png).

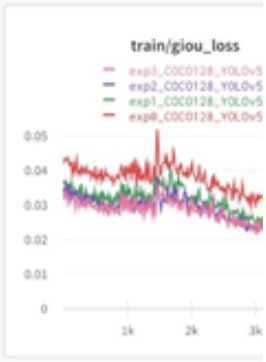
Weights and Biases Logging: To use must install wandb. But shows a very useful model of how the data is being trained. Many graphs are shown and give an inside look to the algorithm's view.

The picture below shows an example[38] of this visualization software and how much data it gives about the algorithm.

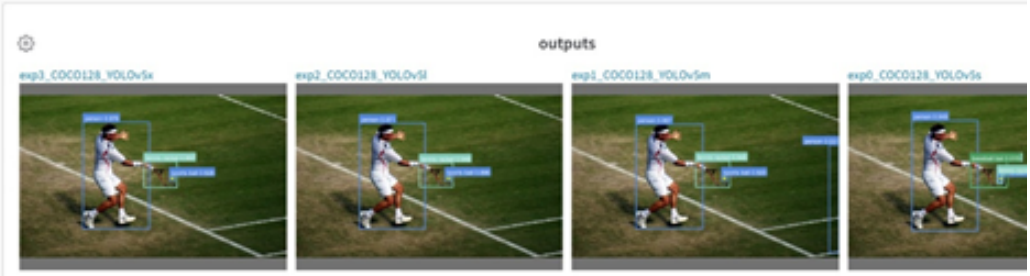
train 3

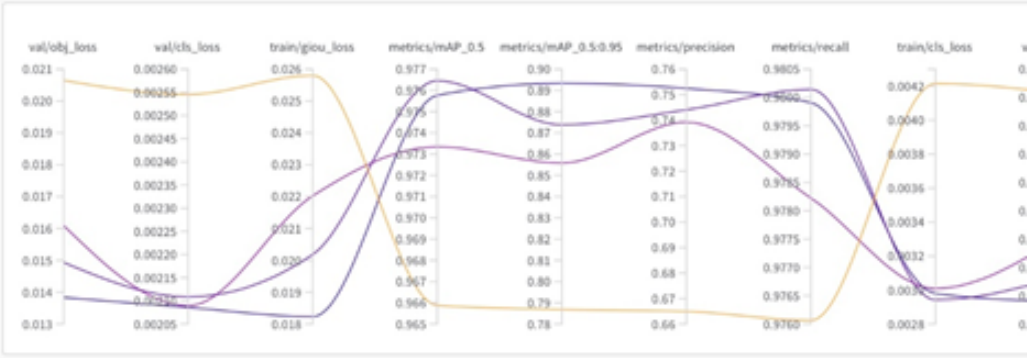




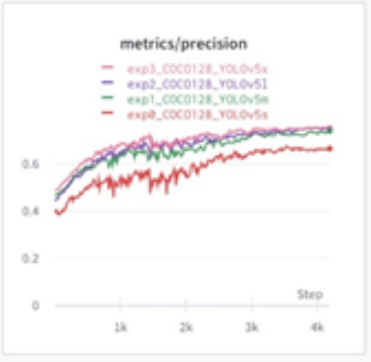


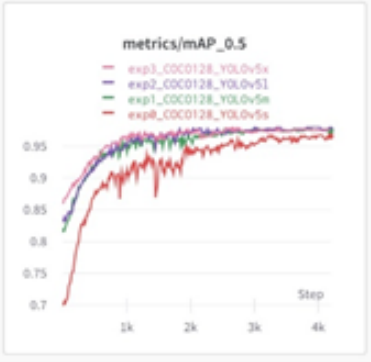
Media 2

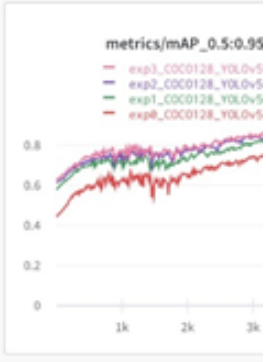




metrics 4



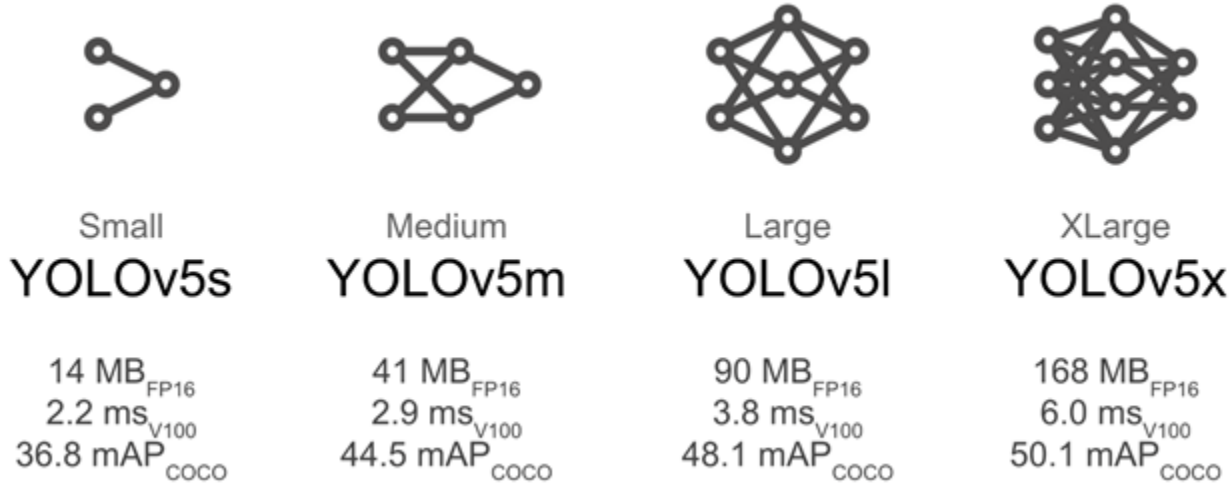




4.2.13 Models

When training data in YOLOv5 for object detection there are many training models to choose from.These models can either be large, accurate, and slow or small, fast, and less accurate.The models indicate how many neural networks are involved in the understanding of data when trained/ processed. The more networks, the slower the model will be, but it will be more accurate.When there are less neural networks involved then the model will be faster and less accurate.

YOLOv5 uses this family of compound-scaled object detection models trained on the COCO dataset. These models include functionality for Test Time Augmentation, model ensembling, hyperparameter evolution, and exporting to ONNX, CoreML, and TFLite.

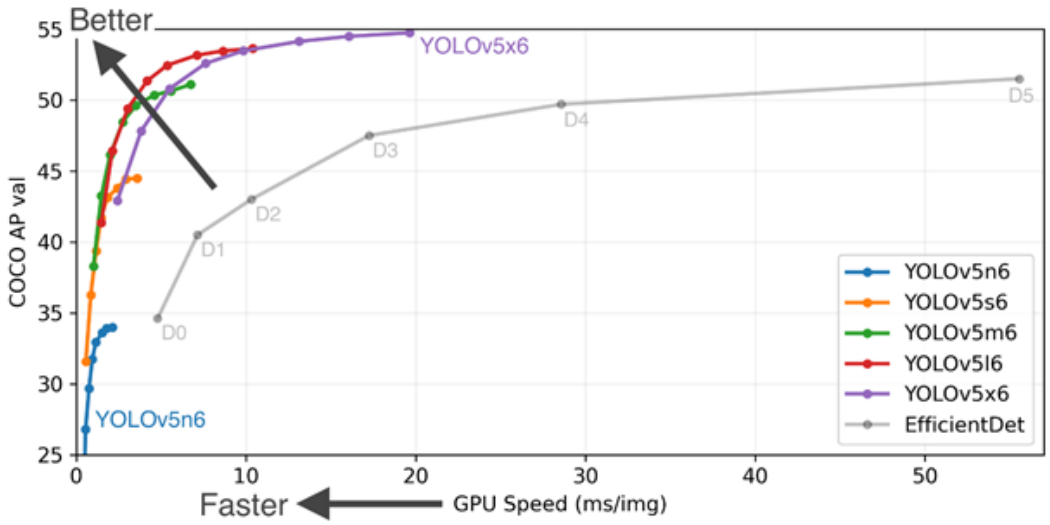


The diagram above[38] was taken from YOLOv5 documentation showing the models' performance at a glance. The picture shows how large the model is, the speed it takes to get results, and then the mean average precision which shows how accurate the model is.

As for the chart below[38], it shows a much deeper understanding than the previous diagram for the specs of each modeling system.

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 -

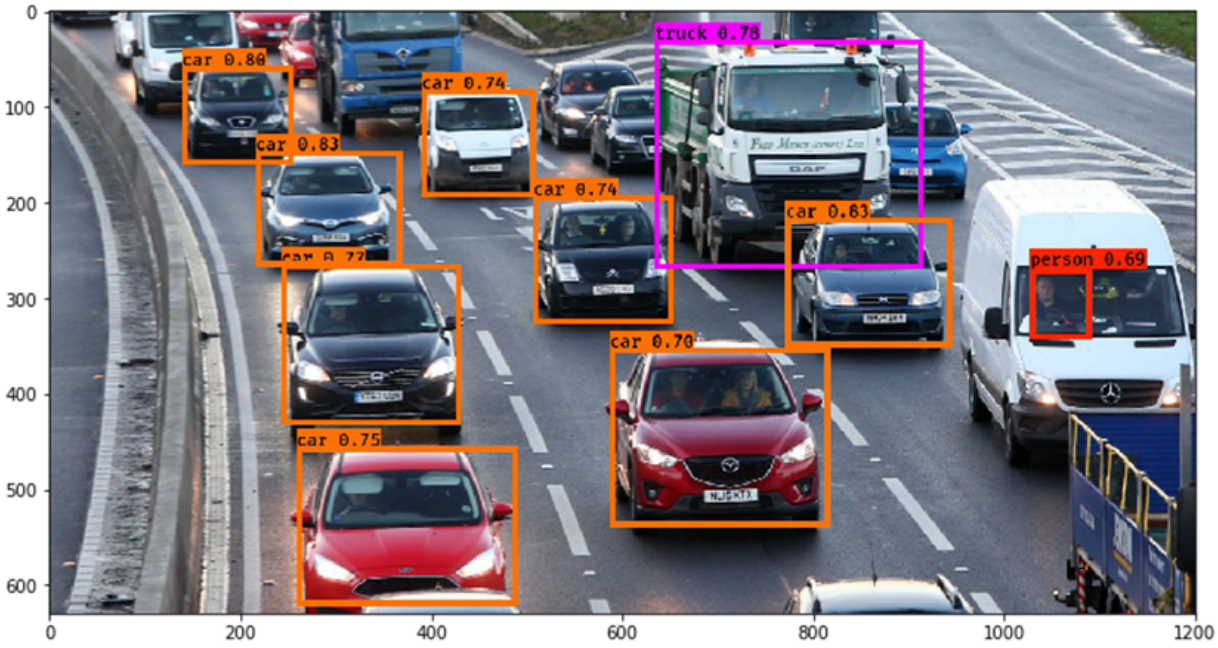
The graph below also shows the data from the graphs plotted to allow for easier viewing of performance between the YOLOv5 models [38]. As the graph shows, the greater the average precision is, the higher the time it takes to process the data.The graph was taken from the YOLOv5 documentation and is a great visual for seeing the trends among the models.The graph also shows a baseline of an efficient object detection model and every version of YOLOv5 models beat the baseline in both speed and precision.



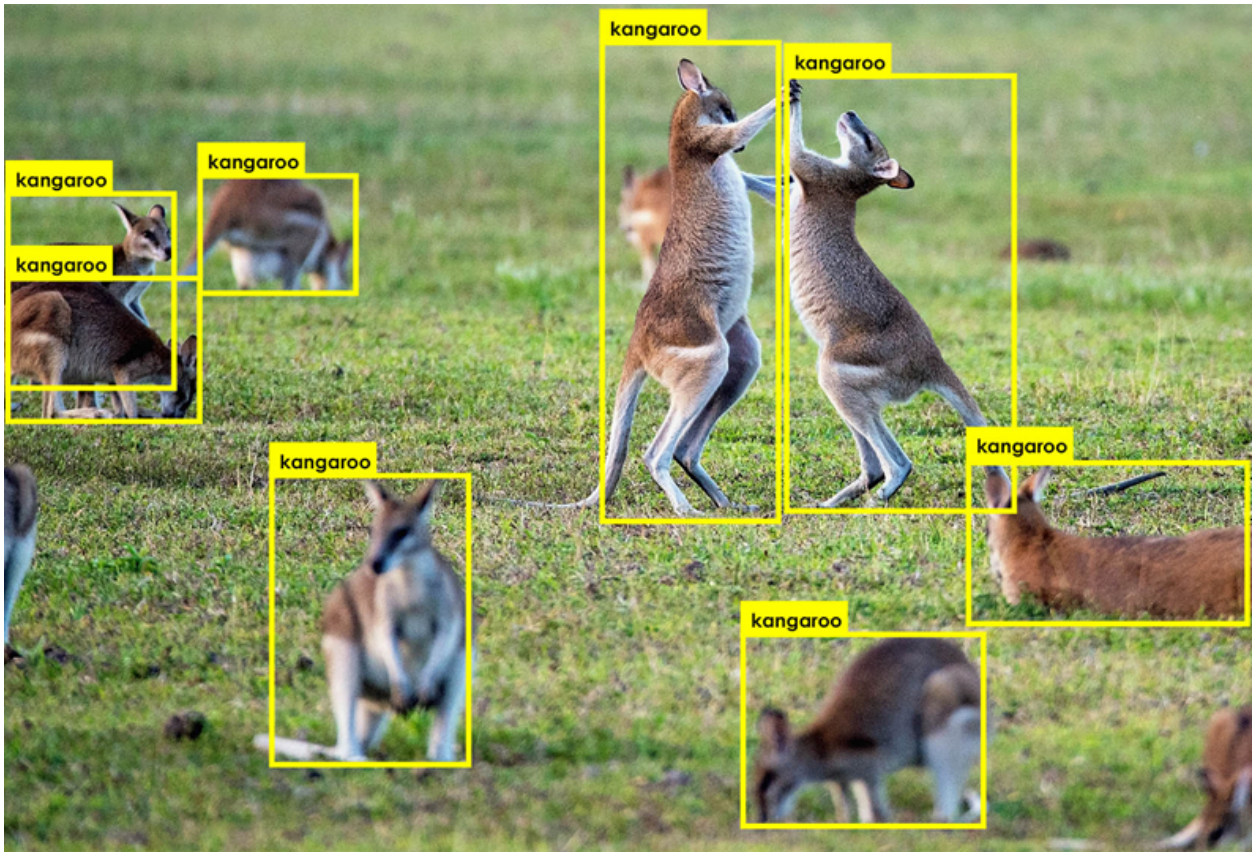
4.2.14 Examples of YOLO

YOLO is widely used as a very fast and accurate object detection model framework. Here are some examples of YOLO being used in the real world:

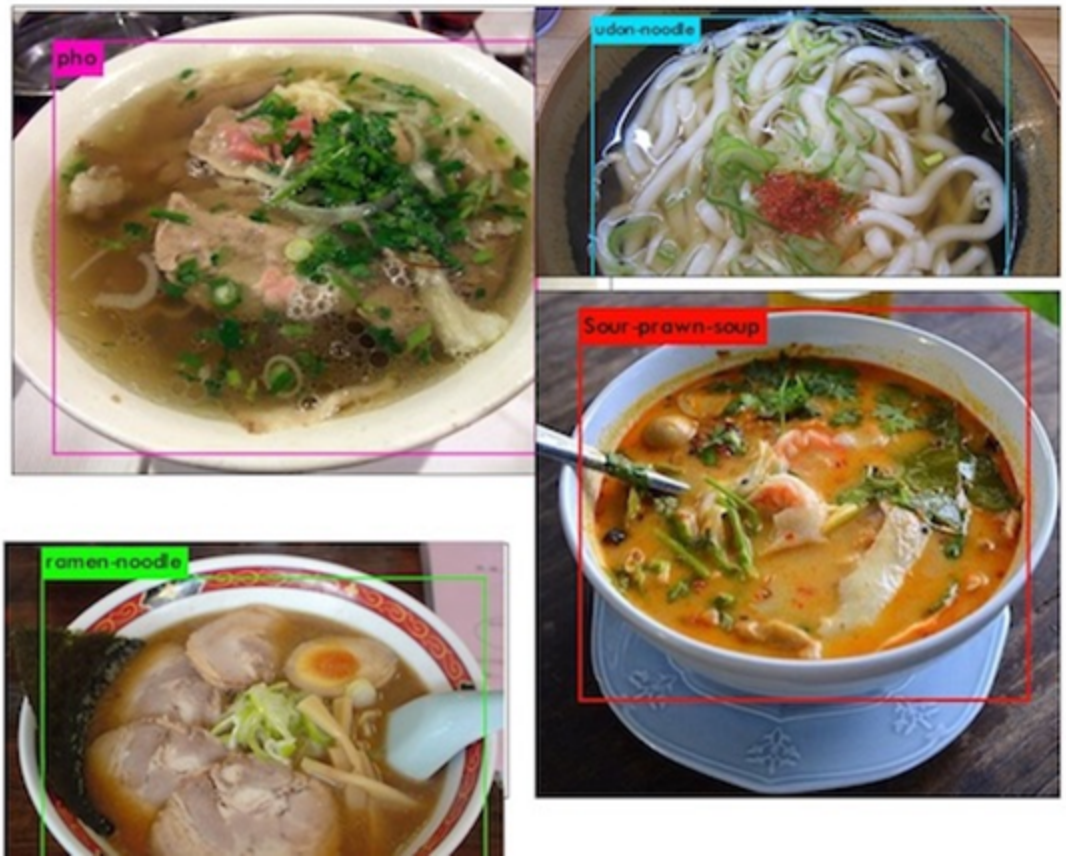
- Vehicle detection
    - YOLO has been used to monitor roads and detect the types of vehicles driving on them. It can tell the difference between cars, trucks, motorcycles, and other types of vehicles. YOLO can also try to detect license plates on the back of cars.
    - The usage of this is that YOLO can be used on red light cameras or to monitor highways for wrecks or truck stops or just for data collection purposes.
    - Vehicle detection has been very successful with using YOLO because of its quick real-time object detection from a camera feed. The picture below shows this scenario in action:



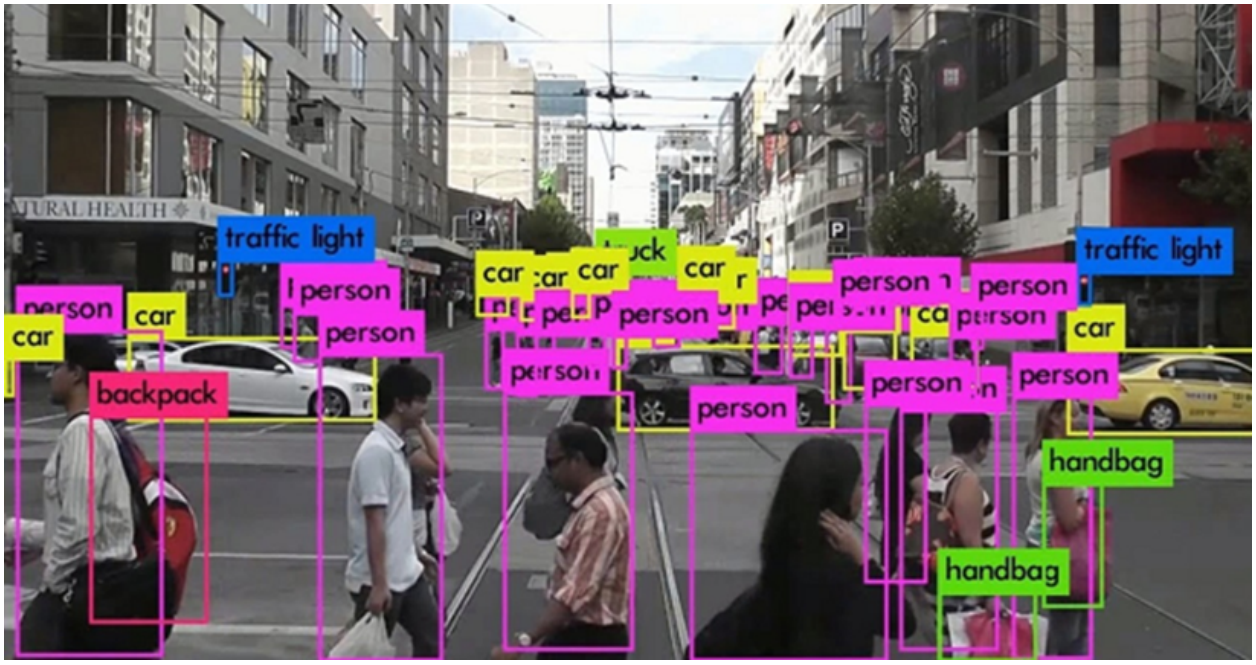
- Animal detection
    - YOLO has been used to differentiate between animals. It can differentiate between cats, dogs, horses, elephants, birds, lizards, and more.
    - The usage of this could be in home security systems, where the system can alert the owner of the house that a bear or raccoon is outside the house. It could also be used for environmental monitoring by collecting data about the types of animals that come through certain areas in a day.
    - Here is a picture of YOLO being used for detecting animals:



- Fruit, vegetable, and food detection:
    - YOLO can be used to tell the difference between different foods growing or not, such as oranges, apples, sandwiches, cake, and more.
    - Some of the usage of this could be monitoring a field of crops and notifying the farmer that the crops are ready to harvest. Other usages would be for more accurate data collection about what types of food are eaten in certain rooms.
    - A picture of YOLO being used to detect foodstuffs is below.



- Human detection:
    - This is the most controversial subject that YOLO has been used to detect, but also probably the most useful. It can differentiate humans from other objects and if trained enough it can differentiate a person from other humans.
    - This has the most amount of possible usage, such as monitoring areas to get data about business of an area at some time. It could theoretically be used for catching criminals and notifying some operators. It can be used for many different data collection scenarios. The versatility of YOLO makes it very easy to put it in real life scenarios.
    - Here is a picture of YOLO detecting a bunch of different people:



All of these usages of YOLO make it very versatile and widely used in object detection. It's speed and correctness make it very suitable for many real-time situations.

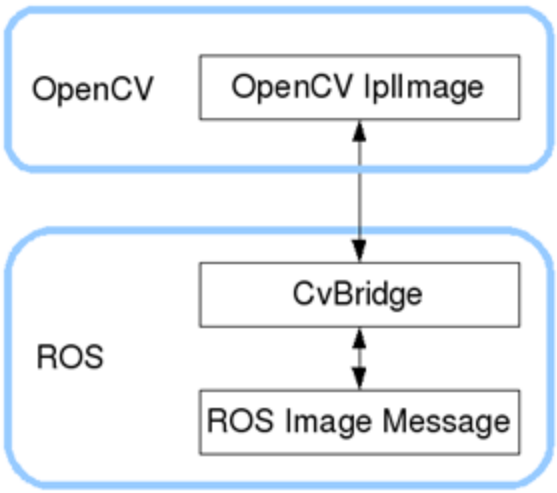
4.2.15 OpenCV

What is OpenCV?

It is an open source computer vision library that can be implemented with ROS in order to achieve object detection through the virtual sensor(camera, lasers, etc.) input. It includes many algorithms that other computer vision and object detection libraries use, such as YOLO.

How does OpenCV work with ROS?

ROS has a library named CvBridge that provides an interface between ROS and OpenCV, this bridge allows the programs to communicate accurately and effectively. This conversion is done by ROS taking images and posting them to sensor_msgs/Image. From there, CvBridge converts the image files from the normal ROS image to the required OpenCV file format and it takes over from there allowing for any image processing needs. This process is described by the image[39] below:



To use CvBridge, the CVBridge library must be imported from cv_bridge within the Python code. The code that accomplishes this conversion is below:

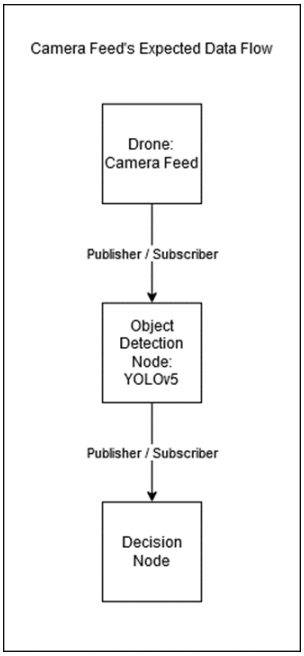
```
from from cv_bridge import CvBridge
bridge = CvBridge()
cv_image = bridge.imgmsg_to_cv2(image_message, desired_encoding='passthrough')
```

To use OpenCV, the cv2 library must be imported within the python code.

4.2.16 Data Flow for Single Drone

1. Drone will normally be publishing its camera feed to its respective topic
2. Object detection node will be subscribed to the camera feed topic
3. Object detection node will publish results in a custom topic
4. Decision node subscribes detection node's output

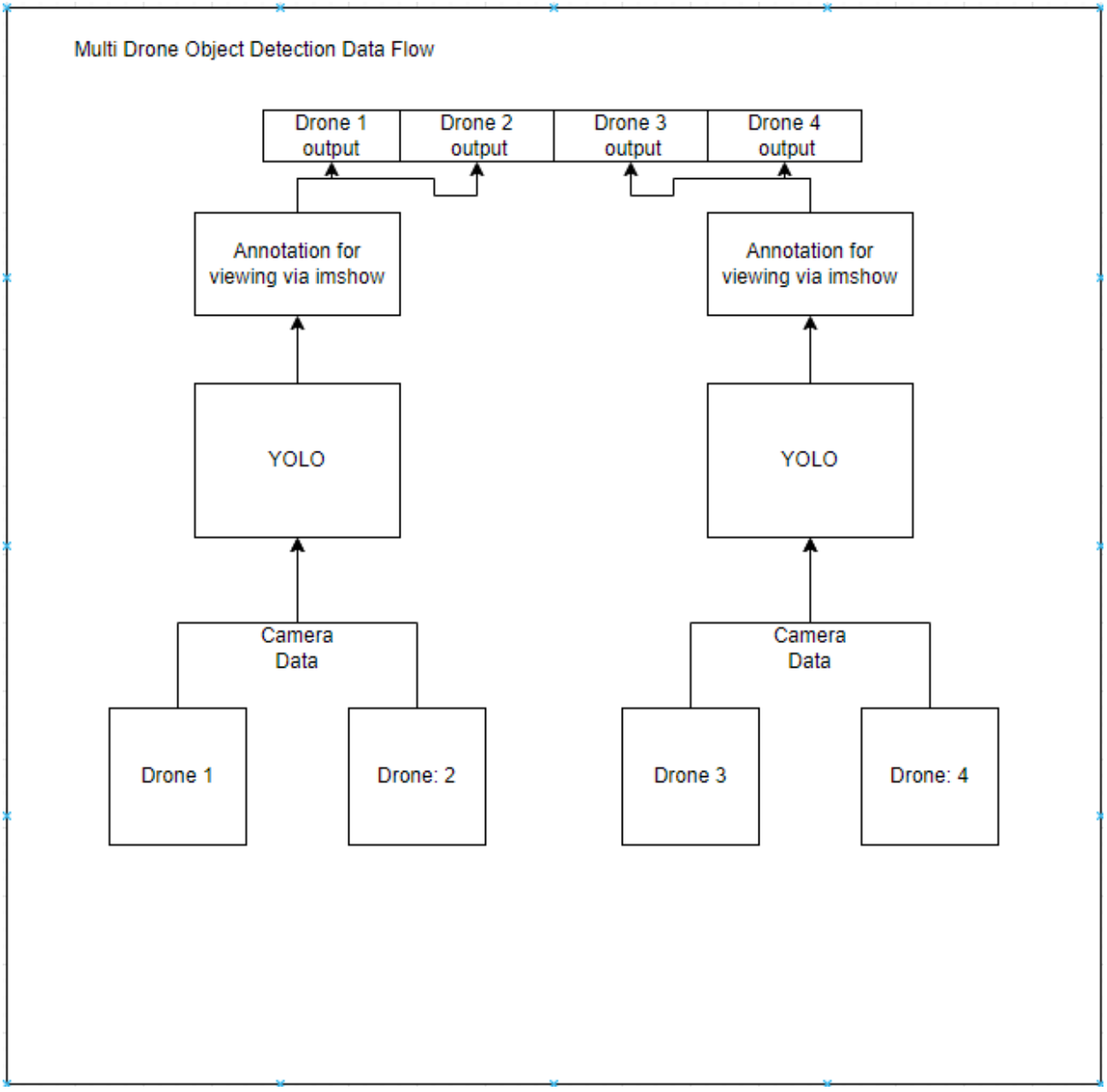
Figure: The planned path for data coming from a drone's camera to the object detection node and decision node.



4.2.17 Multi-Drone Object Detection

There are many design options for structuring nodes. The following diagram will provide an overview of the decisions discussed and made in the following major section.

Figure: How multiple drones will interface with object detection nodes



4.2.18 Broad Discussion About Implementing Object Detection and Design

Getting camera data out of drones?

The current idea is to use publishers and subscribers to topics as the main way to transfer data out of the drones as topics are best used with continuous data flow. Services and actions are ill-fitted for this job. Since there will be multiple drones that need to send data, the first decision needs to be made here. Do all drones publish to a single topic and provide identifying information, or do the drones publish to their own topic? In this case, it would most likely be best if they publish to individual topics, so the object detection node(s) does not need to sort the feeds in software.

Topics will most likely contain a simple prefix to provide identification:

/drone_1/camera_feed

/drone_n/camera_feed

Number of object detection nodes?

The next thing to discuss is the number of detectors. Will there be a single master detector, or will each drone have its own detector? Since there will be multiple topics, a single detector node will have to sort through all the camera feeds in software. Also, if this was implemented in real hardware, a single detector would not be capable of scaling up with multiple drones. Every drone should have its own personal detector.

This approach will make coding faster as there will be no need for a single detector to distinguish which drone sent what message and send a response back to the appropriate channel. Instead, one detector can be coded and duplicated. The duplicated detectors will just need a slight modification to subscribe to its corresponding drone topic. Then, a launch file can launch all detectors at once.

Where to send the detection outputs?

Originally, there was an idea to send the outputs to a master decision node that could tell the drones to converge to a location if the target was detected. However, that directly conflicts with the core principle of swarm behavior. Each drone should be capable of decision making without a central node doing so. In this case, it would make sense to give each drone its own decision node instead. The detector node should publish its outputs for a decision node to subscribe to.

It could also be argued that the results should be published directly back to the drone but giving the drone a decision node should not make a difference in practice. From an organization perspective, the drone would be better off viewed as a piece of hardware. A dedicated decision node is better off functioning as the brain of a drone.

Decision making based on detectors' outputs

As the drones are traversing the map, the aerial view of the BB8 model will present a great challenge for object detection. Depending on how the mode is trained to handle the initial aerial view, detection confidence can vary greatly. An idea is that if a drone "suspects" a target to be BB8, it should lower elevation and fly closer for a better view. There should be a minimum detection confidence before a drone breaks routine and flies closer, otherwise it will constantly be breaking routine to look at false positives.

This behavior of breaking routine should be performed as a ROS action as it is non-blocking and can be canceled.

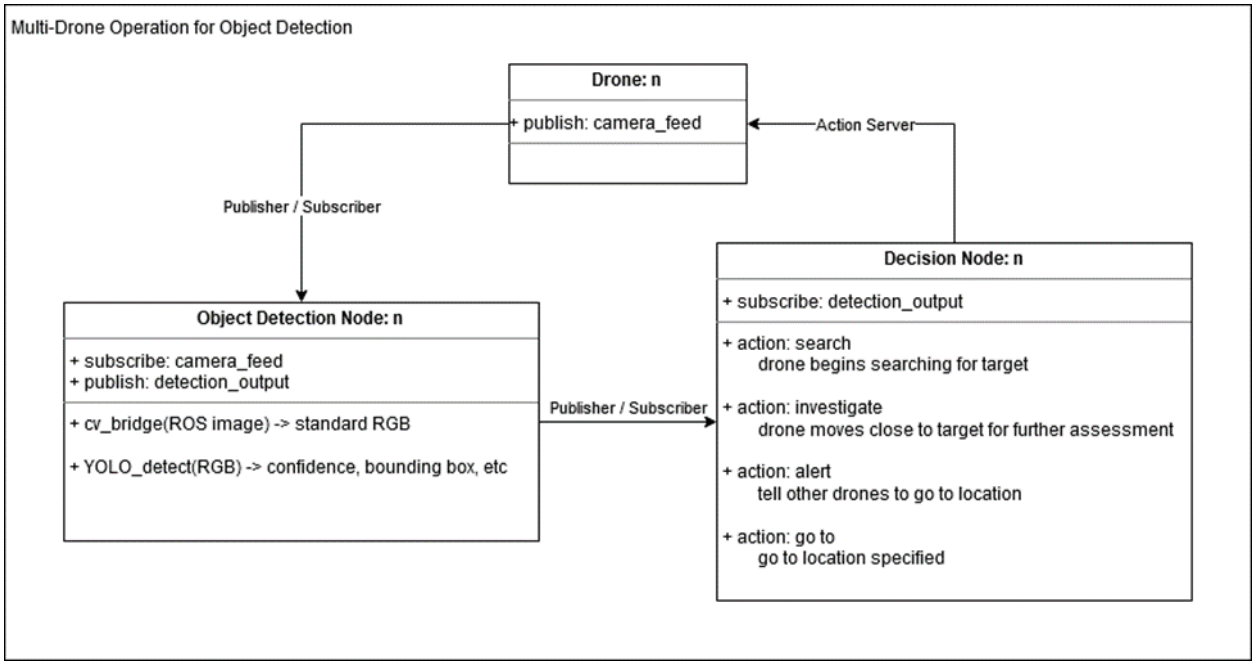
Breaking routine to further assess a target

If a drone breaks routine to get a better look at an object of interest, ideally, it would perform advanced movement and fly 360 degrees around the object to get a better view. If the detector is robust, advanced movement may not be necessary. As the drone reaches a comfortable height and distance away from the target, detection confidence should increase if the suspected target is BB8. There should be a brief timer once the drone reaches the target, where it further assesses the target. If the detection confidence does not reach > 70% and the timer expires, the drone should resume its searching routine.

Swarm behavior upon finding BB8

If confidence reaches > 70%, BB8 has been found! To replicate swarm behavior upon successful detection, the drone that found BB8 should tell all the other drones to fly to its location. This would best be accomplished through a ROS action. Once all drones reach BB8 and get a confident reading on it, that will conclude the simulation.

Figure: How a drone can communicate with its detection and decision node.



This diagram recaps all the design ideas discussed in this section. The decision node will be the main controller for the drone which simply does what the decision node tells it to. The drone itself will have no logic. This design should be easily scalable as every drone should have full autonomy from a centralized controller.

There may be some debate on whether a single decision node should have the power to call all other drones to the target location as that node could be viewed as a centralized controller. This element will have to be reviewed with engineers for further insight.

If a decision node is not permitted to actively call other drones to a location, it could be remedied by having the node publish the location to a topic that all drones subscribe to. Then, individual drones can make their own decision to travel to the location.

If intercommunication needs to be taken to an extreme level, drone proximity could theoretically be implemented where a drone cannot communicate with another drone unless it is within range. This would significantly complicate the project, but it is an idea to think about. To some degree, the previous method of publishing a location to a shared topic still feels like it is a centralized controller.

Eliminating the detection timer?

As proposed, if a drone suspects the target is BB8 with a minimum confidence level, the drone should break routine and approach the target for further detection. Once the drone reaches its optimal detection distance and height, a brief timer should begin. This timer should not be any longer than a couple seconds because at its optimal position, the detector should be outputting its best results. The timer proposal is primarily for the drone to get settled at a good angle.

There is now an idea to eliminate the timer by making the drone fly down to ground level first, then move towards the target. The original idea is to fly down diagonally towards the target as it would be more efficient in distance. With different heights and perspectives for the target to be in, this might result in the detector outputting low confidence. We are also unsure about the ability to tilt the camera during that diagonal descent.

However, if the drone is already descended to ground level and begins approaching the target, it may allow for much higher confidence in detection. By the time the drone reaches its optimal distance, the detector will have had plenty of time to get quality data from a better perspective. If it works out this way, the target will be identified as either BB8 or not BB8 well before the drone reaches its optimal distance.

Simplifying the problem

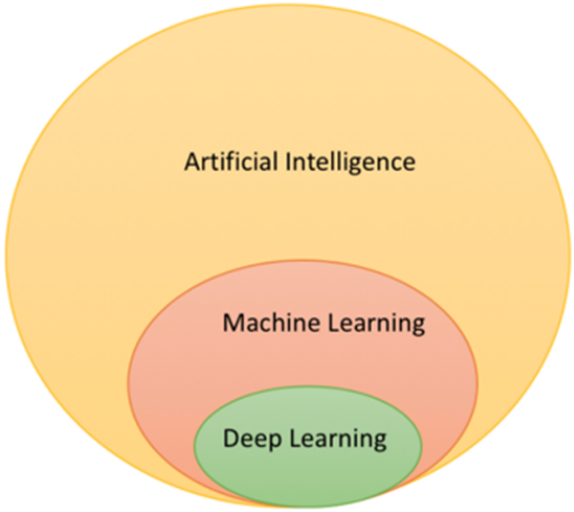
There are many benefits to letting the drone move one-dimensionally towards the target. It drastically simplifies movement which can tie in with path planning. From

current testing and understanding, many movement algorithms in ROS only work in a 2D plane. The concern of needing a tilting camera also gets eliminated from the problem. This is obviously a major benefit for development as it is unclear whether the camera can even tilt in the first place. As for downsides of this simplified movement? Truthfully, not much. Assuming the drones are flying at a low elevation, only a couple seconds at max would be lost making a vertical descent.

4.3 Neural Networks

The following section will discuss different architectures for neural networks and the considerations taken into deciding what's suitable.

Figure: Venn diagram showing that artificial intelligence is a broad category that narrows into machine learning and deep learning. [7]



4.3.1 Deep Neural Networks (DNNs)

What is deep learning?

Deep learning is a subset of the much larger class of artificial intelligence (AI). Broadly speaking, we need to use AI for our drones to be able to recognize and detect our target in the simulated environment. Deep neural networks are the driving backbone for deep learning as it allows an AI to learn and think "smarter".

Why DNNs in general?

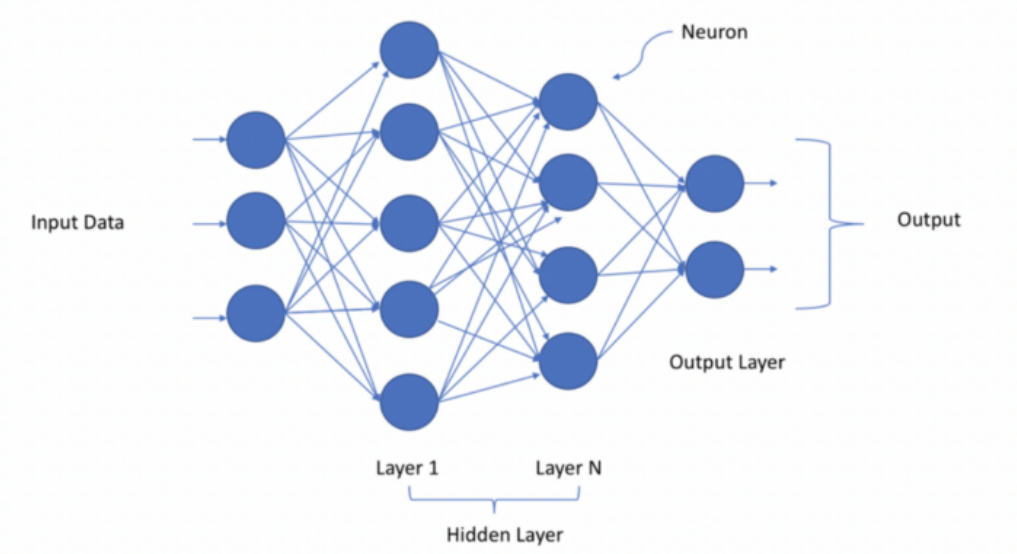
DNNs are desirable for our project case because they perform well with images unlike more generalized methods like AI and machine learning. AI and ML are simply too broad and not specialized enough to accomplish something as complex as object

detection. While they are both capable of independent learning, they are not powerful enough to recognize features in an image and apply that knowledge to other images.

DNN structure

The input layer of any DNN is the “interface” where data enters the network. The input layer is noted to not be hidden. There can be millions of neurons and connections sitting in the hidden layer of a DNN. These layers are hidden for the sake of abstraction and to simplify the understanding of a DNN, but for our purposes, we will need to go into a bit of depth with the hidden layers. Finally, the output layer is where we get results such as the probability that the input image is a “cat.”

Figure: Diagram showing the basic structure of a DNN. Many “neurons” are interconnected to process an input image to an output. [7]



Neurons and connections

Each neuron contains an activation function, and each connection contains a weight. The general idea is that a neuron acts like a “gatekeeping function” for the input connections. The simplest activation function would be a threshold function.

If sum(inputs) > minimum threshold, return 1. Otherwise return 0.

The simple threshold function is considered a “binary” function as it only outputs either 0 or 1. The general idea is that an activation function takes the sum of its inputs and applies a function to it. A general mathematical representation would be the following, where x is a connection:

$$f(\sum x_i)$$

A connection holds a weight which simply helps “pick” what connections are valuable. Let’s make a simple, high-level example with recognizing a stop sign. When it comes to recognizing a stop sign, shape and color are arguably the most important factors. The red color might catch your eye, and the octagonal shape may affirm that the sign is, indeed, a stop sign. In this case, the connections associated with shape and color will have a high weight, signifying its importance in recognition. Text might not be as important compared to shape and color, so it would have a relatively lower weight. Take this simple example and shrink it down to the pixel level.

The activation function can be updated to include the weight now, where w_i is the weight associated with a connection:

$$f(\sum x_i w_i)$$

As data (a value) enters the input layer of a DNN, it gets sent through its respective connections, getting multiplied by the connection’s weight, and arriving as an input to a neuron. That neuron will sum the input along with other inputs before applying its activation function. The result is then outputted and sent along another weighted connection to another neuron.

The output

Under the context of object detection, the final output layer will generally have the same number of neurons as the number of detection classes in the algorithm. For example, if we are detecting cats, dogs, humans, and cars, our output layer should have 4 neurons. Why?

The final step of a neural network for object detection is that they will typically output a percentage that is a certain class. Using the four classes in the previous example, imagine a fuzzy animal is the input image. You would expect the neuron associated with cat or dog to output a high percentage while the humans and cars neuron would output a low percentage.

Training weights in a neural network

As a neural network gets trained, the connection weights are the values being updated. As a detector learns certain features are desirable, the weights associated with the features will increase. In the opposite manner, if certain features turn out to be irrelevant to detection, those weights will decrease. Going back to the stop sign

example, if stop signs started to come in different colors besides red, you would probably care less about the color and factor it less in your decision making.

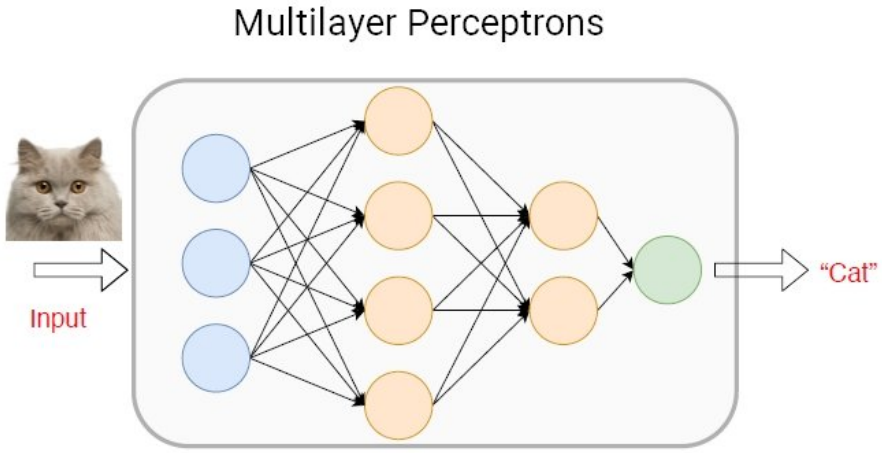
Subclasses of Deep Neural Networks

DNNs are still considered too broad for our purposes. The previous section covered the general trends and behaviors of DNNs, but we need to dive deeper into more specialized solutions for object detection. We're getting close to our architecture, though.

Multilayer Perceptrons (MLPs)

The multilayer perceptron looks shockingly similar to the previous image of a DNN, and that's because it is. MLPs are almost a splitting image of DNNs. It's like the analogy that a square is a rectangle, but a rectangle isn't a square. An MLP is a type of DNN, but a DNN isn't an MLP. The distinguishing factor that confines an MLP in its classification is that every layer is fully connected. An ordinary DNN has no restrictions on how layers should interconnect. The behaviors of a DNN carry over to MLPs easily.

Figure: Diagram showing a neural network called the multilayer perceptron. [8]



MLPs pros and cons

Because MLPs are essentially the generic definition of a DNN, they can accomplish pretty much any general-purpose deep learning task such as classification and even object detection if needed. Just because they can accomplish object detection, doesn't mean they should, though. A big drawback of a fully connected network is that computations are expensive. A single MLP can easily have millions of connections. They also do not handle image sizes well. If a model was trained with

1000 x 1000 images, it would not be able to deal with 2000 x 2000 images. The neurons and connections are inflexible to scaling.

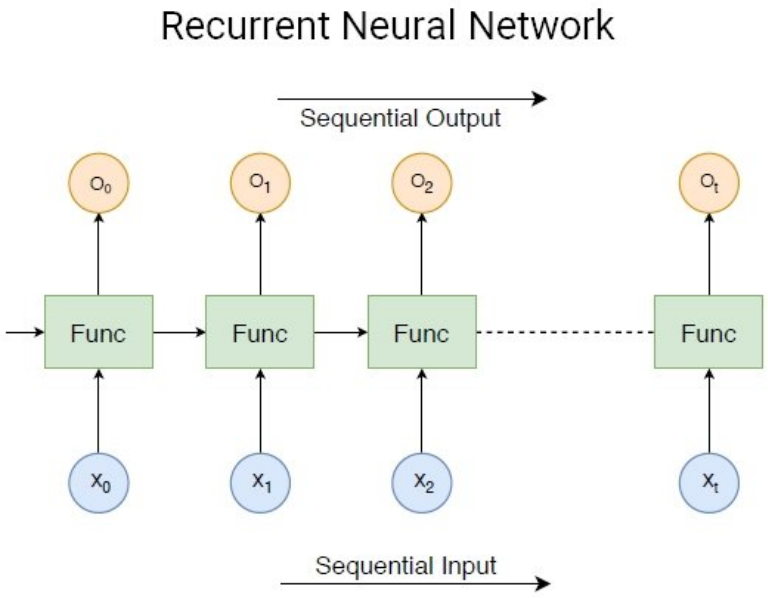
What if we trained the model using 2000 x 2000 images instead? It'd be capable of handling those images, but the computation required would explode out of control now. 2 layers with 2 neurons each would give you 4 connections. 2 layers with 3 neurons each would increase to 9 connections. Simply increasing the number of neurons to handle image sizes would be terribly inefficient and expensive.

Taking it to the logical extreme, what happens if we try object detection on a live camera feed in 4K (3840 x 2160) @ 60 frames per second? There's got to be a better neural network for object detection.

4.3.2 Recurrent Neural Networks (RNNs)

This section won't go too in-depth with RNNs because, spoiler alert, they are not what we are going to use. In fact, they are probably the last candidate of the neural networks, but it's worth mentioning to be inclusive.

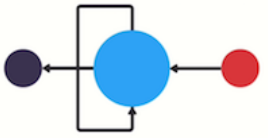
Figure: Diagram showing the structure of a recurrent neural network. It is quite the departure from the look of a densely connected network. [8]



RNNs are excellent in handling sequential data where the length is variable. This should be a good enough indicator that may not be the best candidate for object

detection, but let's cover a bit more ground. RNNs have this idea of "memory" and can loop its output back to its own input.

Figure: Image showing how a RNN can receive its own output. [9]



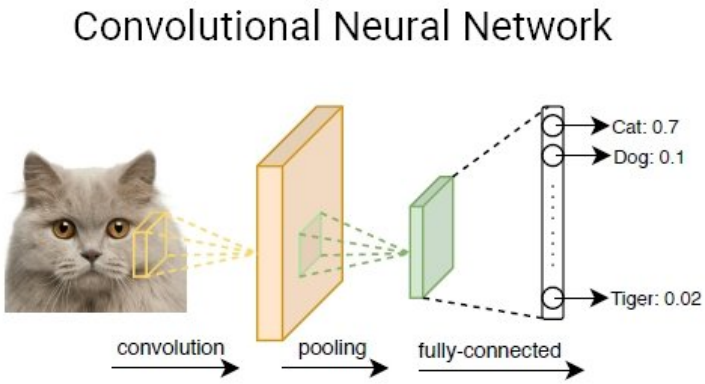
The image is quite simple, but hopefully it offers some high-level insight on the structure of RNNs.

Since RNNs excel at sequential data, this makes RNNs good at language processing, speech recognition, and audio processing. It may kind of sound like RNNs would be good for our use case because it will need to process a continuous, live camera feed, and it does make some sense if presented in that context. However, object detection in this case does not need to rely on past inputs to decide the current input. It would be best to think of each frame of the camera feed as an independent case that needs detection.

4.3.3 Convolutional Neural Networks (CNNs)

CNNs work a bit differently from MLPs, but not drastically like RNNs. The main idea is that certain features are extracted from an image using convolution layers. Then the layers are pooled to abstract features and reduce the image size. Finally, fully connected layers are back and behave similarly to MLPs.

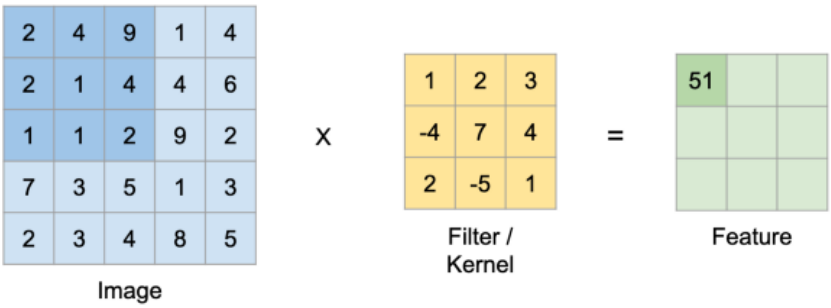
Figure: The architecture of a CNN. It is somewhat like a connected DNN or MLP, however, there are some processing steps in the beginning that makes it stand out. [8]



What is convolution?

Take an image filter (also called a kernel) and place it directly on top of your image and simply multiply element wise. Finally, sum the values from multiplication and you've obtained an element of your features matrix. Convolution is powerful because it allows you to apply all sorts of image processing kernels on your input image to extract key features.

Figure: A convolution step from the entire operation. [10]



An example of a powerful kernel would be the Sobel kernel. It applies a slight blur and
finds the derivative in either the X or Y direction depending on the kernel's orientation. It is the first step required in a method of edge detection.

Something that you might have noticed in the convolution image is that the output matrix is smaller than the input matrix. The output matrix will have the following dimension, where W is the image size and K is the size of the kernel:

$$Output = (W - K) + 1$$

You might not want a shrinking output kernel, so the remedy is to pad the original image. Padding is simply empty values placed around the original image to minimize the shrinking output dimension. The new output dimension will be the following, where P is the size of padding:

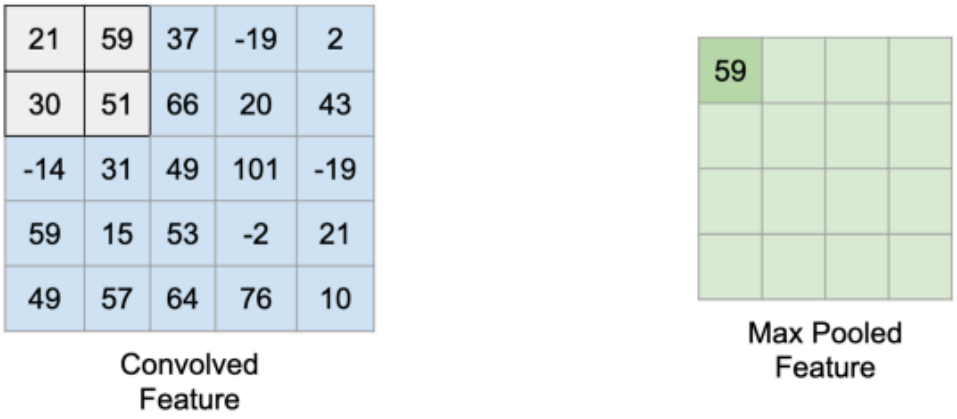
$$Output = (W - K + 2P) + 1$$

Pooling

Pooling is commonly the next step in a CNN, and it provides a few benefits to the overall architecture of a CNN. It creates a "summary" of the features obtained from

convolution and down samples the convolution results. The nature of this operation makes the trained model more robust to locational changes of features.

Figure: The first step of the pooling operation. In this case, it is a “max pool” operation. [10]



Pooling operations

There is min, max, and average pooling, but max pooling is the most common choice when pooling. The max pool operation is typically used to extract the extreme features from convolution. Let’s use edge detection as an example. After edge detection, the output image will typically be black, and all edges will be shown as white. Black pixels will have the value of 0, and white pixels will have the value of 255. By max pooling our edge detection output, we will shrink the image while focusing on extracting the white edges.

Pooling an image gives the following output dimension, where W is the image size and K is the pooling size:

$$Output = (W - K) + 1$$

This formula is the exact same as the basic convolution’s output dimension.

What about padding?

With pooling, we don’t want to pad the convolved features. Downsampling the features is a benefit of padding as it makes future computations cheaper. The next question might be, “If we are trying to simplify the image, why are we padding in the convolution step?” Padding in the convolution step isn’t mandatory, but it provides us with the opportunity to extract more features near the edges. Once we have obtained as

many features as possible, then we can do pooling to summarize the features. Of course, this is just a high-level explanation on padding. It's worth noting that many algorithms do opt out on padding during convolution.

Behavior of pooling

Briefly mentioned in the beginning of the pooling section, because this operation summarizes features and down samples, locational changes of features won't impact the model as much as without pooling. For example, when performing convolution operations, features can be mapped to a 1000 x 1000 output layer. If we were detecting edges of a stop sign, many pixels in the output layer would be dedicated to the edges of the sign giving us a large octagon. This octagon's position has a large impact in such a large image.

By down sampling with pooling, we can shrink the octagon into a smaller image size. Humans can still recognize the octagonal shape despite its size, and the learning algorithm will focus less on the exact position of the octagon as it has less pixels to work with. Instead, the algorithm will learn to generalize that there are edges around the general location.

Fully connected layers

The fully connected layers are the final layers of a CNN. These behave just like the layers in an MLP, except that they will be much faster. Thanks to pooling layers in the previous step, the number of neurons required in the fully connected layers will be significantly less than if it were not pooled. As a result, we can get the benefits of using fully connected layers at a much lower computation cost. Finally, just like an MLP, the output layer will be probabilities that the input image is of a certain detection class.

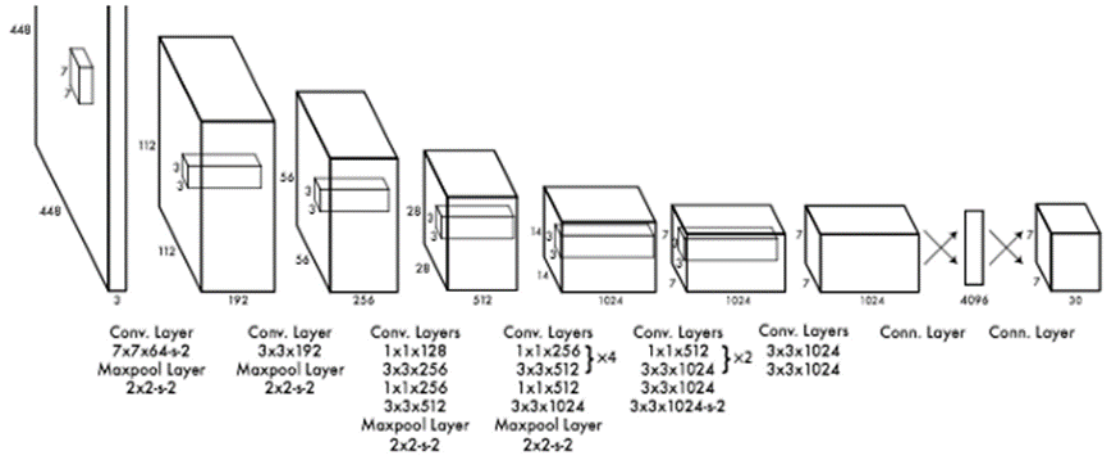
Let's quickly recap the behavior of CNNs and relate it back to why they are a good candidate for object detection. First, CNNs apply an assortment of image processing kernels onto the original image. These kernels will extract desirable features from the image. Then, the extracted features will be pooled to shrink the image and generalize the location of features. Finally, the pooled features will run through fully connected layers that perform the decision making and outputs probabilities. This process allows for the behaviors of MLPs while reducing neurons and generalizing features all at the same time.

This overview on CNNs is very generalized. There are many different CNN architectures that exist with different variations on convolution and pooling. Some modern CNNs take a departure from this approach and do entirely different things.

How CNNs relate to YOLO

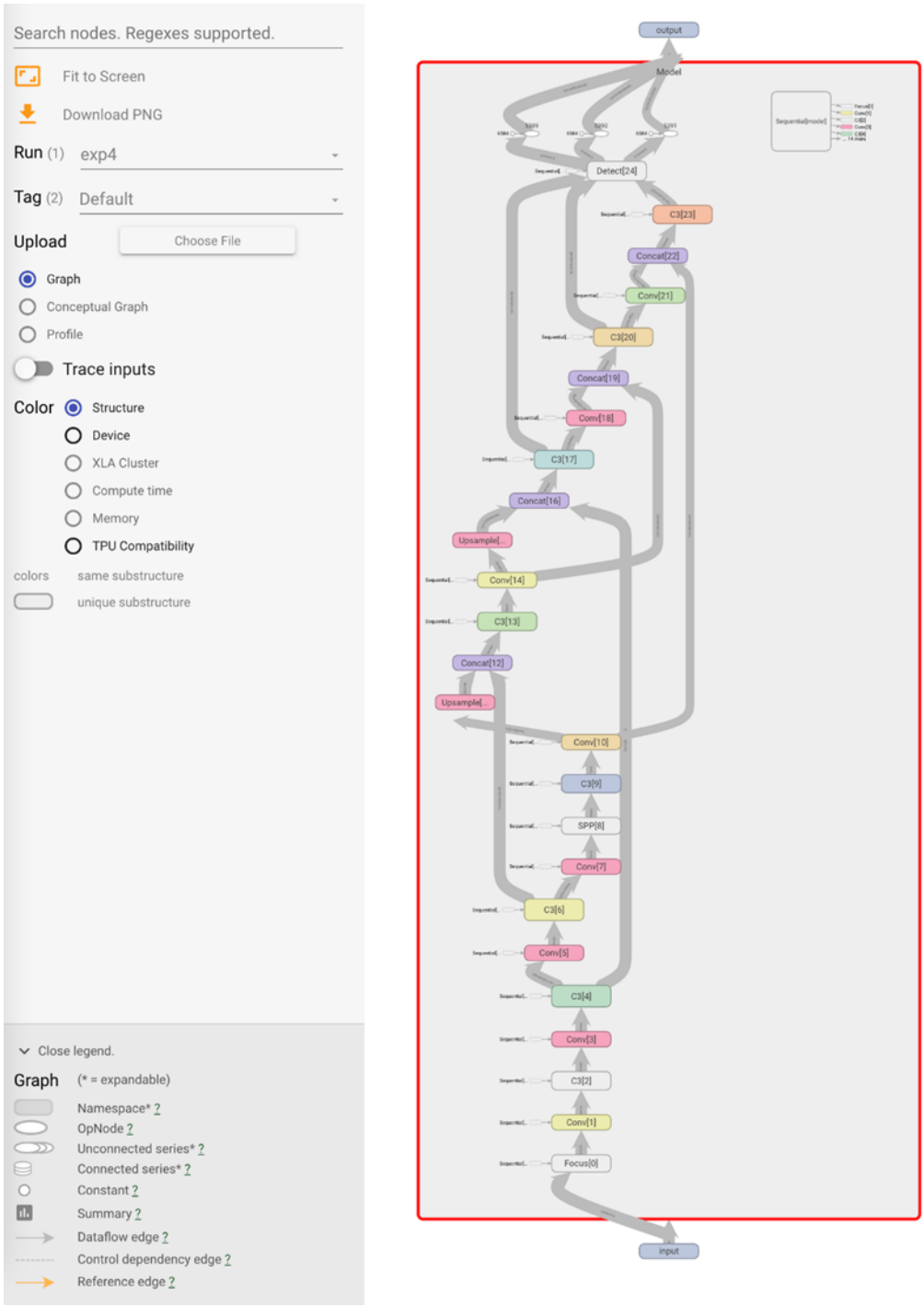
For historical context, YOLOv1 follows the architecture of a CNN quite closely. As the basic CNN description mentions, there is a convolution layer that gets pooled. Then the pooled layer is fully connected to the output. In this case, convolution and pooling happens four times. Then, it is followed up with many more convolution layers. As you see, the concepts of a CNN apply quite well in this scenario.

Figure: The internal architecture of YOLOv1. [11]



This next section will briefly cover YOLOv5's structure. It is significantly different than YOLOv1's architecture, but at the same time, the basic ideas and processes of CNNs are there. YOLOv5 comes with some new tricks up its sleeves to make training more powerful. We will not dive too deep into structure as it is extremely complex and the research paper is not out yet, but it is safe to say the fundamentals are still there.

Figure: The internal structure of YOLOv5 in Tensorboard. It contains convolution, C3 linearization, spatial pyramid pooling, up sampling, concatenation, and detection layers. [12]



The image may be blurry due to downsizing. It is fine if the labels are illegible as the exact details are not a concern. Refer to the image caption to get a list of processing layers used.

You'll notice YOLOv5's architecture is extremely complex compared to YOLOv1, but some of the founding ideas still hold. Convolution and pooling are still there,

however, there is only a single pooling step! Pooling has also been upgraded to a new “spatial pyramid pooling” (SPP) technique.

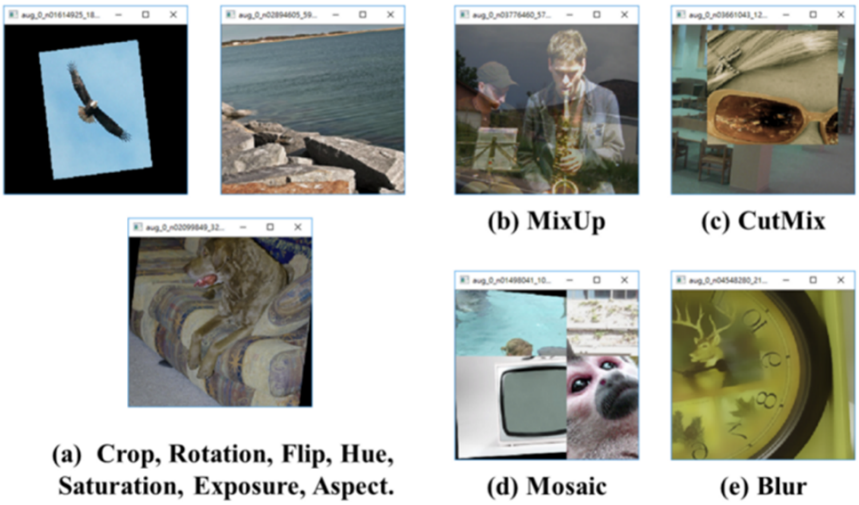
> An interesting thing about SPP is that it allows for variable-size input images. This was something not mentioned in the basic CNN section as CNNs are not technically limited to fixed-size inputs. However, more primitive CNNs, such as MLPs will run into that limitation. Having a fixed-size input makes them harder to use in general applications [13]

This limitation occurs because there needs to be a mapping of variable-size inputs to a finite output. In the case of a fully connected MLP, there are a fixed number of neurons and weights associated with a model. What happens if we put a 2000 x 2000 image into a 1000 x 1000 trained MLP? There are not enough neurons and weights to handle that input size! This pooling technique is capable of mapping to a fixed output size.

Data Augmentation

Data augmentation is arguably the biggest difference between v1 and v5. For example, there is an augmentation technique called “mosaic” that was introduced in v4. It takes four different training images and combines them to create a challenging image for the detector to learn from. Different kinds of image processing tricks make the training images harder to work with, forcing the detector to heavily generalize and focus less on finer details. [14]

Figure: Image showing different sorts of augmentation techniques. [14]



Data augmentation is desirable because it essentially creates new data out of thin air. When you have a limited dataset, augmentation becomes even more important just for the sheer volume. As for general training, like mentioned in the previous

paragraph, augmentation is excellent at making the network generalize more and think smarter.

YOLOv5 comes packaged full of different augmentation methods that it automatically applies for you during training. For the most part, the augmentation parameters do not need to be modified, but they are still accessible as hyperparameters. For the purposes of this project, we do not intend to modify hyperparameters unless there is a large excess of time that allows for it.

4.4 Swarm Behavior Algorithms

4.4.1 Swarm Intelligence

Swarm Intelligence is the collective behavior of independent systems. SI systems are primarily built from a collection of agents that are fed information about each other in order to make real-time decisions in their environment [56]. Each agent is governed by a set of simple rules and in some ways are considered “dumb” agents individually. However, the interactions between the agents built by these simple rules can lead to a higher intelligence over the entire swarm, almost like a hive mind [56]. This intelligence is what allows the swarm to be directed towards whatever the goal may be.

4.4.2 Swarming Behavior Models

Boids

A Boid is an artificial lifeform that is designed to emulate the behavior of a flock of birds. The agents within a Boid rely on three simple rules in order to derive the global intelligence of the swarm [56]:

1. *Separation*: Maneuver to maintain a certain distance from fellow flock members.
2. *Alignment*: Maintain a local heading that is continuous with the global swarm heading.
3. *Cohesion*: Maintain a certain distance from the flock’s center of mass in order for the swarm’s formation integrity to hold.
4. Optional Rules:
    a. *Obstacle Avoidance*: Each agent must relay information on detected obstacles both to itself and to fellow flock members about nearby objects in the flight path. The swarm agents will then adjust themselves as necessary while maintaining rules 1-3.
    b. *Goal Seeking*: If the swarm has an ultimate goal in mind such as finding a certain object or place, then all agents will continue searching for it individually and relaying that data to fellow members. When the target is

found, the local intelligence will relay the information across the swarm members, ultimately changing the global level intelligence from searching to navigating.

We can utilize this model to allow for effective flocking of the drones as they traverse the environment.

4.4.3 Metaheuristics

Stochastic Diffusion Search

This is a probabilistic approach to implementing swarm behaviors. It is best suited for global searching problems where the goal can be subdivided into smaller objectives that are independent of each other [62]. Each agent makes a hypothesis based on its input data. The hypothesis is then evaluated against each sub-problem which, depending on the binary outcome of the evaluation, may activate or deactivate the agent. Agents share hypotheses with each other across the swarm's communication network in order for each agent to update their hypothesis based on the one that resulted in a correct solution to solving the sub-problem [62]. Over time, the global hypothesis will result in a single best solution that allows the swarm to solve the overarching problem.

Ant Colony Optimization

ACO is another probabilistic approach for swarming intelligence algorithms. It is best suited for path planning problems when the system is provided with a graph [63]. Agents act greedily by searching for the optimal solution in the current set of possible choices. When an agent has selected the optimal solution at the current time step, it then records its positional data along with other information [57]. This information is then used either in future iterations of the swarm's search or transmitted to other agents in the swarm so it may lead them down a more globally optimal solution.

ACO is especially popular among several areas in swarming robotics and can be used as a form of path planning for the overall swarm. Ultimately, the problem is broken down into finding the shortest path in a weighted graph. Below is an example pseudocode of the ACO algorithm [63].

```
procedure ACO_MetaHeuristic is
    while not terminated do
    generateSolutions()
    daemonActions()
    pheromoneUpdate()
```

```
        repeat
end procedure
```

Edge Selection

Each agent must construct a solution in the graph to maximize the effectiveness of the global solution. Similar to Dijkstra's algorithm, the agent will consider each adjacent edge and the corresponding pheromone level. The global solution is formed as each kth agent moves from state x to y at each time step [63]. The kth agent then computes a set of possible expansions to its current state and randomly selects the edge based on updated probabilities [63]. The probability of moving from state x to y is determined by the following expression [63]:

$$p_{x,\,y}^{k} = \frac{\left(\tau_{x,\,y}^{\alpha}\right)\left(\eta_{x,\,y}^{\beta}\right)}{\sum_{z \in allowed_z}\left(\tau_{x,\,z}^{\alpha}\right) \cdot \left(\eta_{x,\,z}^{\beta}\right)}$$

Where:

- p is the probability of traversing edge x → y for agent k,
- $\tau_{(x,\,y)}$ (Tau) is the pheromone value released from traversing edge x → y.
- α (Alpha) controls the influence of $\tau_{(x,\,y)}$ for edge (x, y).
- $\eta_{(x,\,y)}$ (Eta) is the desirability of performing a transition along edge (x, y).
- β is a parameter that controls the influence of $\eta_{(x,\,y)}$.
- $\tau_{(x,\,z)}$ is the trail level for other possible state transitions from the current state, x.
- $\eta_{(x,\,z)}$ is the attractiveness for other possible state transitions from the current state, x.

Pheromone Updating

When all agents have completed their solutions, we modify each agent's trail levels depending on if they were good or bad moves to make at that time step. Increasing the trail level indicates a good solution and may carry forward to the next iteration, while decreasing means the opposite [63]. While there are several ways to update the pheromone values, one generic method is the following expression [63]:

$$\tau_{x,\,y} = (1-\rho)\tau_{x,\,y} + \sum_{k}^{m} \Delta\tau_{x,\,y}^{k}$$

Where:

- $\tau_{(x, y)}$ (Tau) is the pheromone value released from traversing edge x → y.
- ϱ is the pheromone evaporation coefficient
- m is the number of agents within the swarm
- $\Delta\tau^{k}_{x, y}$ is the amount of pheromone deposited for the kth agent.

We anticipate that this algorithm may prove the most effective solution for the project as it is most suited for determining optimal paths through an environment. This may reduce the time needed to find the target if the swarm's intelligence is constantly being updated as new data is fed between agents.

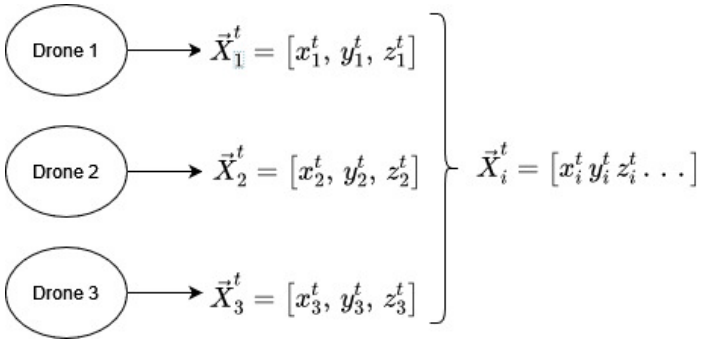
Particle Swarm Optimization

PSO is a global optimization algorithm that is best suited for problems whose solution is represented as a point or surface in an n-dimensional space [58, 60]. The hypotheses of the swarm's agents are plotted in space and initialized with an initial velocity and communication networks between the other points. Particles are then moved through the solution space and their fitness evaluated after each time step. Particles eventually move towards groups that have better fitness values.

In PSO, each agent has three components to track [58, 60]:

1. Current timestep velocity
2. Distance to Personal best location
3. Distance to Swarm's best location

We can initialize each agent's position vector as the following [58, 60]:



Where X represents each drone's position vector at time step t in 3-dimensional space. All position vectors are stored within a single matrix X_i. To update this position

vector, we must define the movement vector and speed using a velocity vector. This vector is continuously updated as well using the following function [58]:

$$\vec{V}_i^{t+1} = w\vec{V}_i^t + c_1 r_1\left(\vec{P}_i^t - \vec{X}_i^t\right) + c_2 r_2\left(\vec{G}^t - \vec{X}_i^t\right)$$

Where:

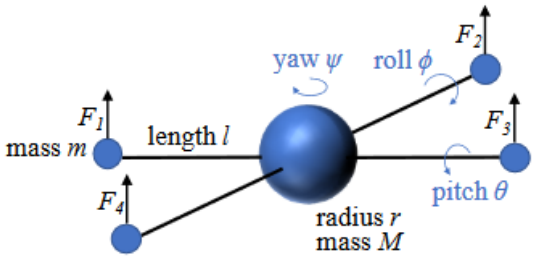
- V^{t+1} is the velocity of drone i at the next time step.
- V^t is the velocity at the current time step for drone i.
- P^t is drone i's best solution at time step t.
- P^t - X^t is the distance from drone i's current position to its personal best solution
- G^t is the overall swarm's best solution
- G^t - X^t is the distance from drone i's current solution to the global solution.
- *w* is the inertia of the particle
- *r* is a random value between 0 and 1 that either increases or decreases the influence of each component in the final magnitude of the velocity vector.

With this information, we can then calculate drone i's next position at time step t+1 using the following formula:

$$\vec{X}_i^{t+1} = \vec{X}_i^d + \vec{V}_i^{t+1}$$

This process is repeated for all solutions in the PSO landscape so the swarm can consider the entire set of possible solutions and choose the optimal one.

One factor we must also consider is the overall quaternion of each drone in the swarm. When updating the position vector using the above mathematical model, we also need to consider how the drone approaches new positions [60]. This will be done by developing a rotation matrix which maps a vector from the body frame to the inertial frame.



Kinematic frame of a standard quadrotor drone. [60]

Using the above model, the derived rotational matrix [60] would be:

$$R = \begin{bmatrix} \cos(\phi)\cos(\psi) - \cos(\theta)\sin(\phi)\sin(\psi) & -\cos(\psi)\sin(\phi) - \cos(\phi)\cos(\theta)\sin(\psi) & \sin(\theta)\sin(\psi) \\ \cos(\theta)\cos(\psi)\sin(\phi) + \cos(\phi)\sin(\psi) & \cos(\phi)\cos(\theta)\cos(\psi) - \sin(\phi)\sin(\psi) & -\cos(\psi)\sin(\theta) \\ \sin(\phi)\sin(\theta) & \cos(\phi)\sin(\theta) & \cos(\theta) \end{bmatrix}$$

The linear motion of the drone would then be derived from the Newtonian expression:

$$m\ddot{x} = \begin{bmatrix} 0 \\ 0 \\ -mg \end{bmatrix} + RT_B + F_D$$

Where:

- x is the position of the drone.
- g is acceleration due to gravity
- F_D is the drag force
- T_B is the thrust vector in the body frame.

We use the above calculations along with the angular motion to derive the state space vectors for the drone.

The state space vectors are the key components which give us the dynamic model for each drone in the swarm. This information is derived using the following set of equations [60].

$$\begin{aligned} x_1 &= x_2 \\ x_2 &= \begin{bmatrix} 0 \\ 0 \\ -g \end{bmatrix} + \frac{R \cdot T_B}{m} + \frac{F_D}{m} \\ x_3 &= \begin{bmatrix} 1 & 0 & -s_\theta \\ 0 & c_\phi & c_\theta s_\phi \\ 0 & -s_\phi & c_\theta s_\phi \end{bmatrix}^{-1} \times x_4 \\ x_4 &= I^{-1} \cdot [\tau - x_3 \cdot (I \cdot x_3)] \end{aligned}$$

Where x_1 is the velocity of the drone, x_2 is the acceleration, x_3 is the angular velocity, and x_4 is the angular acceleration. With these components, we can relay it to the other drones within the swarm along with the current position to help the emergent intelligence make better decisions in the PSO algorithm.
[60] provides an overview of the pseudocode used to implement the PSO method with a small swarm of quadrotor drones in a real-life scenario.

(1) Initialize $\mathbf{S}$.
for $k = 1, 2, \ldots, N_D$ **do**
  $\mathbf{x}_0^k$ = random vector $\in \mathbb{S}$
  $\mathbf{v}_0^k = \mathbf{0}$
  Evaluate $f(\mathbf{x}_0^k)$
  ${}^p\mathbf{x}_0^k = \mathbf{x}_0^k$
  Deploy d^k at $\mathbf{x}_0^k$
  if $f({}^p\mathbf{x}_0^k)$ is better than $f({}^g\mathbf{x}_0)$ **then**
    ${}^g\mathbf{x}_0 = {}^p\mathbf{x}_0^k$
  end if
end for
$t = 0$

(2) Update $\mathbf{S}$.
for $k = 1, 2, \ldots, N_D$ **do**
  $\mathbf{v}_{t+1}^k = w \cdot \mathbf{v}_t^k + c \cdot [\phi_1({}^p\mathbf{x}_t^k - \mathbf{x}_t^k) + \phi_2({}^g\mathbf{x}_t - \mathbf{x}_t^k)]$
  $\mathbf{x}_{t+1}^{*k} = \mathbf{x}_t^k + \mathbf{v}_{t+1}^k$
  $\mathbf{x}_{t,0}^k = \mathbf{x}_t^k$
  $n = 0$
  while $\mathbf{x}_{t,n}^k \neq \mathbf{x}_{t+1}^{*k}$ **or** a collision is not predicted **do**
    $\mathbf{c}$ = Controller$(\mathbf{x}_{t,n}^k, \mathbf{x}_{t+1}^{*k})$
    Actuate d^k with $\mathbf{c}$
    $(n = n + 1)$
    Evaluate $\mathbf{f}(\mathbf{x}_{t,n}^k)$
    if $f(\mathbf{x}_{t,n}^k)$ is better than $f({}^p\mathbf{x}_t^k)$ **then**
      ${}^p\mathbf{x}_t^k = \mathbf{x}_{t,n}^k$
    end if
  end while
  if $f({}^p\mathbf{x}_t^k)$ is better than $f({}^g\mathbf{x}_t)$ **then**
    ${}^g\mathbf{x}_t = {}^p\mathbf{x}_t^k$
  end if
end for
$(t = t + 1)$

(3) Repeat (2) until a termination condition is met.

$\mathbb{S}$: The search space of the problem
$\mathbf{S}$: The swarm of the drones
N_D: The number of drones in $\mathbf{S}$
d^k: The k-th drone
$\mathbf{x}_t^k$: The position of d^k at iteration t
$\mathbf{v}_t^k$: The displacement of d^k at iteration t
$f(\mathbf{x}_t^k)$: The objective function value of $\mathbf{x}_t^k$
${}^g\mathbf{x}_t$: The global best position of $\mathbf{S}$ at iteration t
${}^p\mathbf{x}_t^k$: The personal best position of d^k at iteration t
$\mathbf{x}_{t,n}^k$: The position of d^k in the n-th control period at iteration t
$\mathbf{c}$: The command output of the controller

Psuedocode of the swarm search algorithm proposed by researchers at Kwangwoon University [60]

Artificial Swarm Intelligence

ASI uses a set of control algorithms that are modeled after natural swarms. These control algorithms are used to influence collective intelligences of human groups. Each human agent is treated as an active member of the control system. Typically, this method is used with human teams to either make decisions or forecast data [68]. The idea is analogous to the honey bee swarm methodology in its decision making processes.

The swarming process derives from when a hive exceeds its current home size and must relocate to a new home. Hundreds of scout bees are deployed within a certain radius from the current hive to search for a new home. Each bee agent in the scout swarm must consider several factors and relay this data back to the overall bee hive [65]:

1. The hive must be large enough to store the needed honey.
2. The hive must be insulated to maintain optimal temperatures during different seasons.
3. The hive must be close to a reliable water source.
4. The hive must be close to reliable pollen sources.

Once this information is relayed to the overall hive, the emergent intelligence makes a decision on where to relocate the hive. This multivariable problem can be turned into a mathematical model which would allow the same principle to be applied to a drone swarm. The model variables would differ from the honey bee example, however the emergent intelligence would be able to make better decisions using input data from the overall swarm.

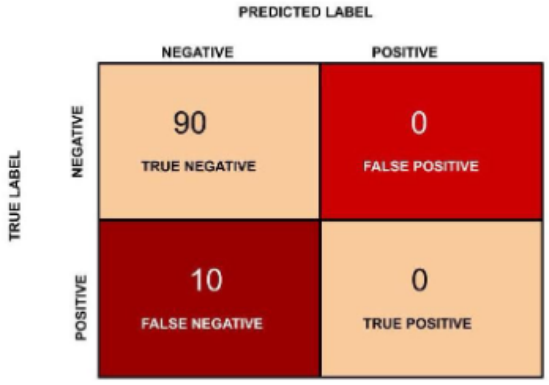
The downfall with this approach is that ASI is designed to be primarily used with networked human groups [68]. Thus, implementing this approach using drones may cause a reduction in search efficiency.

4.5 Object Detection Metrics

4.5.1 Confusion Matrix

The confusion matrix is a machine learning tool commonly used when we want to gather information on the performance of our model. This information is then used to calculate various metrics like model Accuracy, Precision, Recall, and other important measurements [59]. Below is an example of what a confusion matrix looks like.

We treat True Positives (TP) as the object detection model correctly identifying an instance of the BB8 target. True Negatives (TN) are all data instances that we correctly identify as not the BB8 target, such as buildings, trees, cars, and other unrelated objects.



Example of a Confusion Matrix. [63]

False Positives (FP) are all data instances where the model incorrectly classified something as the BB8 target. False Negatives (FN) are all instances where the model incorrectly classified an instance of our target as not the target. While not the best measurement by itself, it is an important baseline metric we are interested in seeing. This is why we use the following metrics defined below.

Please note that the above example is not a depiction of our model's performance as we do not have any data collected yet. This is purely an illustration for the reader to better understand how we plan to test and measure the object detection model.

4.5.2 Accuracy

Accuracy is used in machine learning when we want to measure how many predicted instances were close or matching the accepted value which is any instance. It is also worth noting that Accuracy is a useful measurement when datasets are symmetric. While we do not have any data as of yet, we plan on creating an even split in the number of Actual Negative and Actual Positive data so our data distribution is symmetric. Below is the defined formula we use to calculate estimated accuracies for the object detection model [59].

$$A = \frac{TN + TP}{TN + FP + TP + FN}$$

Where *A* is the accuracy, *TN* is the true negative, *TP* is the true positive, *FP* is the false positive, and *FN* is the false negative.

4.5.3 Precision

Data scientists typically use the precision metric when the cost of a False Positive is high. Precision indicates how many predicted positives are True positives. In the case of our drone project, we value precision since it could prove disastrous and fatal in certain real-life scenarios. In military settings, the object detection model being fed the drone camera data will have to have an extremely high precision value. A false positive value would indicate that a U.S. or ally soldier has been identified as an enemy target and will likely be fired upon from friendly forces. Thus, measuring precision is valuable for the sake of friendly ground and air units. Below is an illustration of the precision formula [59]:

$$P = \frac{TP}{TP + FP}$$

Where *P* is the precision value, *TP* is the True Positive value, and *FP* is the False Positive rate. The denominator here is the total predicted positive rate.

4.5.4 Recall

Recall typically defines how many actual positives our object detection model captures by labeling them as Positive. Having a low recall could prove detrimental in the situation of search and rescue missions. If drone swarms using our software were deployed to areas struck by natural disasters to search for missing persons, having a low recall could cause the drone to mis-identify a person as an inanimate object. This could lead to emergency services to postpone searches for certain areas leading to more deaths for those stranded. The formula for Recall is displayed below [59]:

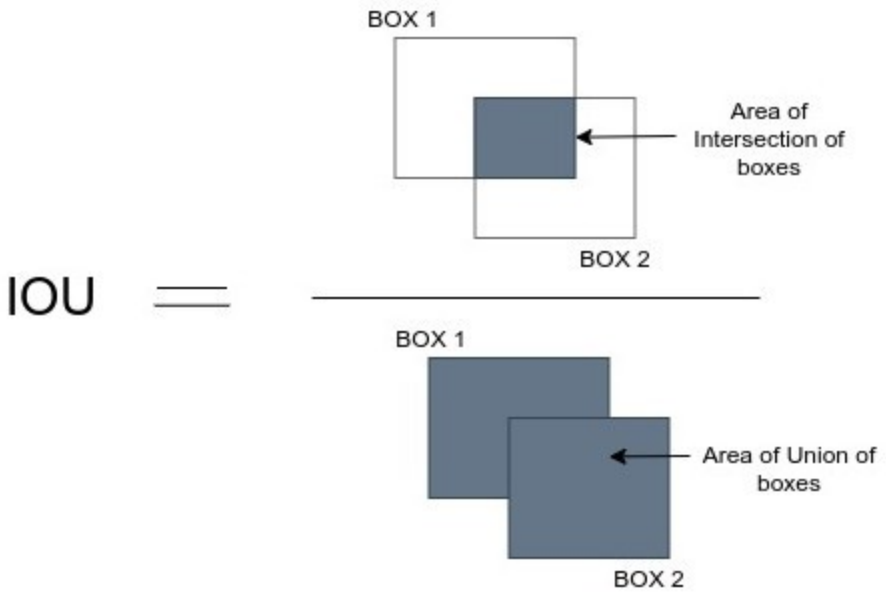
$$R = \frac{TP}{TP + FN}$$

Where *R* is the recall, *TP* is the true positive rate, and *FN* is the false negative rate. The denominator results in the total number of actually positive results.

4.5.5 Intersection over Union (IoU)

IoU describes the overlap between two bounding boxes drawn over an image. One bounding box is drawn by a human to establish a ground truth. The second box is drawn by object detection. IoU values greater or equal to 0.5 is commonly viewed as the minimum for a “good” prediction. Larger values are better in this case.

$$IOU = \frac{Area\ of\ Intersection\ of\ two\ boxes}{Area\ of\ Union\ of\ two\ boxes}$$



4.5.6 Mean Average Precision (mAP)

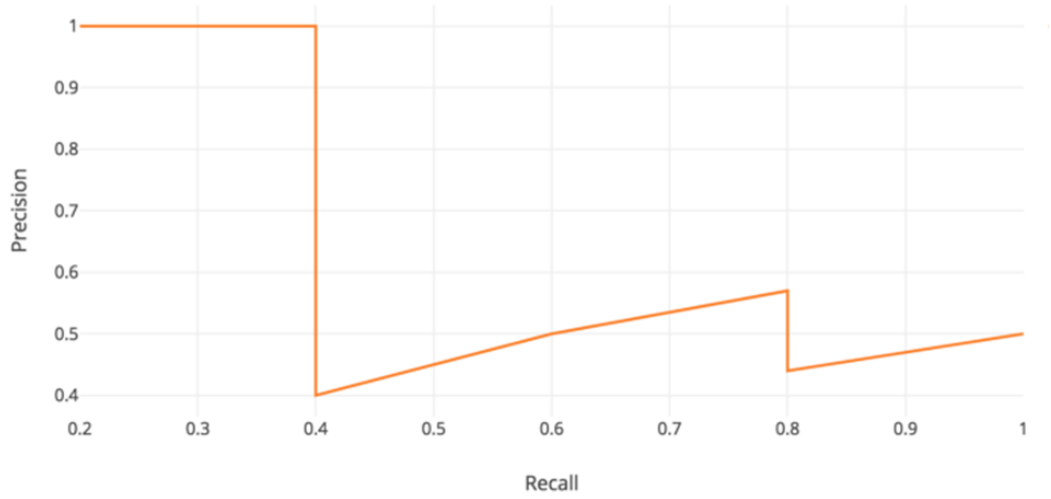
Knowledge of precision, recall, and intersection-over-union will be needed to understand mAP. If we ran an object detection algorithm on some test images, it would return a bounding box for the image, its confidence level, and label(s) for the object(s) detected. What we can do is rank our predictions based on its confidence level:

Figure: Example table of prediction rankings along with its precision and recall. [2]

Rank	Correct?	Precision	Recall
1	True	1.0	0.2
2	True	1.0	0.4
3	False	0.67	0.4
4	False	0.5	0.4
5	False	0.4	0.4
6	True	0.5	0.6
7	True	0.57	0.8
8	False	0.5	0.8
9	False	0.44	0.8
10	True	0.5	1.0

The table trends show that precision and recall are inversely proportional. If we plot the points using precision vs recall, we'd get the following graph:

Figure: Graph that plots precision vs recall of the data from the table above. [2]



Whenever there is a false positive, the trend will drop. Conversely, true positives will bring up the trend.

This finally brings us to the biggest step of understanding mean average precision: average precision. Average precision is simply the area under the curve of a precision vs recall graph. Mean average precision is derived from the fact that there are multiple object classes when dealing with object detection. The average precision of every object class is taken, then averaged.

Take, for example, 4 classes in an object detection algorithm: cats, dogs, humans, and cars. At the end of object detection, the algorithm will produce 4 tables detailing precision and recall. Precision and recall are plotted, then the area under the curve is taken giving us average precision (AP) for 4 classes. Finally, the 4 APs are averaged together to give us our mean average precision (mAP).

Mean average precision is an important metric used in object detection to compare different algorithms against each other. The higher mAP value corresponds to the better algorithm. It should be mentioned that competing algorithms' mAP values should only be compared against each other if the data set is the same.

COCO Dataset

The dataset used is very important. You cannot compare mAP values of different algorithms if they are not trained and tested on the same dataset. This is where COCO comes in.

> Common Objects in Context, most commonly referred to as COCO, is simply a large dataset that has become the standard in object detection. It comes prepared with labeled training data and testing data. [3]

There are many variants of the dataset, but many object detection algorithms will commonly use standard COCO or “COCO test-dev”. The dataset itself does not matter too much, but it is important to reiterate that algorithms must only be compared under the same dataset.

4.7 Swarm Behavior Metrics

4.7.1 Object Collision

If the path planning algorithm is unable to exclude all obstacles within the environment, or an unexpected obstacle has been encountered, the swarm must be able to react appropriately to avoid loss of any drones within the swarm. The goal for this is to install virtualized sensors that relay information to the master swarm control node. Sonar, laser, and other long range detection systems are options we can use and can have their effectiveness easily measured. The number of objects a drone collides with in a given path will be how to determine efficiency of sensors and appropriate trajectory adjustments.

4.7.2 Maintenance of Formations

As the swarm traverses the urban environment, the optimal formation must be maintained to maximize our range of visual input. Otherwise, we run the risk of increasing the time taken to find the target or worse yet, missing the target entirely. There are two options on how to maintain the preset formations, both of which can also provide acceptable measurements for the object collision metric. The first option is to install long range sensors on each drone that is pointed in the direction of travel at all times. If this sensor encounters an object, the drone will adjust its trajectory accordingly without halting the movement of the entire swarm. This temporarily sacrifices the formation and possibly missing key areas, but will provide optimal results in environment traversal. The second option is to halt the entire swarm, adjust the swarm’s position and density if necessary before navigating the obstacle, then shifting back to the original path planned. This option does allow for greater efficiency in

obstacle avoidance and will maintain an optimal search arc, but will drastically increase the time it takes for the swarm to traverse the environment.

4.8 What is ROS?

ROS, which stands for Robot Operating System, is an open-source middleware package that communicates between the metal hardware of the robot and the software packages that the developer creates. Despite the name, ROS technically isn't an operating system for the computer, it's a superficial program for the robot to allow developers to communicate and modify changes between the robot and the software. To run ROS, the technical aspect requires Ubuntu 18.04 LTS and higher. With the advantage of using ROS, the developer can treat the program as portable, since the API is in uniform with each robot. Before ROS, if you want to transfer a program to another robot, the API from the manufacturer requires you to develop using their framework. This will cause tedious time on porting code, rather than testing and expanding the developed programs into other devices, this is why ROS is commonly sought for.

ROS has been commonly used by the industry, it has been reported by ABI Research in 2019, that "nearly 55% of the world's robots will include a ROS package by 2024". Companies such as Amazon, Microsoft, 3M, Robotnik, ClearPath Robotics, and even Boston Dynamics all use at least a ROS package in their systems.

ROS 1 is also known as version 1, was released on November 7th, 2007, while the first distribution was later released on March 2nd , 2010, called ROS Box Turtle. And the latest ROS 1 distribution is ROS Noetic Ninjemys, which was released on May 23rd, 2020, and the End of License Date, will be on May 2025, after that, everything will be migrated towards ROS version 2.

ROS version 1 has been developed using C++03, which is an outdated version that doesn't incorporate C++11 features in the API. It also has been developed using Python version 2, while the latest version of Python is 3.9.7, Python 2 has been deprecated since January 1, 2020.

4.9 ROS Versions

To explore which ROS version 1 package is optimal for our needs, we will be researching the various ROS version 1 distribution. For our need we need to have several packages that make it optimal to communicate and exchange with at least five drones in the gazebo environment as well if our deep learning algorithms are supported or not such as YOLO.

What we need

- Object Detection/Classification Integration:
    - Does YOLO work w/ any of the distributions?
- SLAM:
    - What are some available packages that contain SLAM that are compatible w/ ROS?
- Swarm Behaviors:
    - Possible techniques to incorporate communication exchange between drones?
- Path Planning:
    - Simple Path Planning Packages to get Started

4.10 ROS Noetic Ninjemys

ROS Noetic Ninjemys is the thirteenth ROS distribution and was released on May 23rd, 2020 and will be the last ROS distribution supported by ROS version 1. The end of license date for ROS Noetic Ninjemys will be May 2025, and then migration towards ROS 2 will be moving forward. It is focused to run on Ubuntu 20.04 LTS, however other versions of Ubuntu and flavors of Linux can be run on it.

4.11 Packages for ROS Noetic Ninjemys

4.11.2 ROS Update

YOLOv4 for ROS is available but for ROS Melodic:
https://github.com/Tossy0423/yolov4-for-darknet_ros

Possible configuration files need to be updated to be reconfigured for ROS Noetic Ninjemys from ROS Melodic, more conclusive testing needs to be done.

4.11.3 SLAM

Possible packages for SLAM with ROS Noetic:

- ROS-Noetic-SLAM-Gmapping:

```
sudo apt install ros-noetic-slam-gmapping
```

- ROS SLAM Toolbox:
    - Improved speed up of the SLAM Karto mapping library
    - Improved visualization and stability packages

- Maintained Package for SLAM w/ ROS:

```
https://github.com/SteveMacenski/slam_toolbox
```

- Hector SLAM:
  - No API Documentation for Library
    - Uses LiDAR SLAM

```
sudo apt-get install ros-noetic-hector-slam
```

4.12 ROS 1 Architecture

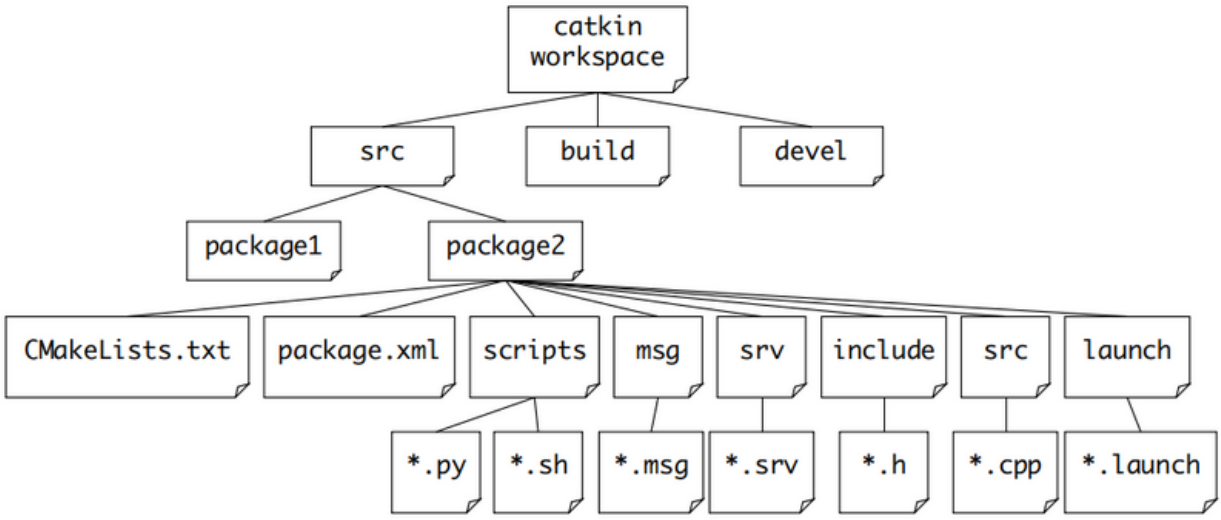
ROS 1 is composed primarily of many nodes working together through topics, the basic level of communication. These topics can be broken up even further to include services and actions. Topics are necessary for relaying code for the robot to move, understand sensor readings, and communicating with other robots. The environment allows this inter-communication of nodes to create this network of information. Everything is run through roscore, which is a collection of nodes that are required to be running if you want ROS to work.

4.12.1 ROS Packages

ROS packages are necessary for launching ROS code. They are a folder hierarchy that is made through the usage of:

- catkin_create_pkg <package name> <package dependencies>

Catkin is the name of the work space environment builder that ROS uses, below is a picture of the architecture that forms the catkin_ws:



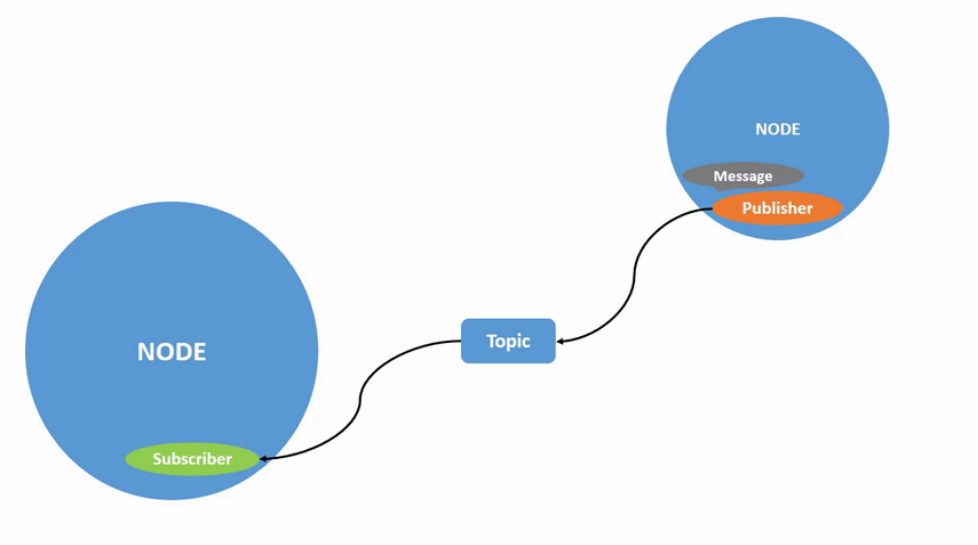
As shown a ROS package needs to contain a launch folder (where all launch files are), the src folder (where all code is stored, ROS can only read cpp and python code), a CMakeLists.txt (shows all the rules needed for compilation), and a package.xml (has information about the package and any dependencies).

4.12.2 Topics, Services, Actions

Topics

Topics are a form of communication that uses a publisher-subscriber architecture to transmit information across the environment and nodes [69]. Publishers write messages into topics and subscribers retrieve the information within topics so that nodes can use and respond to environmental changes. Useful for having information available at all times for other ROS systems. Topic message files have the extension .msg and are stored in the msg directory.

Individual components of ROS topics can be illustrated with a graph structure such as the one below:



Example of a ROS Topic [65].

Services

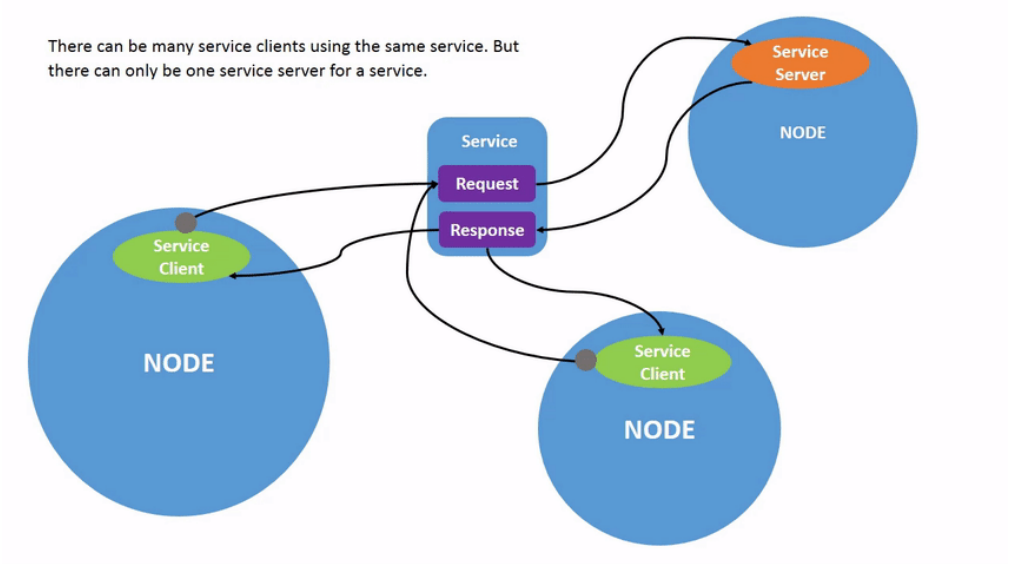
Services are a specific functionality that any node can call [70]. Services consist of the service server and the service client. The service server provides functionality to any clients who call it. The service clients are the ones that request information from the service server. Services are synchronous because a ROS program cannot continue until it receives a result from the service.

- rosservice info /service_name returns info about the following:
  - Node: The node that provides the service. (service server)
  - Type: The kind of message the service uses.
  - Args:The arguments the service takes when called.

Service messages have the extension of .srv and are contained in the srv directory.
A service message consists of three components [70]:

- ServiceMessage: The service message. It creates a connection to the service server.
- ServiceMessageRequest: The object used for creating a request to send to the server.
- ServiceMessageResponse: The object used for sending a response from the server back to the service client when done.

Individual components of ROS services can be illustrated with a graph structure such as the one below:



Example of a ROS Service [66].

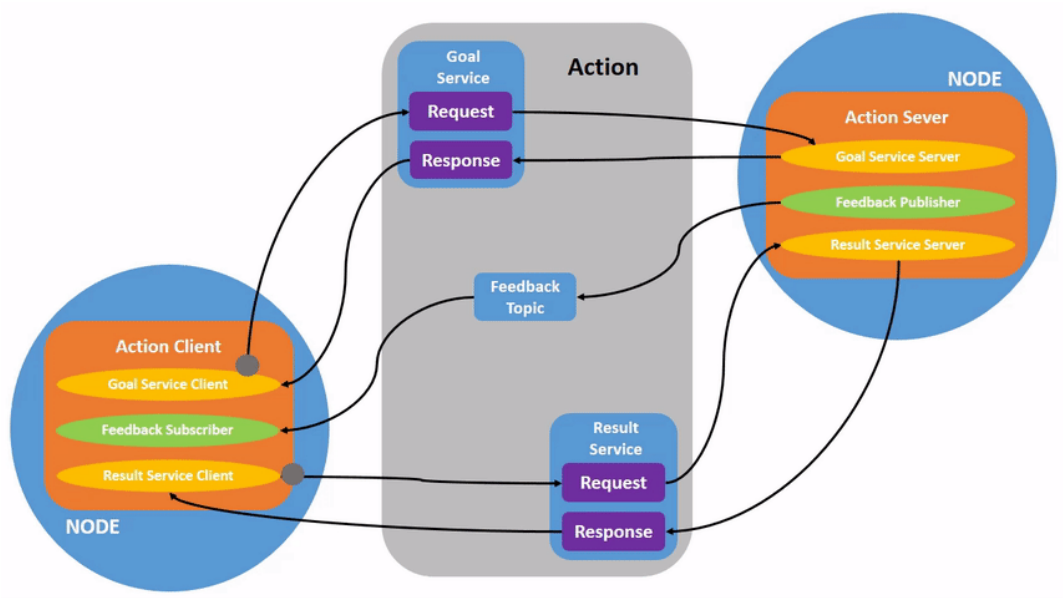
Actions

Actions also create a functionality and allow any node to call it. The difference between actions and services is that a service requires a robot's whole attention and must be finished before doing other things. Actions are more akin to multithreading, meaning that the robot can do other things at the same time that an action is occurring [71]. Actions also allow the user to provide feedback while it is being performed, unlike services. Actions are also asynchronous, which allows them to be used while other things are going on at the same time.

Defined using a .action file where it is used to define the below portions [71]:

- Goal: A wanted behavior that is sent from the action server to the action client.
- Result: Upon goal completion the action server sends a message to the action client.
- Feedback message: Server is able to send incremental information about the completion of a goal.

Individual components of ROS actions can be illustrated with a graph structure such as the one below:



Example of a ROS Action. [67]

Messages

ROS messages take the form of a data structure with its own defined attributes. These messages are what are transmitted through topics, actions and services. The information then can be accessed through rospy calls in the node's source code.

ROS Code Information:
Needed for all python files:

```
#! /usr/bin/env python
```

Creating a node:

```
rospy.init.node('node name')
```

Creating and using a publisher:

```
<publisher name> = rospy.Publisher('/<topic name>', <message name>, queue_size=<number>)
```

```
<publisher name>.publish(<var>)
```

Creating and using a subscriber:

```
<subscriber name> = rospy.Subscriber('/<topic name>', <message name>, callback)
```

Callback is a defined variable that will transmit info to subscribers from outside sources.

Common looping for ROS:

```
rospy.spin()
```

Basic launch file:

```
<launch>
        <node pkg="package name" type="python code name" name="node name"
output="screen"/>
</launch>
```

Launch files can contain multiple nodes, launch files, and more. This allows for a lot of versatility when creating programs for ROS.

Useful ROS Console Commands

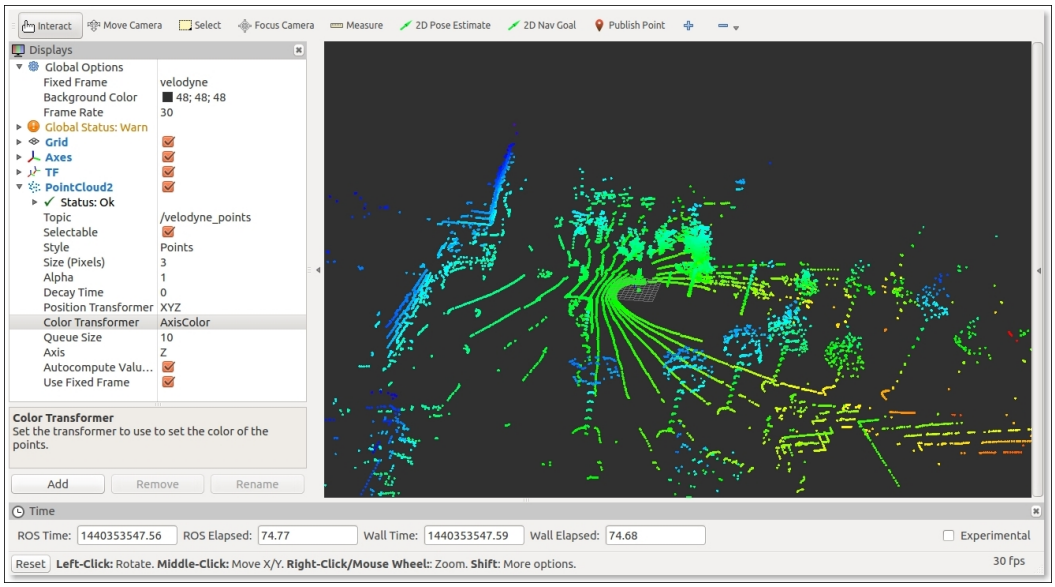
Command	Description
roscd <package name>	Directly move into the relevant package directory.
rospack list (| grep <package name>)	List all current available ros packages. If grep is included, checking if a specific package exists in the list.
mkdir <directory name>	Create a new directory in linux
touch <filename>	Creates a new file in linux
chmod +x <filename>	Gives a file execution permissions.
rospack profile	Refreshes ROS to check for new packages.
catkin_make	Compiles the entire src folder in the catkin_ws folder.
catkin_make --only -pkg -with -deps <pkg	Compiles a specific package.

name>	
source devel/setup.bash	Sources the compilation into setup.bash. Required after performing either catkin_make or catkin build.
catkin build	Accomplishes the same thing as catkin_make just more verbose.
catkin build <package name>	Builds a specific package
rm -rf build/devel	Perform this if you want to use catkin build, and you already used catkin_make.
rosparam list	Lists all parameters stored in the parameter server.
rosparam get<param name>	Get the value of a specific parameter in the parameter server.
rosparam set <param name><value>	Set the value of a specific parameter in the parameter server.
roscore	Launches the main process in ROS.
export | grep ROS	Displays all linux environment variables for ROS.
catkin_create_pkg <pkg name><dependencies>	Create a ROS package
rosrun <pkg> <script name>	Run a specific script file.
rostopic list	List all available topics in ROS.

rostopic info <topic name>	Provides the information on a specific topic.
rostopic echo <topic name>	Shows the traffic coming from a specific topic.
rosmsg show <msg name>	Shows the structure and datatypes of the information sent in a message.
rosservice list	Shows a list of all available services in ROS.
rosservice info <service name>	Show the information pertaining to a specific service.
rosservice call <service name> (Double TAB)	Call a service in ROS, the double tab will request any needed arguments.

4.12.3 RVIZ

RVIZ is a built in visualizer for ROS which shows a 3D view of the robot and the sensors connected. It is very helpful for debugging sensor related issues and seeing how the robot reacts in certain environments that gazebo doesn't display.



4.13 Path Planning

4.13.1 ROS Navigation Stack

The ROS Navigation Stack is a set of concepts and packages that allow a robot to perform path finding in their surroundings. There are 3 main stages in the stack that will be broken down into the topics, plugins and packages needed to create this stack. These concepts and steps will be the foundation of how we get our drones to individually create their paths at first then transfer the planning to a swarm master node to coordinate the paths between the other drones. The first stage is configuring the navigation stack.

4.13.2 Navigation Stack Configuration Concepts

Using tf2 transformations with the robot

Originally tf was used in the configuration stack but it was marked as deprecated and is replaced by tf2. Now tf2 uses a broadcaster and a listener in most cases along with a frame for any robots on the map and a world frame. These frames are used as reference so that the paths created don’t crash into obstacles or other robots that may be on the field. An important tool to use when creating broadcasts in tf2 is the view_frames.py tool. This tool generates a graph diagram of how broadcasts refer to frames that are in the drone environment.

Another tool that is important for keeping track of transform is the tf_echo tool. This tool is essentially run in the command line to check the transform between any two frames in the environment. But the main tool that will always be used is rviz. Rviz is a ROS visualization tool that can layout all attributes of a given robot in an environment. For example, the user can see scan output from any sensors, camera feeds etc.

Writing a local path planner plugin for ROS

Now since ROS assumes that the robot is not one of the demo units we will need to write a custom plugin for our drone, otherwise we can skip this step. However it is important to understand the structure of the plugin which is written in C++ , and XML. The components of the plugin are a header, source, plugin body, plugin registration. Once the programmer has created and tested these components it can be called inside a node or launch file. For more details on the code examples and demo content you can refer to the ROS wiki page [19]

Tuning Navigation

There are many things to consider and keep track of when creating the navigation stack so that means that many things can go wrong. Some of these things can be traced to the odometry of the robot, the localization variables and frame, sensors and other components that are needed to make navigation run smoothly. According to ROS wiki, three things to mainly pay attention to are the range sensors such as laser scans or LiDAR, odometry, and localization. So for aunty debugging and adjustments to make the drone smoother will likely reside in one of these three components.

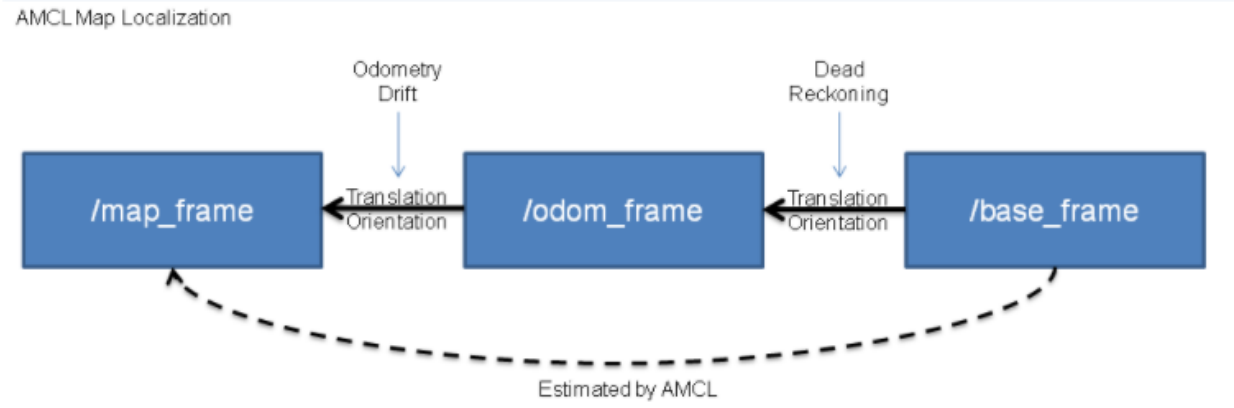
Range sensors

One of the easiest ways to check if there is an issue with a sensor is to see if it is working correctly in Rviz. Usually Rviz has the ability to generate an error that pertains to why the sensor is malfunctioning. It might be that it is not being published and has no corresponding topic (publisher).

Odometry

Odometry is an important topic to make sure is accurate and stays accurate. This can be dictated by other ROS packages referred to as AMCL. AMCL is a localization system that is implemented in ROS to obviously keep track of the robot moving in 2D. It uses adaptive KDL sampling or Monte Carlo localization (Dieter Fox) algorithms to keep track of the pose of the robot in a known map. As far as troubleshooting goes and fine tuning goes, this is where the programmer can change many parameters that have to

do with the overall filter, the laser model, and the odometry model. Now AMCL also takes into account the transforms that are performed. The /base_frame publishes a transform to /odom_frame and the finally to /map_frame but AMCL meanwhile is estimating the /base_frame to the /map_frame as shown below from the ros wiki.



In the figure, there are two types of drift shown, dead reckoning as well as Odometry drift. These are both calculated in AMCL to modify the data passed between the publishers. [26]

Configuration Parameters for AMCL in ROS

These are reference in the constructsim course [16]

Overall Filter Parameters

- *Min_particles* (integer; default: 500) - minimum number allowed to use.
- *max_particles* (integer; default: 2000) - maximum number allowed
- *Update_min_d* (double; default: 0.25m) - the translation movement needed before performing an update on the filter.
- *update_min_a* (double; default: 0.2 rads) - rotational movement needed before performing an update on the filter.
- *resample_interval* (integer; default: 1) - number of filter updates needed before performing a resample.
- *transform_tolerance* (double; default: 1.0 sec) - time to post date the transform that was published, that way it can be checked that the transform is valid in the future.

- *recovery_alpha_slow* (double; default: 0.0 (disabled)) - the exponential decay rate for the slow average weight filter. This is commonly used when evaluating random poses. If used it is recommended to use 0.001.
- *recovery_alpha_fast* (double; default: 0.0 (disabled)) - the exponential decay rate for the fast average weight filter. This is commonly used when evaluating random poses. If used it is recommended to use 0.1.
- *set_initial_pose* (bool; default: false) - makes the AMCL set the initial pose from the *initial pose parameters instead of waiting for the inital_pose messages to come in.
- *Initial_pose* (Pose2D; default:{0.0, 0.0, 0.0}) - the x, y, z (meters) and yaw(rads) coordinates of the initial pose for the robot's base frame in a global frame.
- *always_reset_inital_pose* (bool; default: false) - defines that it is needed to provide an initial pose to the AMCL either via the topic or the initial pose parameter (granted that *set_inital_pose* is set to true). Otherwise this will default to the last known pose given to AMCL.
- *save_pose_rate* (double; default: 0.5 Hz) - max rate in hertz to store the last pose and covariance to the parameter server with the following names; *~initialpose* and *~initial cov.* This saved pose will be used on other runs to initialize the filter. (-1.0 to disable this parameter)

Laser Model Parameters

- *Laser_min_range* (double; default: -1.0) - minimum scan range to be considered for calculations. -1.0 will cause the laser's reported min range to be used as the returning value. Therefore by default a laser min scan range is usually used.
- *laser_max_range* (double; default: 100) - max range of a scan that can be considered. -1.0 will cause the laser's reported max range to be used as the returning value.
- *max_beams* (integer; default: 60) - the number of evenly spaced beams to use in each scan when updating the filter.
- *_z_hit* (double; default: 0.5) - mixture weight for the z_hit part of the robot model.
- *z_short* (double; default: 0.05) - mixture weight for the z_short part of the robot model.
- *z_max* (double; default: 0.05) - mixture weight for the z_max part of the robot model.

- *z_rand* (double; default: 0.5) - mixture weight for the z_rand part of the robot model.
- *sigma_hit* (double; default: 0.2 meters) - standard deviation for the gaussian model that is used with the z_hit part of the robot model.
- *lambda_short* (double; default: 0.1) - exponential decay parameter used with the z_short part of the robot model.
- *laser_likelihood_max_dist* (double; default: 2.0 meters) - max distance to do the obstacle inflation on the map for usage in the likelihood_field model.
- *laser_model_type* (string; default: “likelihood_field”) - define what type of laser model to use: beam, likelihood field, or likelihood_field_prob (same as likelihood_field just with beamskip feature).

Odometry Model Parameters

- *robot_model_type* (string; default “differential”) - establish what model to use “differential” or “omnidirectional”.
- *alpha1* (double; default: 0.2) - approximates the expected noise inside the odometry’s rotation estimate of the rotational component of the robot’s motion.
- *alpha2* (double; default: 0.2) - approximates the expected noise inside the odometry’s rotation estimate of the translational component of the robot’s motion.
- *alpha3* (double; default: 0.2) - approximates the expected noise inside the odometry’s translation estimate of the translational component of the robot’s motion.
- *alpha4* (double; default: 0.2) - approximates the expected noise inside the odometry’s translation estimate of the rotational component of the robot’s motion.
- *alpha5* (double; default: 0.2) - only used if the model is “omnidirectional” but is a translation related noise parameter.
- *odom_frame_id* (string; default: “odom”) - specifies which frame to use for odometry.
- *Base_frame_id* (string; default: “base_footprint”) - specifies which frame to use for the robot base.
- *global_frame_id* (string; default: “map”) - the name of the coordinate frame that is published by the localized system based on the map.
- *tf_broadcast* (bool; default: true) - if needed we can set this to false to prevent the AMCL from publishing the transform between the global and odometry frames.

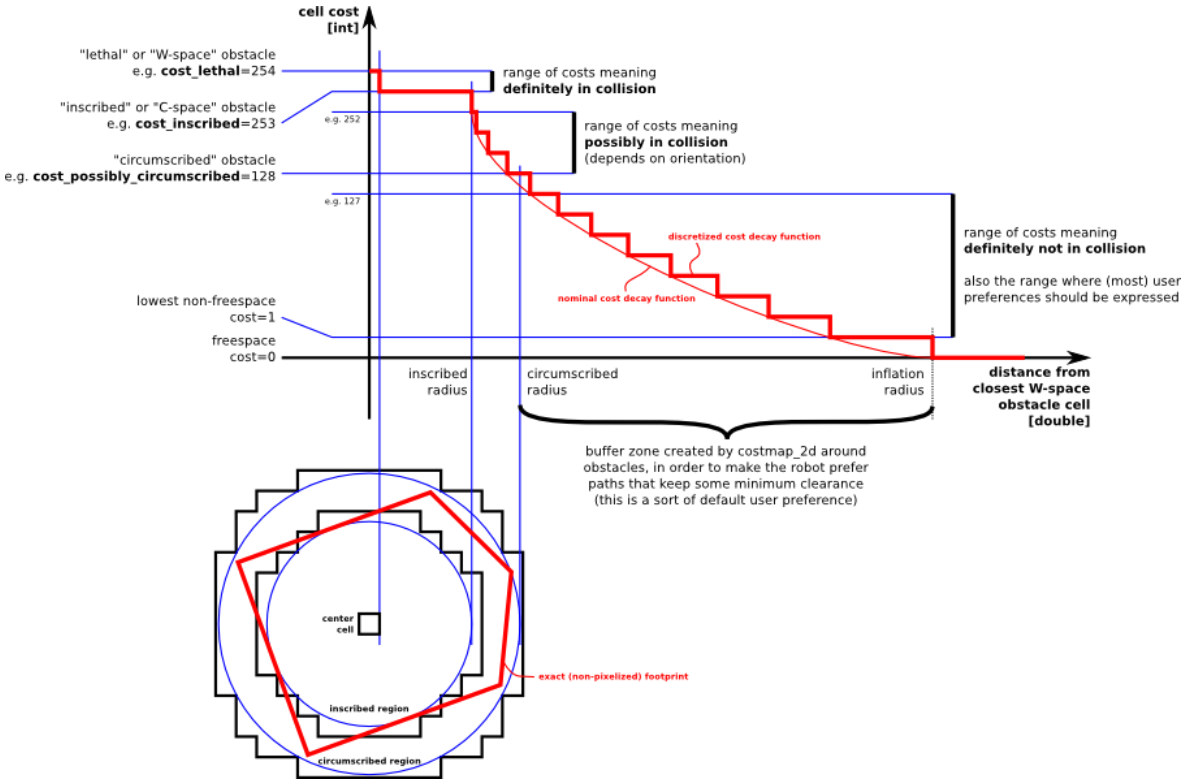
Localization

It is important to make sure that localization is performing correctly. Now usually this is addressed by adjusting parameters mentioned above. However, to verify that the map is drawn correctly through AMCL, it is best to run gmapping, which is another ros package that is an OpenSlam wrapper that helps build a 2D floor plan of the map that the drone is occupying. The map is built from pose and sensor data that is provided from the robot. So it is wise to manually drive the robot through the map to get an idea of what the map looks like. Like AMCL there are many parameters that allow for tuning these mappings which can be referenced in the ros wiki [31]. Note that OpenSLAM uses a few different SLAM algorithms like ORB-SLAM, cartographer, or visual SLAM. It is also very important to set an initial pose for all robots in the map.

Costmaps

For the purposes of all practical usages, there are only 2D costmaps that are used in ROS. There are conceptual discussions on the application of 3D costmap, but there are no practical applications yet. Costmaps subscribe to all the sensors that are on the robot to keep updating the map so that it can avoid obstacles. These decisions are published and contribute to how the path of the robot is determined. Costmaps determine what spaces are safe to move to which aren't through defining the cost of a space. There are 5 definitions of cost that are used. [25]

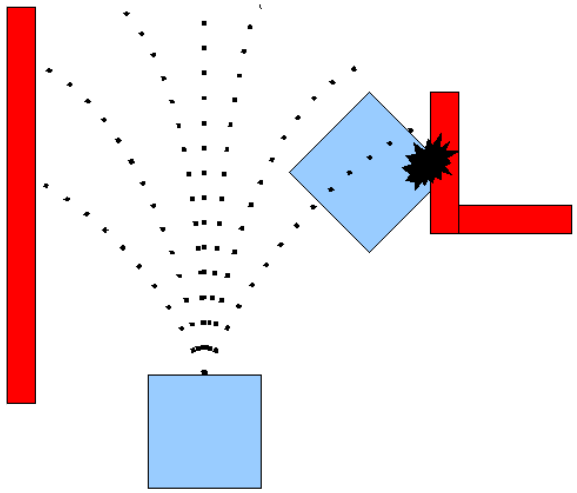
- *Lethal* - if the robot hit one of these spaces there would be a definite collision.
- *Inscribed* - this space is less than the inscribed radius so that means if the robot's center is within the space then there is a definite collision with an obstacle.
- *Possibly circumscribed* - if the center of the robot is in this region, depending on the orientation of the robot there could be a collison.
- *Freespace* - the cost is 0 and nothing would cause a collision for the robot in this region.
- *Unknown* - there is no information for this space. The user can determine if that is a safe place for the robot to travel.



This figure relates all the costs to the frame of the robot. Another interesting thing about the Costmaps is that just like any other package it can be fine tuned for a better map generation and also will help improve the performance of the path finding components. [25]

The Local Planner

This planner is represented in ROS as a package called dwa_local_planner, where dwa stands for dynamic window approach. The package takes a global path plan and the costmap and then generates velocity parameters to publish to the robot's base or in this project's case the rotors of the drone. The basic idea of the dwa algorithm is to sample the current control space velocities. Then for each sample perform a simulation of what would happen if that was applied to the robot for a short time. It then scores each simulation and discards any collisions. Finally it picks the best simulation and publishes the velocity. Then it is just a matter of repeating the process when necessary.



Example of the DWA algorithm in action. This link also leads into the parameter definitions of how to calibrate dwa_local_planner. [24]

In summary, to correctly implement the hector drone to perform paths in a map that is to simulate an urban environment we will need to use the concepts put forth by ROS. This means we will have to tailor our parameters with the idea in mind to avoid other drones, and obstacles that they may encounter in the map. It also has to have the precision to check alleys or side yards for the target, which means the costmap will have to be appropriate to perform navigation in such a narrow space.

4.13.3 Frontier Exploration

Frontier exploration is the concept of creating frontiers of unknown regions for a robot to explore. Each frontier is given a cost to help determine what path will be the most beneficial to take. This type of exploration takes advantage of SLAM and the ROS navigation stack to autonomously navigate and map an unknown environment. In ROS this package is known as explorer_lite [69]. However, it in no means takes advantage of any swarm intelligence or coordination with other drones so this is an initial step to map the environment before performing the search.

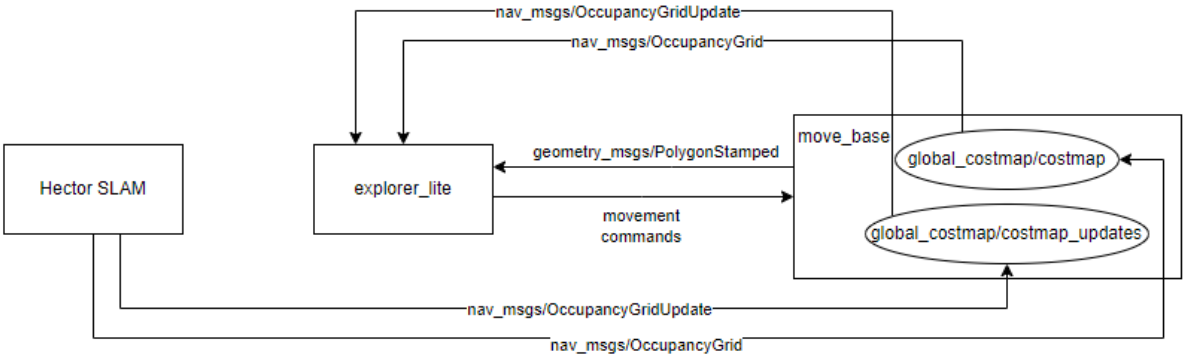


Diagram describes how explorer_lite fits into ROS navigation stack. This is a variant of what you see on the ros documentation since costmaps are used in place of standard map. [69]

Once the map is drawn using this method it is saved and ROS Navigation and AMCL are used in place of the explorer_lite package. The benefits of this is to reduce resource overhead when running all drones and allow us to run the search more effectively. In summary, frontier exploration is the first step before autonomously searching for the target.

4.15 Multiple Drone Pathfinding

The design the team has in place allows for the swarm controller to essentially modify any given drones path depending on where they are in Gazebo's 3D space. Each drone will be creating their own path independently for the most part. Then the swarm layer is going to provide the following parameters for the paths to follow.

- Any given drone should not cover the same ground another drone is covering. (No intersections in paths if possible.)
- If a drone detects the target, but cannot achieve the correct confidence in identification. The swarm controller should be able to modify the drone's path to increase confidence or change all paths to focus on a smaller zone and get a better angle.
- Swarm controllers should be able to monitor the SLAM and path data to make decisions for the drones and be able to modify the data to adjust for new paths that follow a swarm pattern.

The actual pattern of the swarm should prioritize detection of a target. In the earlier section where we covered the more conceptual algorithms for a swarm, a few of the insect based patterns for ants and bees seemed to be more insightful since their goal as a swarm was to find food, nectar and other beneficial resources. The goal seems very similar to what our swarm is trying to achieve.

For our approach on a high-level is that there will be a separate package that will act as a swarm controller. Since each drone will be taking care of its own path as it travels through the map the swarm has to dictate the ground covered in the path relative to other drones, what to do if the path is complete and the target has not been found, as well as if something happens to one of the drones. In each condition, it is a good idea to work out how each of these situations will be dealt with.

- Ground covered: Based on the decided initial positions the swarm controller will have to make sure that the drones are efficiently covering the correct ground.
- Endpoint decision: if none of the drones are able to find the target a new path should be drawn until it finds the target. If a drone does find the target the new path should either be corrected to get a better angle or a new drone should be moved to get the angle. If the drone finds the target within parameters of object detection, the drones should halt.
- Drone malfunction: If a drone malfunctions in the swarm, the swarm controller should abandon communications to it and focus on the still running drones.

4.16 ROS 2 Challenges Story

4.16.1 What is ROS 2?

Just like the prior description of ROS for version 1, ROS 2 also stands for robot operating system that is open-source middleware that communicates between the metal hardware and the software packages that developer program.

ROS 2 updates on what ROS 1 was great about and improved on what it didn't do well. Old, deprecated packages and API that was built on the foundation of ROS 1 are no longer supported and need to be removed for ROS 2.

Instead of Python 2 for ROS 1, Python 3 was used in mind for ROS 2.
Instead of C++03 for ROS 1, C++11 features were used in mind for ROS 2.

4.16.2 ROS 1 vs. ROS 2

There were several motivations for our decision to go with ROS 2. ROS has not been popular within the industry due to its lack of support for real-time, safety, certification, and security, all of which are major requirements for industrial participants. As such, part of the developers' goal with ROS 2 is to meet those requirements and make ROS much more industrial friendly. Since our project is geared towards industrial and research applications, we believed that ROS 2 would be the best version to integrate it in.

It is also worth noting that as of 2025 the final ROS 1 release will lose all support and developers will fully migrate to ROS 2. Since this project is ongoing and may be improved upon in future senior design projects, we felt that developing the project in a version of ROS that will continue to see long term support is most beneficial for future teams. For technical changes, there are several components of ROS 2 that our team felt would be better to work with than those in ROS 1.

In terms of programming language support, ROS 2 has full integration of Python 3 and its associated libraries whereas all but ROS 1's Noetic distribution supports Python 2 exclusively. Programming libraries key to the development of object detection like Tensorflow, Pytorch, and other machine learning libraries are designed to integrate with Python 3.

Another key difference which motivated our decision is that ROS 1 uses the roscpp and rospy libraries which provide support for C++ and Python 2 plugins. An important note is that these two libraries are independent of each other and as such,

certain features of ROS 1 are exclusive to one library or the other. ROS 2 eliminates this limitation by integrating the RCL layer, mentioned in section 4.1.1, which means that ROS 2 functions can be used much more easily between the C++ and Python 3 libraries. It also allows new functionalities to be more easily implemented since they only need to be compatible with the RCL base library.

In ROS 1, it was difficult to integrate multiple Nodes in the same executable file with intra-process communication. With the introduction of Nodelets, this made the process significantly easier to do. In ROS 2, this is directly included in the core functionality and has been relabeled “components.” Handling multiple nodes within a single executable file is incredibly useful as a single drone may have multiple components that need to pass information between each other and even outside the drone to the object detection and path planning software. We can then start all components from a single ROS 2 launch file and remove any communication overhead that we would have had from previous ROS 1 distributions.

Launch files are one of the most important parts of any ROS application in either version. These files allow developers to start all our nodes from a single file and pass arguments to our components without much difficulty. While this is common across both ROS 1 and ROS 2, ROS 2 allows launch files to be programmed in Python 3 rather than in ROS 1’s XML. This allows ROS 2 launch files to be far more modular and customizable. Since we will need to have extreme flexibility when integrating swarm behaviors, this will be incredibly useful for the project.

Also mentioned in section 4.1.1, ROS actions were not part of the ROS 1 core functionalities, but an add-on that was driven by the need for asynchronous service communication. The issue for synchronous services is that they could experience a deadlock, where two services are unable to advance in their operation because of a constant feedback loop of communication. In ROS 2, actions are now part of the core features and have full support with both the C++ and Python 3 libraries.

4.16.3 ROS 1 vs ROS 2 Conclusion:

- Many packages support older packages (e.g., ROS Indigo, Kinetic) that are deprecated, and since ROS Noetic is the last distribution, most packages aren’t bothering with updating to focus effort there, rather into ROS 2 due to LTS (Long Term Support).
- ROS 1 uses inefficient running mechanisms such as requiring the network of nodes known as roscore to run the whole system, while ROS2 only doesn’t do that and has completely gone over to using a DDS middleware, which is faster in execution.

- ROS 2 has very few packages and resources such as SLAM, ODS, Swarm Behaviors, which will make it difficult to transition into the project.
- ROS 2 also has 0 - 1 public packages that are available from 3rd parties for drone support. PX4 has a ROS 2 bridge for drone support, however, migrating and connecting the packages on top of PX4 will add future headache in long run.
- ROS 1 has more available packages that are still in use, as well as fully documented guides and tutorials online while ROS 2 has little to none available resources.

4.16.4 Motivations for Regressing to ROS 1

- As previously mentioned, the lack of packages and community support is a huge downfall for continuing to work on drones.
- Despite only having PX4 - a third party plugin package - that has a ROS 2 Bridge with PX4, has no available resource for packages that support Drone Support.
- We spent over a month on incorporating a drone into the Gazebo environment and ROS 2, however, with little success, such as adding URDF files for ROS 2 and SDF for Gazebo, we only have a drone to instantiate into the environment. We were able to apply force to our rotors through the Gazebo environment directly, which barely picks up the drone with a lot of force, it still fails.
- We were then able to turn on the rotors by acting as “wheels”, which looks like it’s turning clockwise, however it’s only adding enough force for wheel support, which is not enough for rotators.
- ROS 1 has a lot of resources and community support for Drone Movement, ODS, Swarm Behaviors and SLAM.
- Overall, the team agrees that we shouldn’t focus on the physics of the drone, and more on the overall objective of the project: ODS, Swarm Behaviors, and SLAM.
- ROS 1 has strong support in the caliber as well as help in assistance at UCF Robotics Research Center - Dr. Gita Reese Sukthankar - has only experience in ROS 1.

4.16.5 Building ROS on Ubuntu 20.04 LTS

The following steps were provided via the ROS wiki Page [32]. Which you can refer to for more specific commands whereas this document will contain the more general steps.

1. Configure your repositories in Ubuntu to include the ros noetic distribution.

2. Update Ubuntu using the following command.

```
sudo apt update
```

3. Install using the command.

```
sudo apt install ros-noetic-desktop-full
```

4. Setup the environment by sourcing the noetic *setup.bash.*

```
source /opt/ros/noetic/setup.bash
```

5. Build the dependencies that you need to run ROS.

```
sudo apt install python3-rosdep python3-rosinstall python3-rosinstall-generator
python3-wstool build-essential
```

6. Run rosdep.

```
sudo apt install python3-rosdep
sudo rosdep init
rosdep update
```

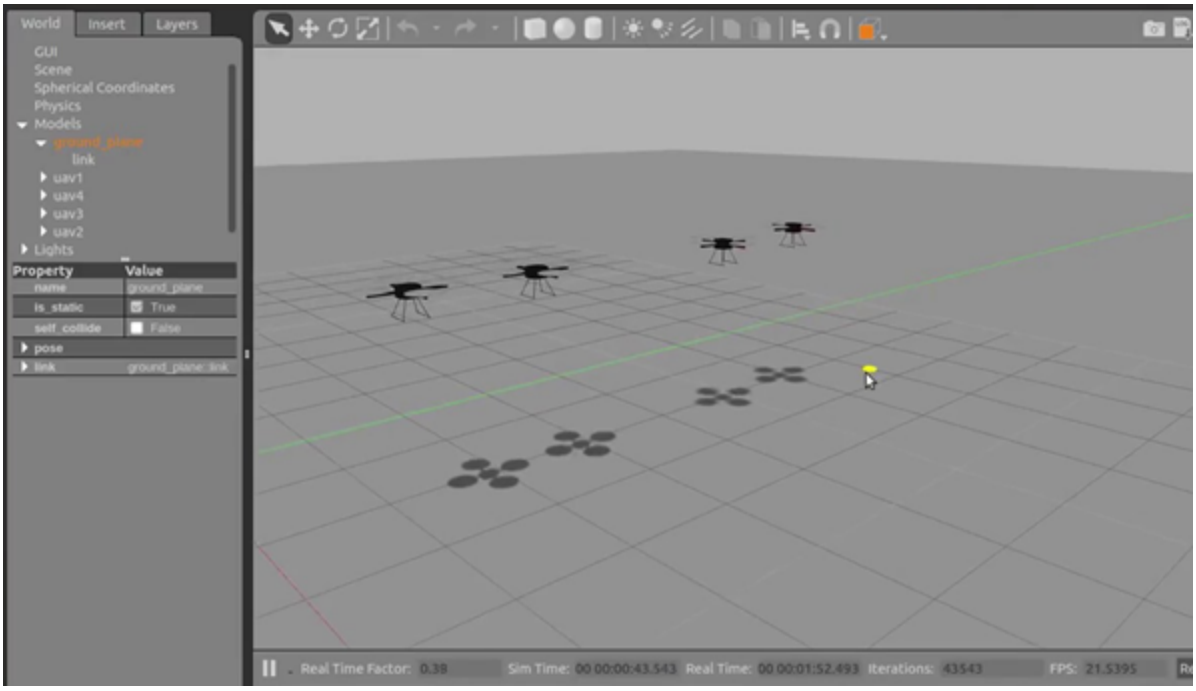
7. Build the workspace and source the *devel/setup.bash.*

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

4.17 Gazebo

4.17.1 What is Gazebo?

Gazebo is a robot simulation application that allows testing robots in a 3D virtual environment. Testing various algorithms such as path finding solutions, object detection and classification, and swarm behaviors can all be possibly done within the simulator. This can save a lot of money and time before being implemented in the real world. Getting to test and mess around the environment without the worries of configuration to hardware, and just have it ready for testing is what makes Gazebo a great platform for ROS.



Screenshot of Gazebo application running five drones

Gazebo is currently on the latest version 11.0.0. which was released on January 30th, 2019. Although we are integrating Gazebo with ROS 2 – Foxy Fitzroy, Gazebo can operate independently from ROS, Gazebo supports multiple plugins that can substitute the necessary functionality of ROS into Gazebo.

4.17.2 Installing Gazebo on Ubuntu

1. Simply use the following two commands

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

2. Then you need to add the ros packages to integrate with gazebo using the following command.

   ```
   sudo apt install ros-foxy-gazebo-ros-pkgs
   ```

5. Real Implementation

5.1 Spawning a Single Drone in Rviz and Gazebo

Prior to spawning a single drone, ROS and Gazebo must have access to their respective model files. Note that while ROS and Gazebo access the same model parameters for generating and interacting with, their file formats are different. ROS uses the Universal Robot Description File(.urdf) type. Gazebo uses the Simulation Data File(.sdf) type. Both file types contain code almost identical to XML code which is used to specify the physical body of the robot, sensors, collision properties, and other information. We must detail this information so other ROS packages can safely interact with the model inside Gazebo.

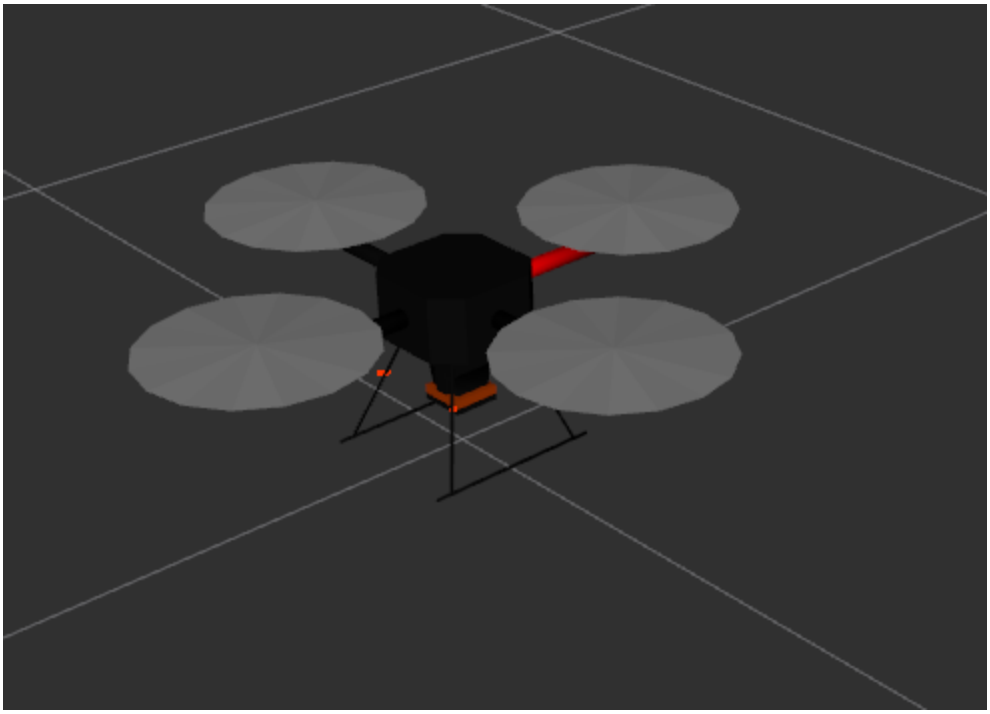
5.2 Handling Meshes within URDF and SDF File Types

Our previous attempts at spawning a drone proved unsuccessful. Gazebo has a premade quadrotor model that incorporates a mesh file(.dae) within its SDF. Despite attempts at importing the mesh file into both the SDF and newly created URDF files, ROS showed repeated error messages of being unable to properly find it. Our solution was to use a set of pure URDF and SDF files that did not have any dependencies.

5.2.1 Spawning the Drone

Hector Drone

The Hector drone is a commonly used drone in ROS simulation. There are many existing deployments of it for previous versions of ROS. Currently, we have adapted this package for ROS Noetic which entailed modifications in plugins and function calls that have been deprecated in the latest version of ROS and Gazebo. After these modifications, the Hector drone can render in an environment already loaded with a camera, an IMU sensor and laser scanner. Which means it has the basic components to easily implement pathfinding through the ROS navigation stack. The original code for the drone is a git repository. [34]



Hector Drone rendered in Rviz.

From this point on, launch files, object detection and path planning will have to be adapted to with a new map that the Hector drone has not been in before as well as adopting a new navigation stack to make sure it is up to date with ROS Noetic. It is important for the team not to get hung up on the implementation of the physical simulation. The scope of work that needs to be done is the path planning, the ODS and the swarm behavior. Therefore as long as the license headers, which define the conditions of reuse are met then there is no issue with using the drone.

Each drone will have their own makeup of nodes, services and topics. The key to making multiple drones lies the ability to make sure we don't overlap important components during spawning. If that happens then there will be communication issues with the Swarm node that is commanding the drones to navigate and search a certain way. That is why it is pivotal to make sure there is a deep understanding of the components of a single drone before spawning in multiple versions. ROS has standards in the software to document and display the different aspects of a drone. The RQT graph shows all the interactions between nodes, topics and services. The view_frame shows the transformation tree structure of a given robot.

5.2.2 Working with Multiple Drones

Spawning the Drones

From what has been researched so far the actual execution of spawning multiple drones in Gazebo via ROS Noetic is a matter of how the launch file parameters are defined the actual spawn. Below is a breakdown of just spawning two drones in the same space. However, this can be propagated to the desired 5 drones that are needed for the project.

Each drone should be in a tag called group with a defined namespace (ns) which in this case should be drone_#. Inside the group tag, we will need to define the a few items

- The robot description using a urdf.xacro file.
- The node for the robot_state_publisher
- Parameter for the publishing frequency
- Parameter for the tf_prefix. This is very important for distinguishing different drones. The value should be drone_#.
- Define the spawning node and the x,y,z, and yaw values to position it appropriately in the world for Gazebo.
- Allocate a corresponding rviz file.
- Launch the takeoff/ land package.

In the initial spawning the context of manipulation of each drone will still be singular so there will need to be a node(s) to establish the paths of each drone that reflect the concepts of a swarm and execute a search for the target [33]. Initial spawning encapsulates any spawned topics and nodes into a given namespace using a group tag in any given launch file.

Each drone that is spawned should contain topics and nodes that pertain to: hector_slam, amcl, explorer, YOLOv5, boid swarm and any other critical packages included in the workspace. For checking that these items are allocated under the correct namespace is right after any roslaunch command. The user can review what has been created when running a launch file in ros.

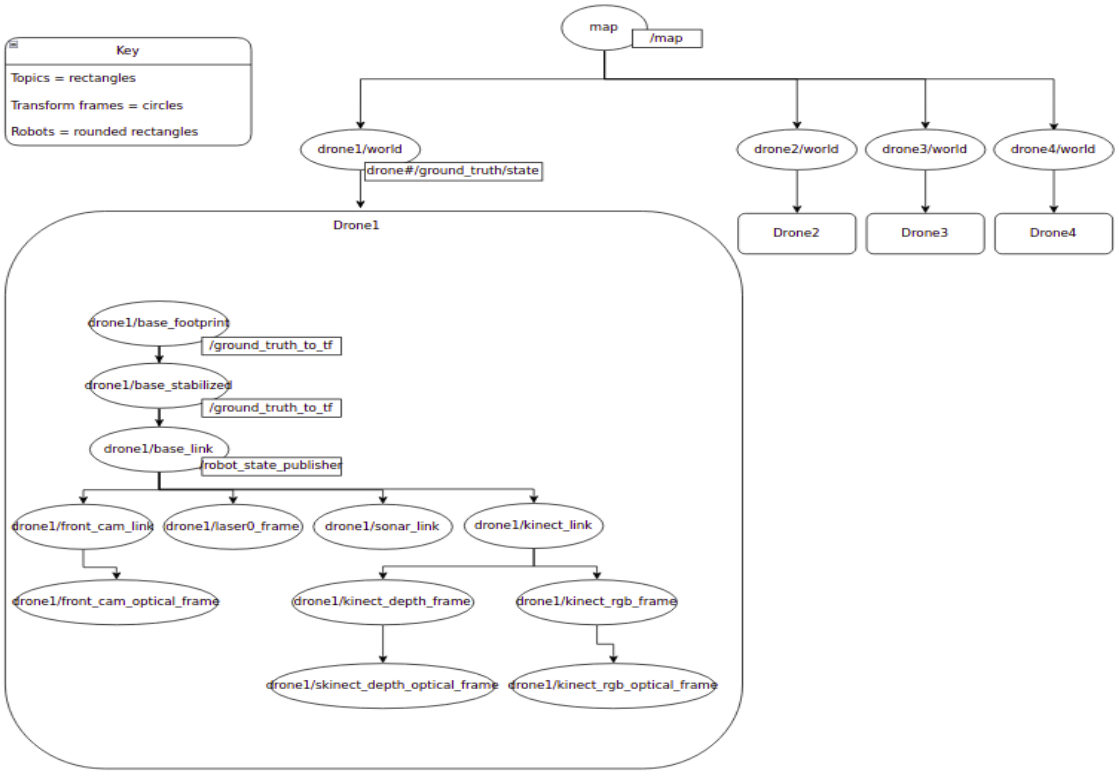
Adding frontier exploration and Navigation to Multiple drones

After spawning the drones, the next issue is how to properly implement the hector slam nodes and explore_lite nodes. The most important thing to consider first is the tf_tree of the drone. Each drone's tree should have a head node denoted as

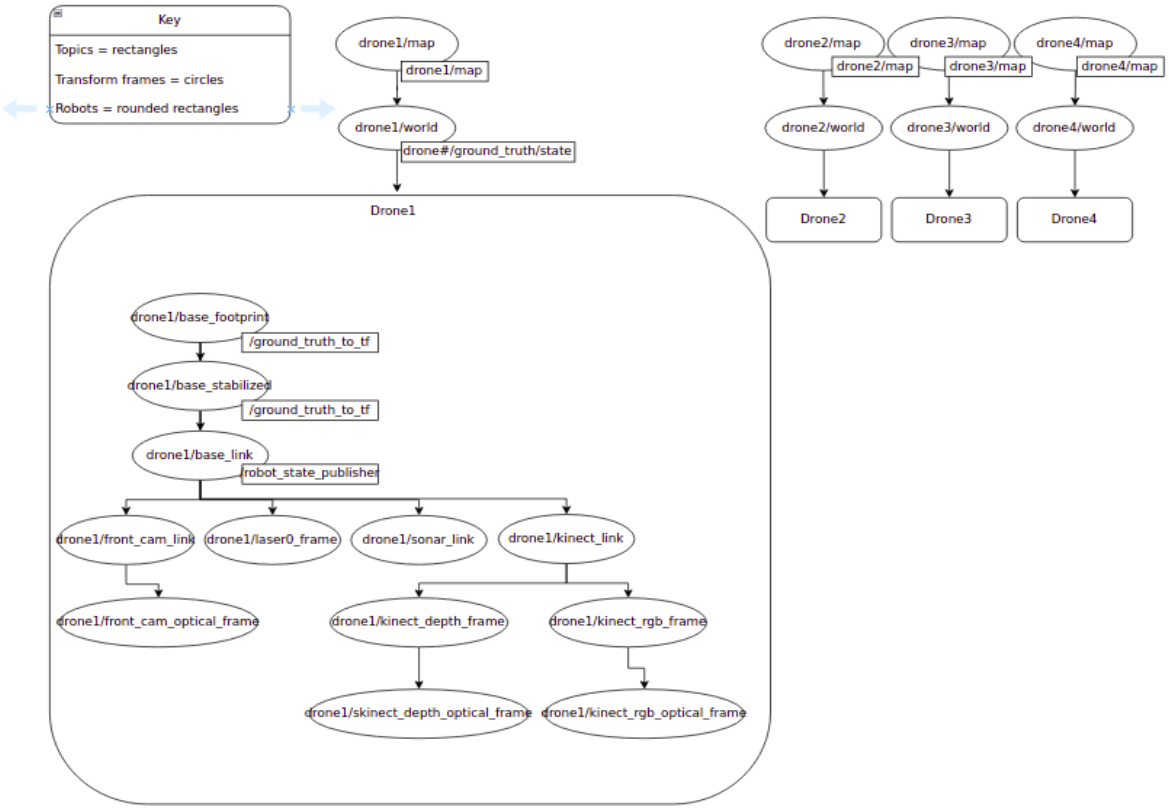
drone#/world level. This contains the odometry for the drone and creates a tree of cohesive frames from the body to the sensors.

As mentioned before, when working with ROS launch files we can use the tag <group> to create a namespace around a newly created node. So each hector slam node that is instantiated has to be under the correct namespace otherwise there is miscommunication and/or overlap. This also applies to the explorer_lite node and the move_base nodes. Once this has been confirmed as correct by the user. Each drone will begin to draw the map relative to their spawning position. From there, we combine the autonomous scans to draw a complete map of the area. Once the map is drawn, we switch gears to the navigation and swarm aspect of the system. Here we need to make sure that amcl nodes are relative to each drone and connect it to the map server (Note that amcl creates the connection between map and drone#/world). Below are the structures of the tf_tree given each scenario.

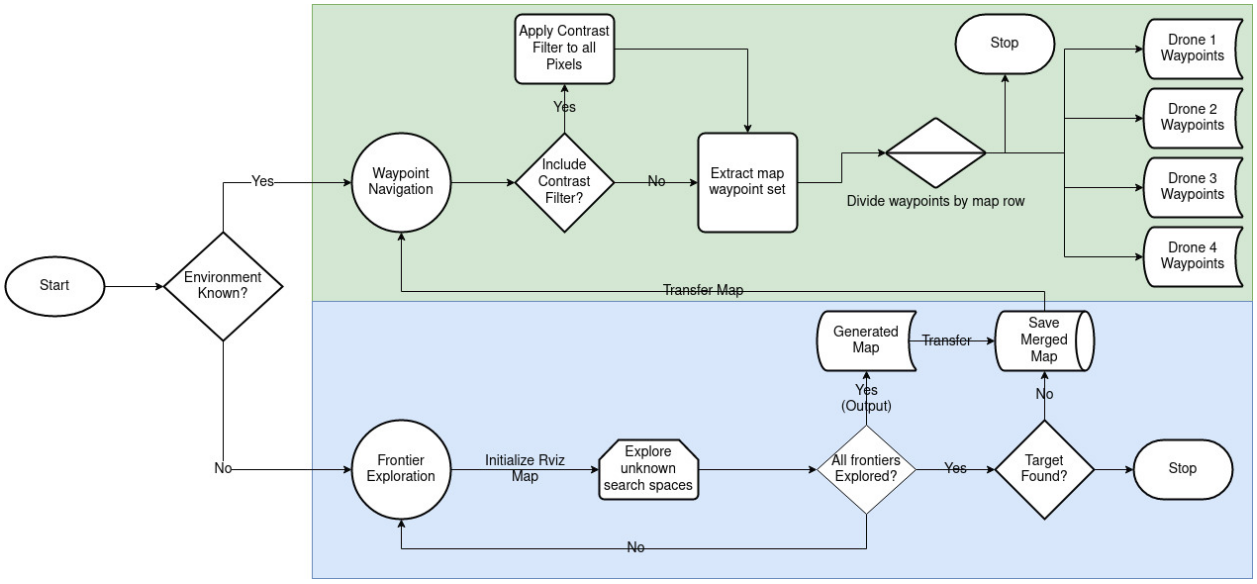
Frontier exploration TF Diagram



ROS Navigation TF Diagram



Project Hierarchy



YOLOv5 for Multiple Drones

YOLOv5 was very easy to implement because of PyTorch Hub. A model can quickly be downloaded from PyTorch Hub and loaded with the trained weights file in just a few lines of code. You can get the model stored locally to eliminate the need for downloading every time.

Each drone requires a subscriber to access their respective camera feeds, and each subscriber is a thread. The YOLO model itself is thread-safe, so we do not need to worry about race conditions occurring during inference. Once inference is done, the results are returned as a *pandas.DataFrame*. We can parse through the DataFrame to get the corners values of the bounding box and draw it using *cv2.rectangle*. Similarly, we can extract confidence and class name from the DataFrame to draw over our image. The processed image can be saved to an array that functions as a buffer before we display it to the screen with *cv2.imshow*.

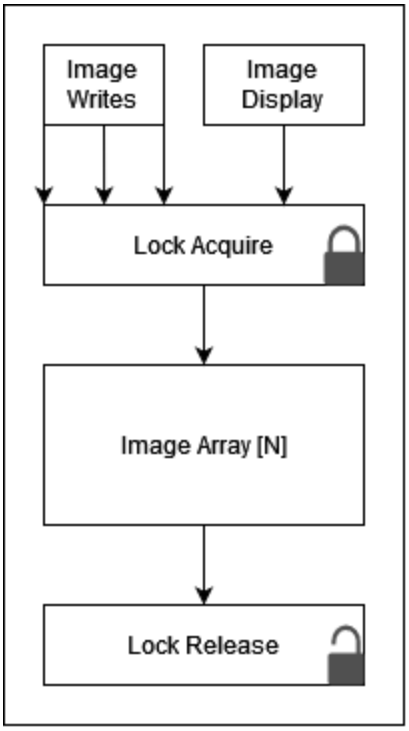
Multiple threads are writing to different sections of the array. However, *cv2.imshow* will need to read the array, so the entire buffer needs to be protected using a *threading.lock* object. Whenever a thread wants to write to the buffer, a lock must be acquired prior to writing. Similarly, a lock must be acquired before *cv2.imshow* is allowed to read data from the buffer.

Originally, each drone thread was supposed to be capable of calling *cv2.imshow* and not need a global buffer. However, *cv2.imshow* is not thread safe, so multiple threads calling the function was not permitted. The solution was to make a single thread dedicated for *cv2.imshow* to display images.

A simpler solution would be making an instance of YOLO for each drone, but each instance of YOLO requires 2.5 GB of VRAM. Needing to propagate that to 4-5 drones, our computers could not handle it, which led to the multithreading solution.

Note:

- Our YOLO inference script is flexible enough to handle theoretically an arbitrary number of drone inputs. The script is hardware limited.
- You can have too many drones for a single instance of YOLO, and the GPU cannot perform inference fast enough. You would be required to create another instance of YOLO to divide the workload accordingly.



5.2.3 World Implementation

Urban World

The world that we are using was provided by the previous team that attempted the autonomous drone swarm project. The world consists of a city block and a suburban neighborhood. Inside this world we will spawn 5 drones and 1 target which will be the droid BB-8 from Star Wars™. Next steps pertaining to the world would be generating a costmap to begin the pathfinding process for the navigation stack.

Upon completion of the design there will be a master launch file that will handle spawning in the drones, spawning in the world and initializing the tools for pathfinding and object detection. This master launch will also initialize the swarm navigation to have the drone behave in a way that allows them to search an area effectively.

5.2.4 World Segmentation & Waypoint Management

Running frontier exploration standalone presents the risk that the target will not be found as the system explores and maps the environment. Thus, we elected to counteract this using the Lean Manufacturing method of Error Proofing by prevention. A secondary phase was implemented which will take the swarm on a planned path throughout the entire environment. At the start of phase 2, the system has a completed occupancy map of our environment which will be used to generate a set of waypoints for our swarm. We have two similar methods of doing this. The first method is a straight parse implementation with no image preprocessing. The second method applies a contrast min/max filter to avoid possible errors during map segmentation.

Method 1: No Pre-processing Method

We will designate this map *M*. The objective is to generate a list of waypoints across the entire map. We do this by parsing map *M*, segmenting it into individual tiles, each with a single waypoint plotted in the tile's center.

We start by intaking map *M*, which is stored as a .pgm file. To visualize the results of the algorithm, we make a .png copy of the map and visualize the information extracted from the pgm version. Both the original map file and its copy are the same size, so waypoint outputs are valid when used on the original map. Using the Python OpenCV library, we can make the copy and convert it into a Numpy array for ease of processing. The mathematical model for extracting waypoints from the map is defined below.

Variable Legend	
I	Image represented by RGB pixel values
W	Set of all computed waypoints
τ	Tile defined by points (x_1, y_1), (x_2, y_2) containing a set of RGB defined pixels
T	Set of all mapped tiles
C	Set of center points of Tile T defined by location (x, y)
t_y	Tile Height
t_x	Tile Width
D	Set of all drones in the swarm
l	A single pixel RGB value within I

We initialize image *I*, which was taken from phase 1 as a 2-dimensional numpy array of pixel values in RGB format.

$$I = \begin{bmatrix} l_{1,1} & l_{1,2} & \ldots\ldots & l_{1,i_x} \\ l_{2,1} & l_{2,2} & \ldots\ldots & l_{2,i_x} \\ \ldots\ldots & \ldots\ldots & \ldots\ldots & \ldots\ldots \\ l_{i_y,1} & l_{i_y,2} & \ldots\ldots & l_{i_x,i_y} \end{bmatrix}$$

I is defined as a tuple of 3 values containing the integers for Red, Blue, and Green. We first extract all tiles containing a mean RGB value greater than 250. In the drawn map, unoccupied areas are designated as white, object edges are black, and unexplored areas are light gray. An RGB value of 250 indicates a white area for the navigation system and allows for correct waypoints to be mapped.

$$T \Leftarrow \forall \tau \in I \text{ such that } \mu(\tau) > 250$$

Once the tiles are extracted, the system calculates the center of each tile using the below formula. These coordinates are then stored into a 1-dimensional numpy array for later processing.

$$C \Leftarrow [T_{i,x} + (t_x / 2),\ T_{i,y} + (t_y / 2)]$$

Lastly, we convert each plotted waypoint to the corresponding location in Rviz. The converted waypoints are then stored in memory.

$$W_D \Leftarrow [\lfloor C_{i,x} \times 0.1 - 150.0 \rfloor,\ \lfloor C_{i,y} \times 0.1 - 150.0 \rfloor]$$

Depending on the size of the map, the waypoints will be divided among the drones based on the number of rows within the search space. If phase 1 is skipped, then all drones will start at the left side of the map by default. Once the waypoints are plotted, they will move across the map in a wave like motion.

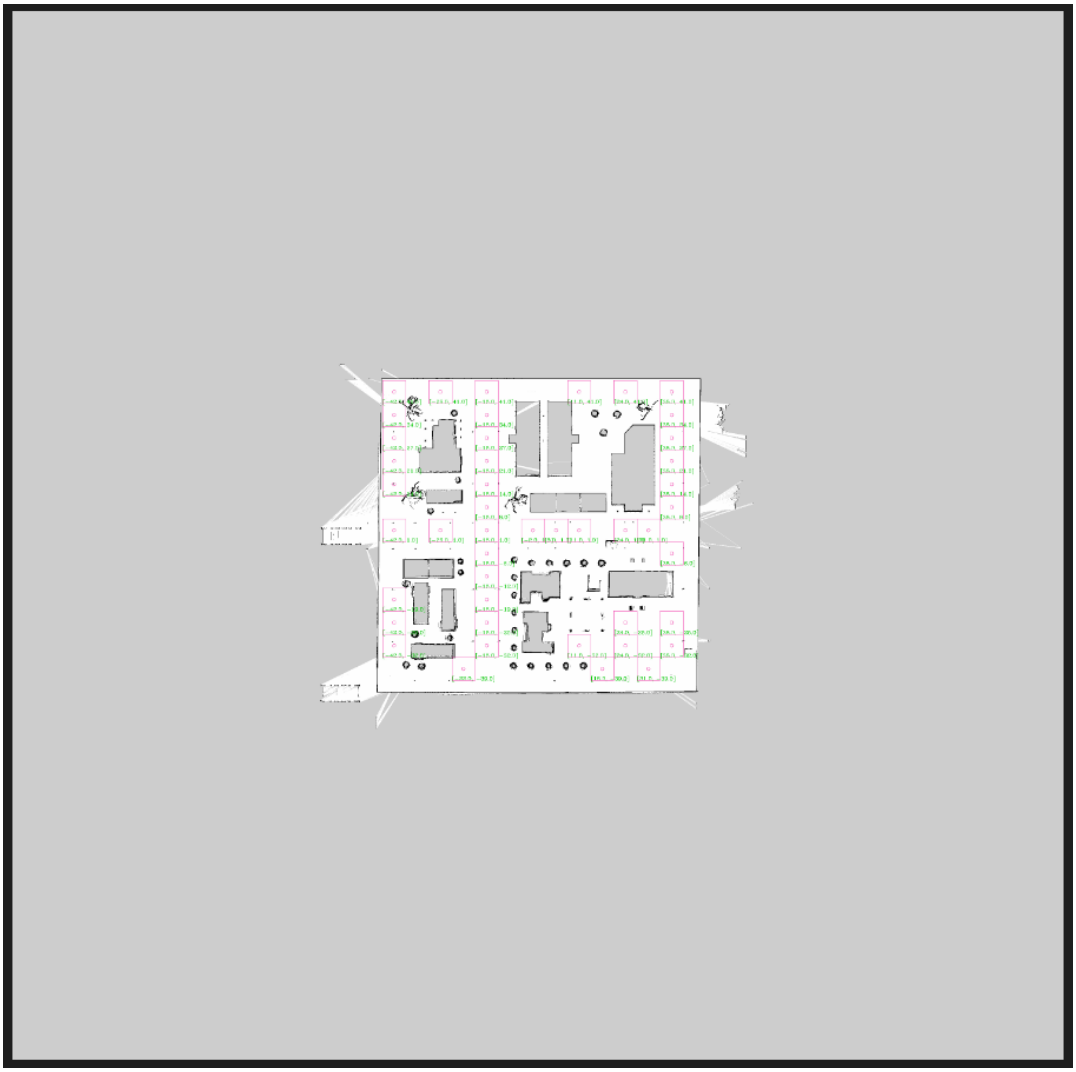


Figure 10.2.5.1: Fully processed image of urban environment map. Note the red boxes are the extracted tiles.

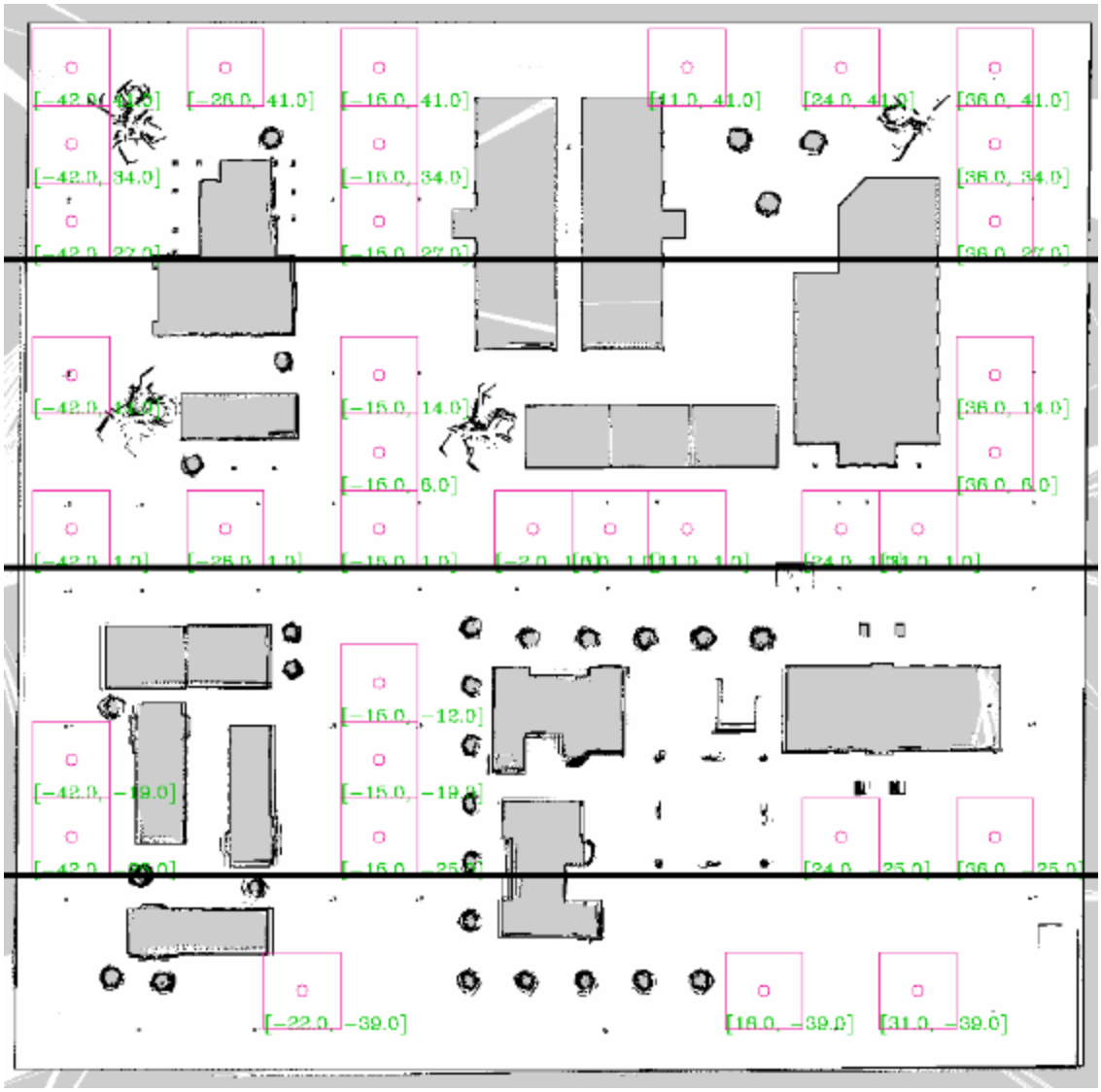


Figure 10.2.5.2: Visualization of plotted waypoints. Note the black lines are cut-off points where below waypoints are written to the next drone.

Method 2: Navigation w/t Pre-processing

The second method utilizes a contrast min/max filter to minimize error when calculating the mean of tiles. As shown in Figure 10.2.5.1, maps delivered from phase 1 have gray pixels for unexplored areas. Grayscale RGB values are (128, 128, 128) and when mixed with key areas in the search space during tile extraction, these pixels can cause noise and reduce accuracy. Thus, we set every non-white RGB tuple to 0 which removes any noise during extraction. However, this process requires extensive computation time. For a 3000x3000 pixel image, the average processing time is 58 seconds, which is unacceptable when the swarm is under the time constraint

requirement. Further development of the algorithm may show an efficiency increase when a probability kernel is implemented to process groups of pixels simultaneously.

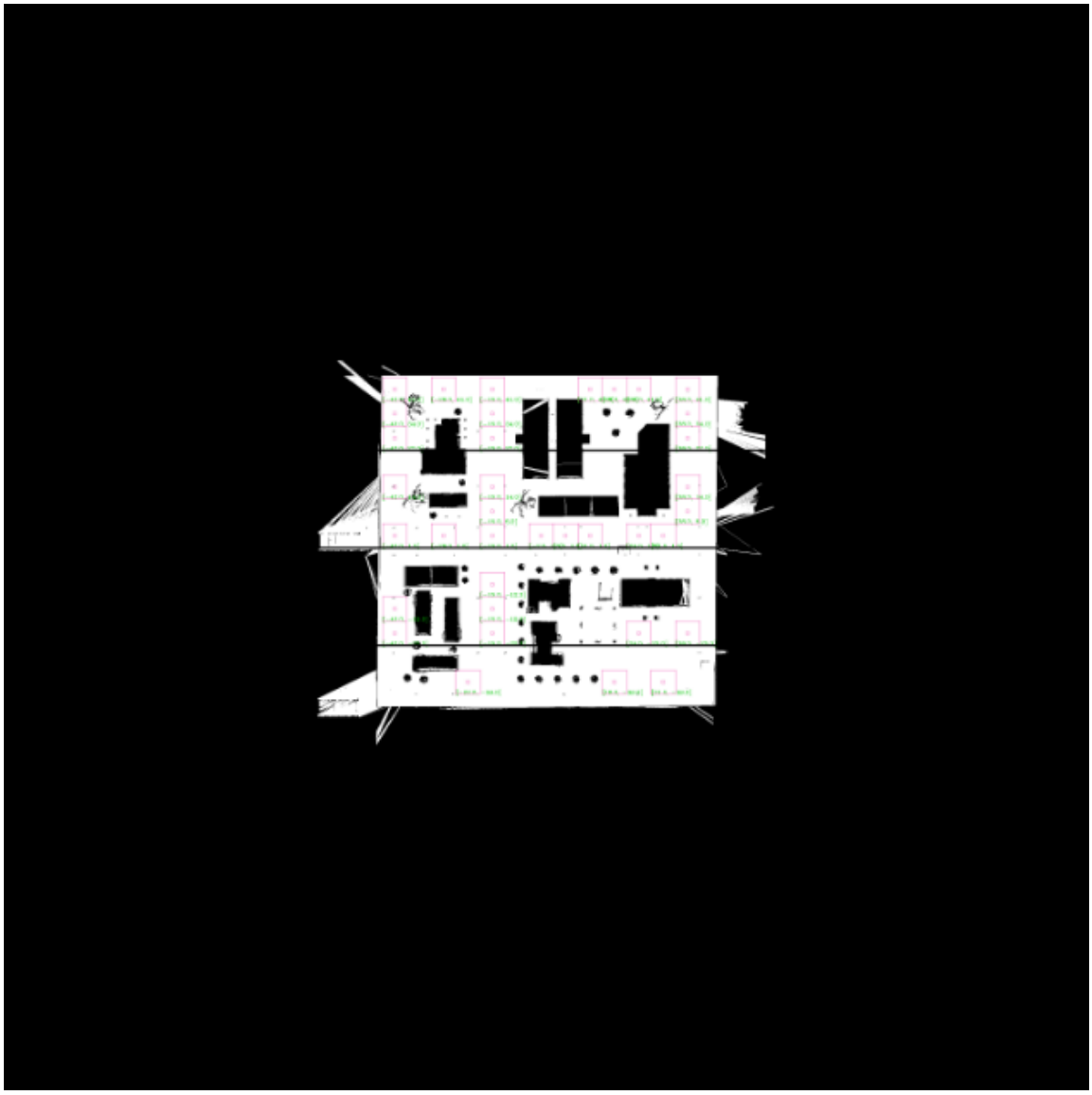
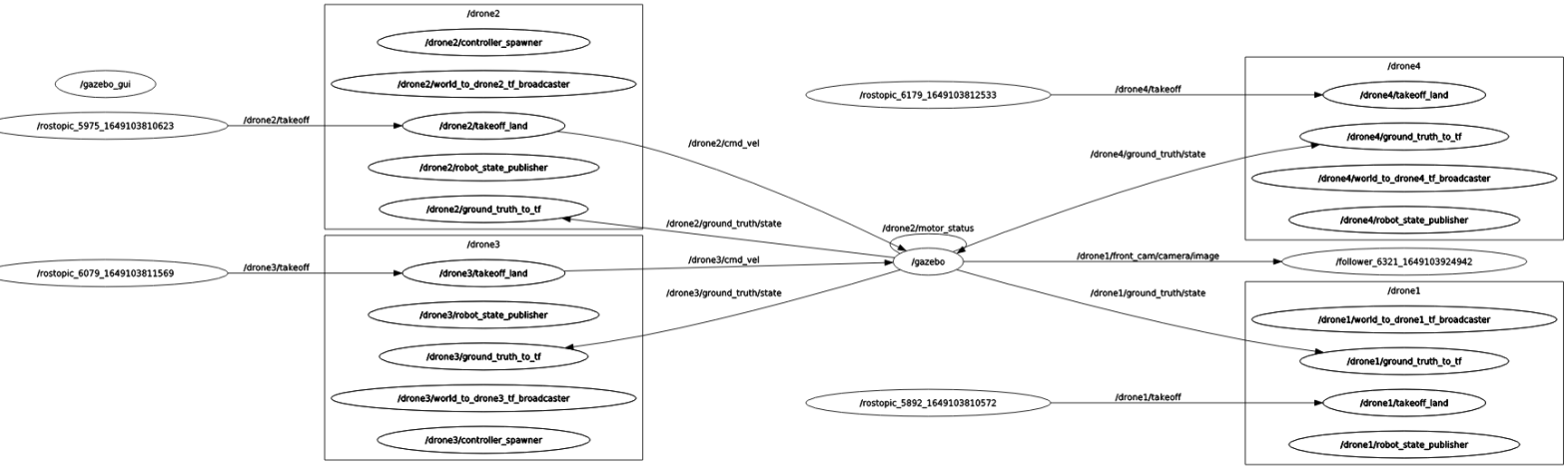


Figure 10.2.5.3: Pre-processed image. Note all gray pixels have been reduced to RGB value 0 to eliminate noise.

6. Diagrams

6.1 Single Drone RQT Graph

ROS also has a standard style of design tool for mapping the current nodes, services and topics that are in use for any single instance of a robot. This RQT Graph is directly generated from the ROS noetic environment.



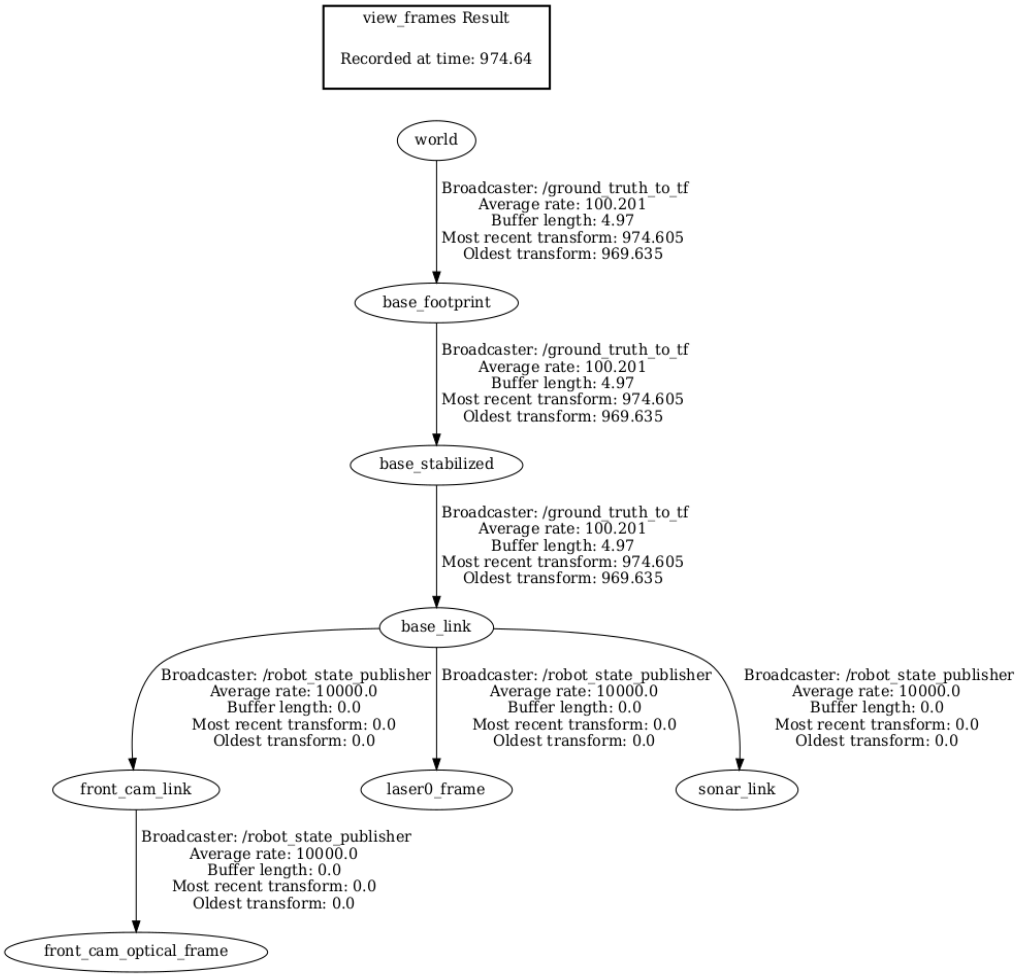
Note: May not show up well in documentation. To regenerate this diagram use the command rqt_graph when running the entire system.

6.2 Single Drone view_frame diagram

This view frame was generated from a single drone using the tf2_tools, this is directly from a live instance of a drone in Rviz. There are a few components to be aware of when reading a view_frame diagram according to the ROS wiki. [27]

- The recorded time is the timestamp of when the view frame is taken.
- The broadcaster is the name of the node that published the transform to one of the components.
- The average rate is the frequency at which the broadcaster is publishing transforms.

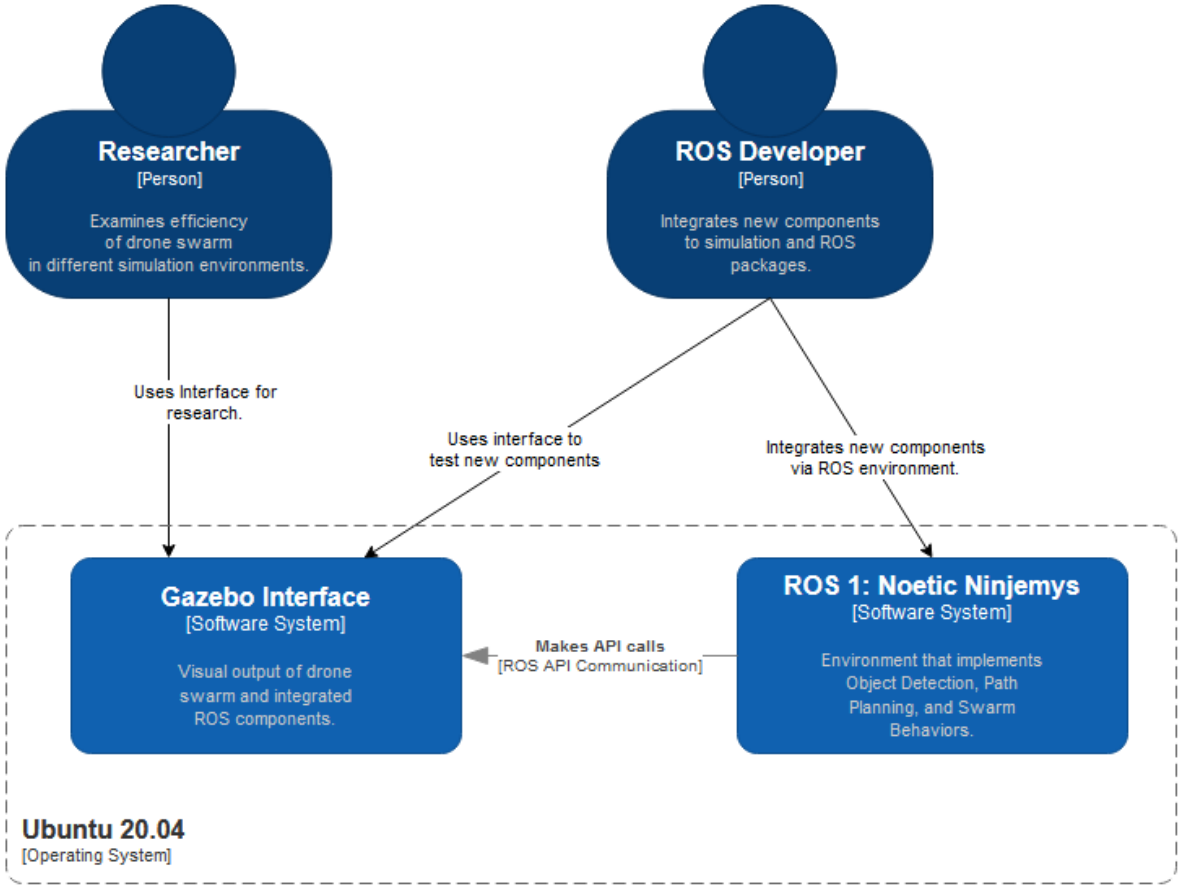
- The most recent transform is a timestamp for the last broadcast received at the component.
- The buffer length is how long the data spent in the tf buffer.



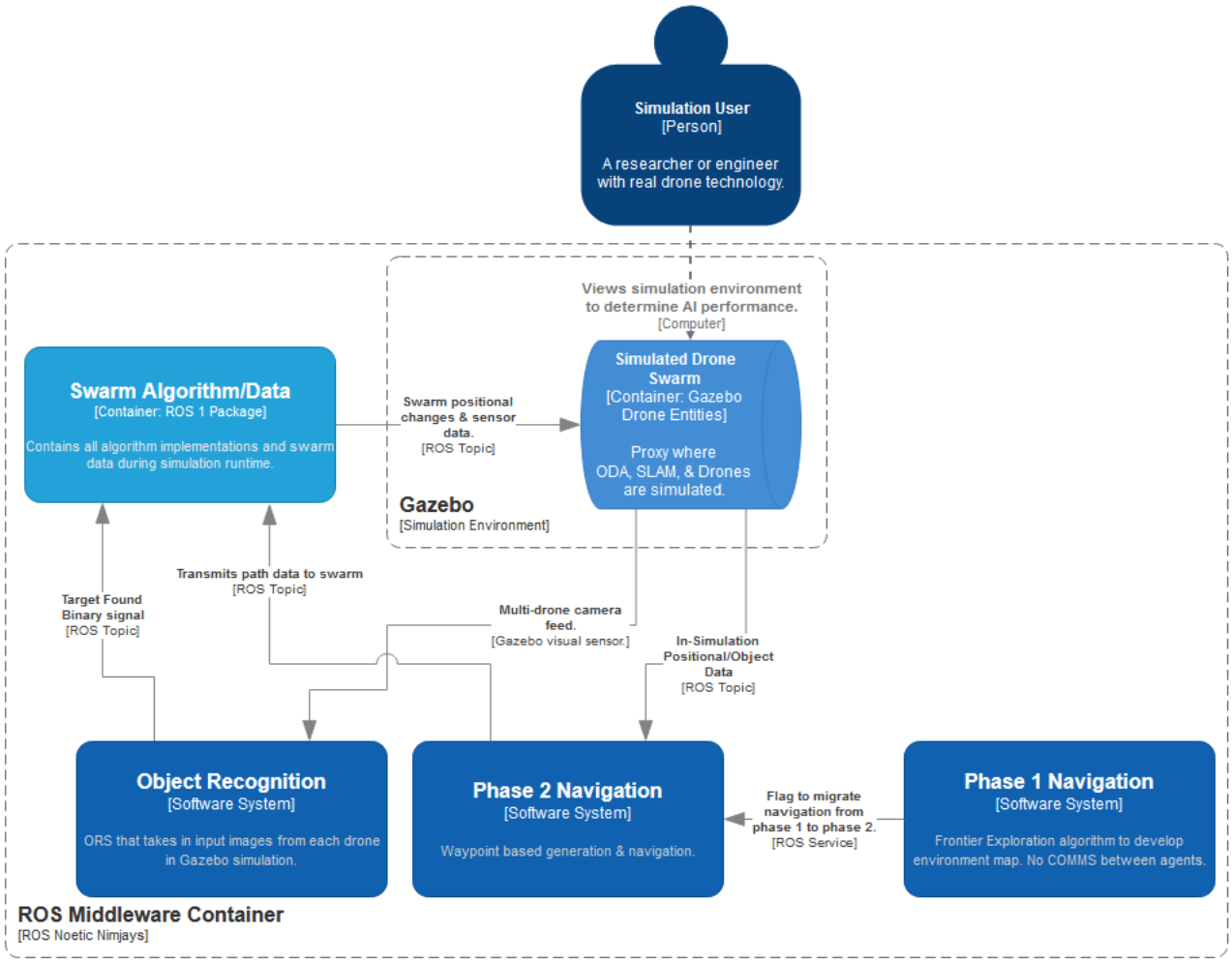
6.3.1 Context Diagram

We anticipate that the only users for our software will consist of those using our algorithms for extended research and other ROS developers integrating new elements. In time, we can expect a remote interface for any extended users like emergency

service personnel. However, integrating the interfaces for these users is outside the scope of this project and thus not included in the context diagram.

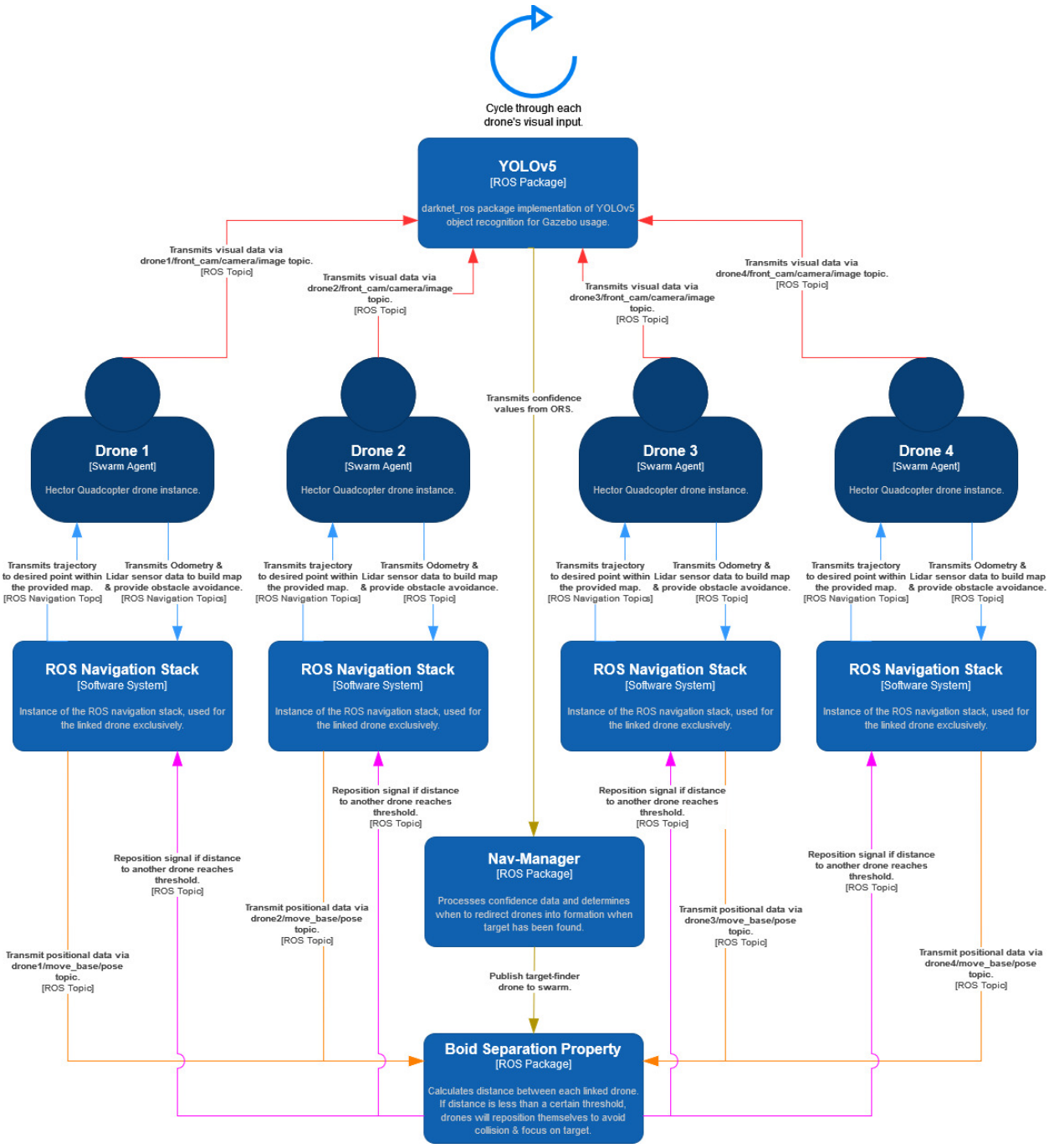


6.3.2 Container Diagram



Proposed Container diagram with existing software components. We define a container as "a separately runnable/deployable unit that executes code or stores data."

6.3.3 Drone Swarm Component Diagram



7. Project Budget

Project Ledger and Estimated Costs		
Item	**Necessity**	**Cost**
Course Materials and Lessons	To educate the team on the ROS and Gazebo software so the project can be implemented.	~$50 / month / person
Robot Operating System	To implement key components of the drone swarm including object detection, path planning, swarm behaviors, etc.	$0
Gazebo	Allows for all ROS implemented components to be used in a visual environment for extended testing.	$0
Pytorch	To assist in the implementation of the object detection model for all trained targets.	$0
Tensorflow	To assist in the implementation of the object detection model for all trained targets.	$0
Ubuntu 20.04	To run the combined ROS Galactic Geochelone and compatible Gazebo environment.	$0
Discord	To provide effective communication between team members.	$0
Github	To assist in effective version control of all user defined code for all components.	$0
JIRA	To assist in the continued documentation and organization of project phases, progress tracking, and other key project management components.	$0
GPU	To maximize runtime efficiency of all simulations since personal computers	$0

	of all team members are insufficient to run both ROS 2 and Gazebo	
Final Estimated Cost	N/A	$300

8. Assumptions and Constraints

8.1 Assumptions

Project Assumptions	
Item	**Description**
Budgeting Resources	UCF has agreed to provide a small amount of funding to the team for learning resources and project management software to simplify our processes.
ROS Minimum Software Components	We assume that ROS contains at least the minimum support for drone simulations so the team can maintain focus on the original scope of the project without requiring auxiliary work.

8.2 Constraints

Project and Personal Constraints		
Item	**Project Impact**	**Workaround**
Major gap in team member knowledge on the development environments being used.	Will delay completion of project milestones and possibly the final completion of the project until sufficient knowledge has been gained.	All members will complete specified courses according to their assigned tasks no later than November 12th.

Software being used requires computing resources that few team members have access to.	May decrease the quality of project deliverables since testing will require more time, allowing for less test cases and test runs of the software.	Store all development on an external virtual machine and allow for remote access to GPU power.
Learning platform requires moderate funding to use.	Some members may not be able to pay for this learning platform, delaying completion of certain tasks.	Either search for alternative learning resources or ask UCF for a small budget detailed in section 6's ledger.

9. Project Timeline and Milestones

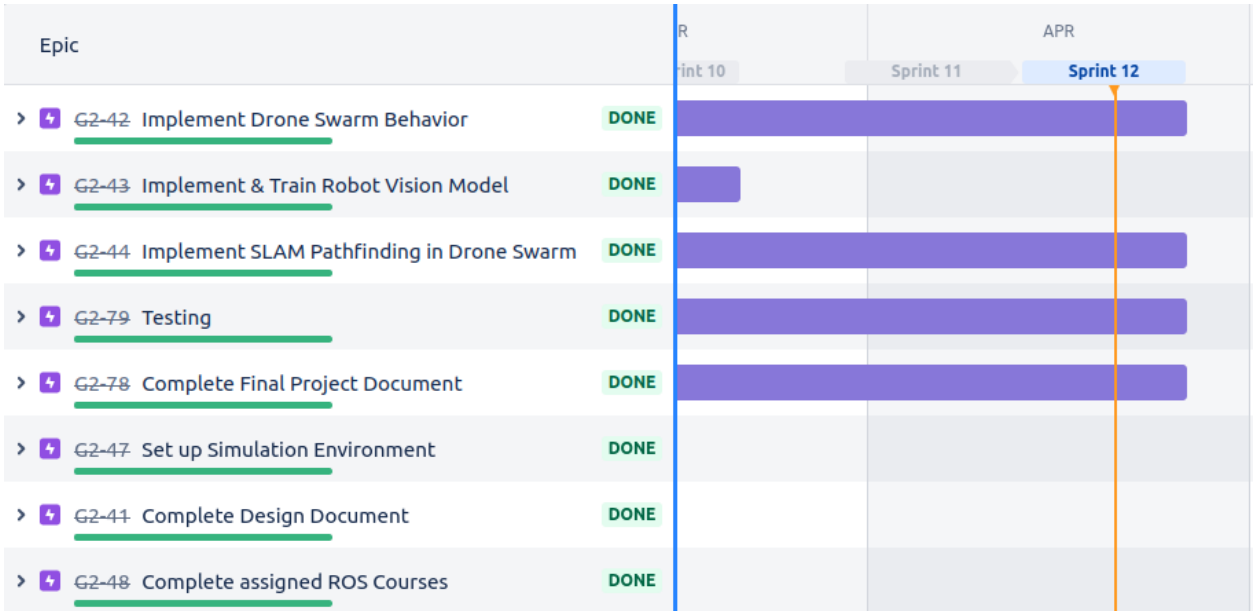
Project Timeline		
Task Description	**Estimated Completion Date**	**Status**
Complete Bootcamp Team Formation	September 16th, 2021	Complete
Kickoff Meeting with Lockheed Martin Sponsor	September 24th, 2021	Complete
Design Document review with TA	September 28th, 2021	Complete
Document Milestone 1: Complete 15 pages.	September 30th, 2021	Complete
Weekly Sponsor Meeting	October 1st, 2021	Complete

Determine ROS version and distribution	October 7th, 2021	Complete
Weekly Sponsor Meeting	October 8th, 2021	Complete
Document Milestone 2: Complete 45 Pages	October 15th, 2021	Complete
Complete ROS Core Courses	October 18th, 2021	Complete
Team/Professor Leinecker Meetup	October 19th, 2021	Complete
Sponsor Meeting	October 22nd, 2021	Complete
Establish ROS and Gazebo Development Environment in Google Cloud VM	October 25th, 2021	Complete
Document Milestone 3: Complete 60 pages minimum	November 1st, 2021	Complete
Weekly Sponsor Meeting	October 29th, 2021	Complete
Finalize software architecture design	November 7th, 2021	Complete
Complete ROS Specialty Courses	November 12th, 2021	Complete
Technical Deliverable 1: Set up 1 Drone	November 16th, 2021	Complete
Weekly Sponsor Meeting	November 12th, 2021	Complete
Document Milestone 4: Complete 90 pages minimum	November 19th, 2021	Complete

Technical Deliverable 2: Render BB8	November 22nd, 2021	Complete
Weekly Sponsor Meeting	November 26th, 2021	Complete
Begin Data Collection for Object Detection model	November 30th, 2021	Complete
Finalize Object Detection model and ROS integration architecture	November 22nd, 2021	Complete
Document Milestone 5: Complete 120 pages minimum	November 22nd, 2021	Complete
Finalize Path Planning model and ROS integration architecture	November 25th, 2021	Complete
Document Milestone 6: Complete 150 pages minimum	November 29th, 2021	Complete
Preliminary Design Document Review w/t Sponsor	December 6th, 2021	Complete
Final Design Document Due	December 6th, 2021	Complete
Technical Deliverable 3: Integrate Pathfinding into a single drone	December 10th, 2021	Complete
Technical Deliverable 4: Achieve minimum 70% accuracy with object detection model	December 13th, 2021	Complete
Technical Deliverable 5: Integrate 5 concurrent drones into single Gazebo simulation	December 15th, 2021	Complete

Technical Deliverable 6: Integrate object detection model into ROS environment and feed camera data to OD model	December 17th, 2021	Complete
Technical Deliverable 7: Integrate Path planning architecture to drone swarm.	December 22nd, 2021	Complete
Initiate Unit Testing of Path Planning Properties	February 25th, 2022	Complete
Initiate Unit Testing of Object Detection Properties	February 25th, 2022	Complete
Initiate Unit Testing of Swarm Properties	April 17th, 2022	Complete
Verify path planning time requirements	April 17th, 2022	Complete
Verify Object Detection accuracy requirements met.	April 19th, 2022	Complete
Technical Deliverable 10: Final Product and Presentation	April 19th, 2022	Complete

9.1 Gantt Chart



Gantt Chart Diagram from 09/23/2021 - 4/19/2022

11. System Operations

11.1 System Abstract

There are two phases for system operation of the swarm drones: SLAM & Navigation. As discussed earlier, before the drone can do an autonomous reconnaissance search for the target, the drone needs to have a map drawn out before navigation.

11.2 Hardware Requirements

Aim for more CPU threads and GPU VRAM to support more drones and YOLO instances.
We strongly recommend an Nvidia GPU for CUDA support.

Minimum requirements for running this system:

- Intel i7 10th generation or higher
- 16 GB RAM
- Strongly recommend a Nvidia GPU for CUDA support.
- Nvidia GTX 1660 w/ 6GB VRAM

Best Performance Specifications:

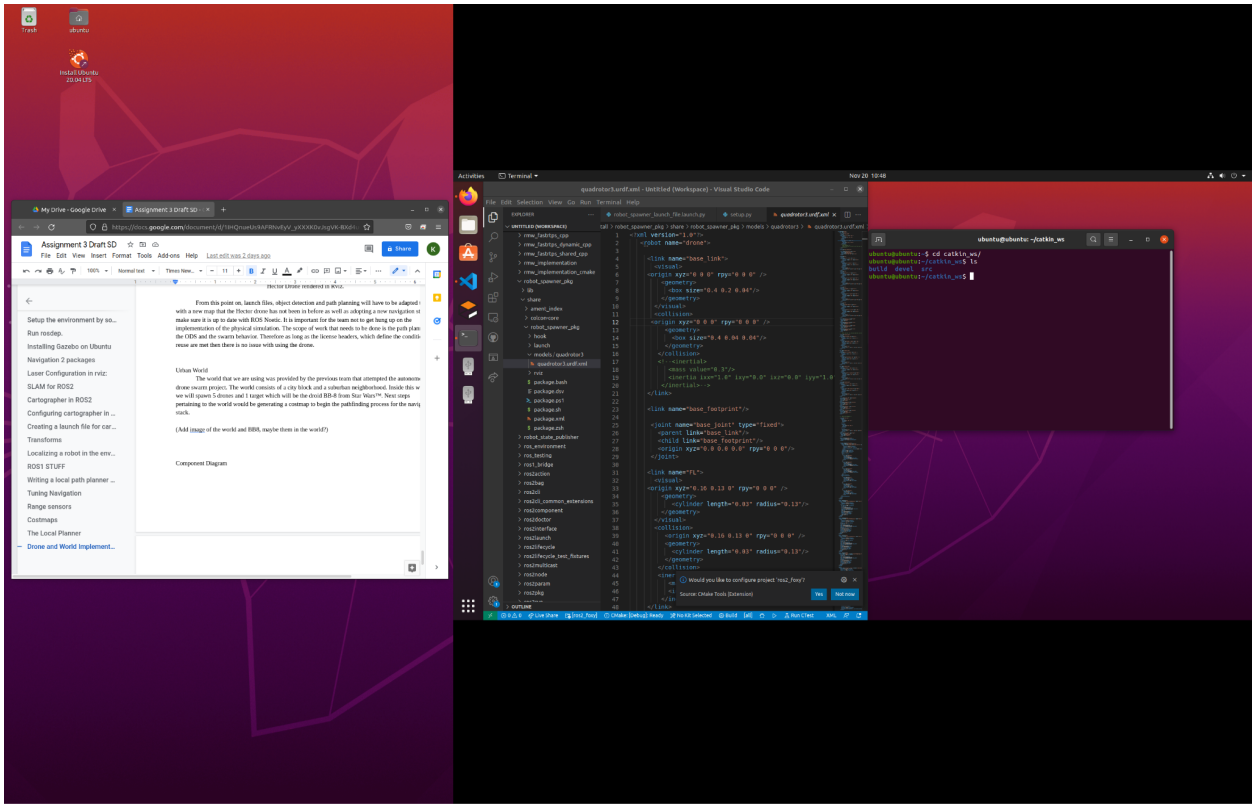
- Intel i9 10th generation or higher
- 64 GB RAM
- Nvidia RTX 3080 w/ 10GB VRAM

11.3 Remote Workstation Setup

With the requirements for ROS and Gazebo, as well for training our datasets, we need a capable machine that can run Ubuntu 20.04 LTS. We tried to run Virtual Machines, since everyone has a PC running Windows 10, and we initially thought running VM's would be a perfect candidate for everyone to remote in and to work in. We had Google Cloud running as a Cloud Host Provider, however it ended up costing more for our needs, as well the Virtual Machines lead to issues with the GPU and 3D engine for running Gazebo and training for YOLOv5.

11.3.1 Persistent Live Disk on USB

Then we decided to have two workstations, Keegan has a spare PC that runs Ubuntu all under a persistent live disk, due to simplicity, which is great for testing and setting up configuration packages. He will then have anyone to chrome remote desktop into his machine to allow any of the members to do any work that is needed. Essentially with the persistent live disk any high end system can become a workstation for the team.



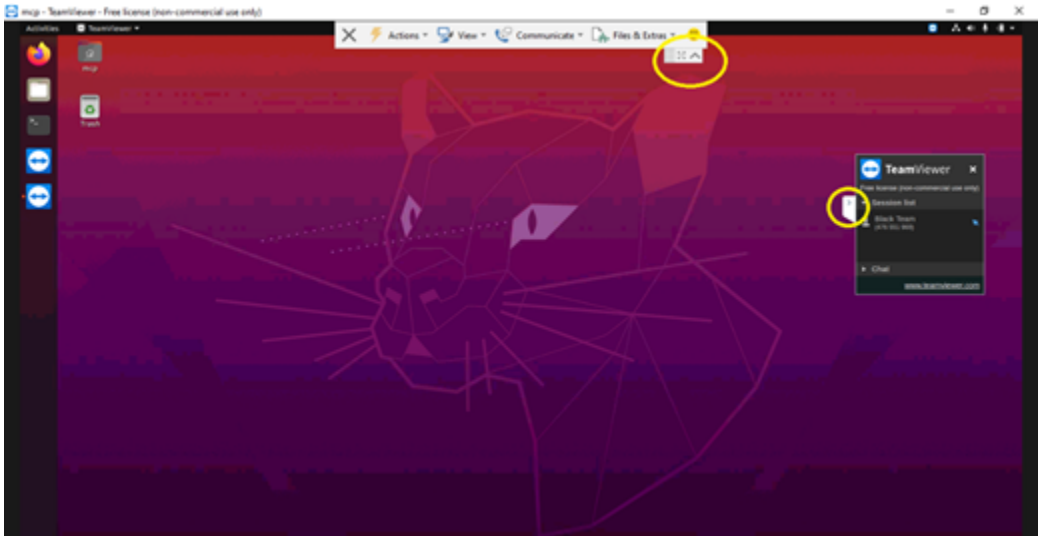
Persistent Live Disk Hardware Specifications:

- Samsung 256GB USB Flash Drive
- AMD Ryzen 5 3600 6-Core CPU
- AMD Radeon 5600 XT GPU
- 16 GB 2400mHZ DDR4 RAM
- Ubuntu LTS 20.04

I have my MSI gaming laptop, which has enough power to run Gazebo and ROS as well as be sufficient in training our datasets for object detection and classification. The laptop is connected through ethernet and is mainly on most of the day unless not needed. We use a team viewer to remotely connect into my machine, since the VNC and RDP gave issues due to port forwarding as well as requiring access to my router to disable certain settings, which I'm not allowed in my apartment complex. Due to the situation, the team viewer was the second-best thing for the team.

Both Machines have the necessary dependencies installed and configured, both running Ubuntu 20.04 LTS, Gazebo 11.0.0, and ROS 2 Foxy Fitzroy.

Team Viewer on the MSI:



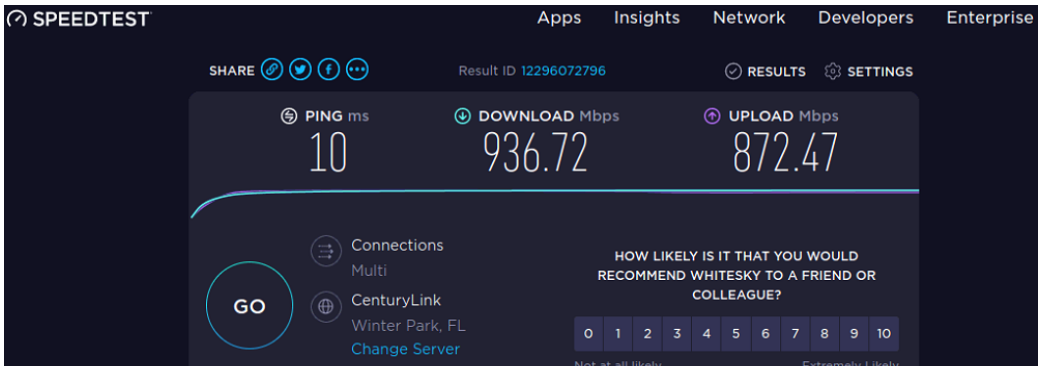
Inital Screen Prompts when User login to MSI

11.3.2 MSI Workstation Setup:

Treated as a server for the team.

- Ubuntu 20.04 LTS
- ROS 2 Foxy Fitzroy
- Gazebo version 11.0
- Teamviewer - Free Version
    - The Software package to allow the team to be able to remote into MSI Workstation
    - VNC (Virtual Network Computing) & RDP (Remote Desktop Protocol) gave issues when configuring the router.
    - Has to port forward to the router, however we have no access privileges to configure the router due to the apartment complex rules.
    - Which leads the team to resort to Chrome Remote Desktop & Team Viewer
- Disable configurations in the power & savings settings
    - Allow the PC to be operational while the lid of the laptop is closed
- When Power on, Team Viewer opens on startup, which allows anyone to reboot the workstation without headache of accessing the station
- To save on the powerbill, the workstation is powered off at the end of the day, unless a user is working on a task later the night, but whoever is the last user, will shutdown the station once finished. It will then have to be powered on at the start of each day

- Ethernet Cable is the network access for the station, allows the fastest speed and download without resorting to WiFi, as well WiFi tends to be spotty in the apartment to solely rely on.



SpeedTest Results of the MSI Remote Desktop

11.3.3 Alienware Setup

The Alienware system was provided by UCF to allow for heavier processing capabilities. Since there were 4 drones all running their own set of nodes there was a lot of overhead that allocated a majority of this system's resources.

- Intel i9 12th generation
- 64 GB RAM
- Nvidia RTX 3080 w/ 10GB VRAM

11.4 Phase 1: SLAM

Make sure you clone the swarm package into your *catkin_ws* source directory, and perform a *catkin build* and *source devel/setup.bash*

Enter the following commands in order with a new terminal in each:
Console #1:

```
- roslaunch swarm world.launch slam:=true
```

This will allow the world and the four drones to be spawned into the simulation, and enables the Hector SLAM on all four drones.

Console #2:

```
- roslaunch bb8 main.launch
```

This spawns an instance of bb8, which is our target, in the middle of the world.

Console #3:

```
- cd bash_script/
- ./takeoff.sh
```

Two bash scripts: Takeoff has all 4 drones lifted off from 1 meter off the ground with a fixed height. Reset Scan resets the hector slam when initially launched.

Console #4:

```
- cd bash_script/
- ./reset_scan.sh
```

Console #5:

```
- rosrun swarm yolo_slam.py drone1 drone2 drone3
  drone4
```

This will pipe all the drone's camera feeds to the YOLOv5 model and make inferences in real time for all 4 drones. These commands will open a new window for each drone showing their live feed and any bounding boxes greater than 70% confidence. The code also stops the drones if it has a confidence above a predefined threshold. Yolo_multi can run any number of drones, but with more drones each feed will go slower.

Console #6:

```
- roslaunch swarm start_frontier.launch
```

This will enable move_base which applies drone mobility and costmaps for the drone to move around any obstacles, as well enable explore_frontiers, which allows greedy frontier-based exploration. The drone will continue to explore the map until there are no more frontiers that can be found.

Once each drone completes a good chunk of the map, we can save each map from each drone, and then merge each of the drone maps into a master map. This will then be used to load for phase 2: navigation.

Console #7:

```
- rosrun map_server map_saver -occ 100 -free 10 -f
  map:=/drone1/map ~/catkin_ws/src/swarm/maps/drone1_map
```

```
- rosrun map_server map_saver -occ 100 -free 10 -f
  map:=/drone2/map ~/catkin_ws/src/swarm/maps/drone2_map
```

```
- rosrun map_server map_saver -occ 100 -free 10 -f
  map:=/drone3/map ~/catkin_ws/src/swarm/maps/drone3_map
```

```
- rosrun map_server map_saver -occ 100 -free 10 -f
  map:=/drone4/map ~/catkin_ws/src/swarm/maps/drone4_map
```

We will then run

```
./DisplayImage maps/drone1_map.pgm  maps/drone2_map.pgm
result/drone_a.pgm
./DisplayImage maps/drone3_map.pgm  maps/drone4_map.pgm
result/drone_b.pgm
./DisplayImage maps/drone_a.pgm  maps/drone_b.pgm
result/master_map.pgm
```

11.5 Phase 2: Navigation

In the second phase, all four drones will be spawned into the world simulation and will perform autonomous navigation and swarm communication with each drone to search for the target - bb8

Move Base allows the drones to move around the environment and communicate with the navigation stack.

AMCL - Adaptive Monte Carlo Localization - allows the drone perform localization with the map and environment

Make sure you clone the swarm package into your *catkin_ws* source directory, and perform a *catkin build* and *source devel/setup.bash*

Enter the following commands in order with a new terminal in each:
Console #1:

```
- roslaunch swarm world.launch slam:=false
```

This will allow the world and the four drones to be spawned into the simulation, and enables the Hector SLAM on all four drones.

Console #2:

```
- roslaunch bb8 main.launch
```

This spawns an instance of bb8, which is our target, in the middle of the world.

Console #3:

```
- cd bash_script
- ./takeoff.sh
```

Two bash scripts: Takeoff has all 4 drones lifted off from 1 meter off the ground with a fixed height.

Console #4:

```
- rosrun swarm yolo_nav.py drone1 drone2 drone3 drone4
```

This will pipe all the drone's camera feeds to the YOLOv5 model and make inferences in real time for all 4 drones. These commands will open a new window for each drone showing their live feed and any bounding boxes greater than 70% confidence. The code also stops the drones if it has a confidence above a predefined threshold. Yolo_nav can run any number of drones, but with more drones each feed will go slower.

Console #5:

```
- roslaunch swarm start_move.launch
```

This will enable move_base which applies drone mobility and costmaps for the drone to move around any obstacles, as well enable amcl to perform localization with the drone. After the drone is set for navigation, the nodes will run

12. Testing

12.1 Performing Tests

All tests will be performed by the corresponding team member who was responsible for that section upon completion and will share their results along with their name in the correct table for that specific test. There are two environments where the testing will be conducted: the persistent live disk of Ubuntu 20.04 and the MSI remote workstation that was allocated before development. The specifications and other details are listed above.

12.2 Testing Scope

The main scope of testing the project will be defined by four distinct levels. The components of the system at the unit, package and the integration levels as well as the sponsor's requirements as the highest level of testing. There are ROS tools and methods we will have to use to make sure we cover the testing in a thorough way that upholds the ideals of the sponsor's company, Lockheed Martin. The most important tests are that this system is able to perform the simulation with the parameters that the sponsor has laid out.

12.3 Tools Used

- Python: Unittest
- C++: gtest
- Rostest
- catkin

12.4 Testing Strategy

ROS contains many tools that can perform unit testing for nodes, libraries, and integrations. According to the official documentation on the ROS wiki page [21], these are the official levels of testing that should be conducted in ROS.

Level 1: ROS node unit test.

This level of testing uses the built in *rostest* tool in addition to the *unittest/gtest* tools mentioned in level 1. The goal is to make sure a node starts up and is able to use

its external API connections for example would be subscribed topics, services and publishing topics that are defined in the node.

Level 2: ROS node integration/ regression test.

This level's testing uses the rostest, gtest and python unittest as well as. However the scope of the tests are to determine that multiple nodes are able to run together. Since ROS is considered a multi-threaded system there can be the same problems that happen in a multi-threaded system such as race conditions and deadlocks where nodes can get held up if commands and actions get scrambled. These types of tests are to try and spot those bugs. Any regression testing that is necessary is recommended to be executed at the lowest level possible in the system.

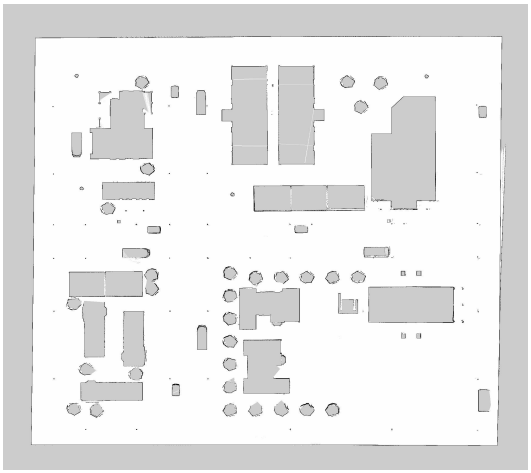
Level 3: ROS Scenario testing

The final level doesn't focus on the code as much but the actual system's ability to execute the task within the following parameters set by the sponsor. That will include testing the speed at which the swarm is able to find the target in the correct amount of time, and that object detection falls within the specifications of the requirements.

12.5 Scenario Testing

12.5.1 Maps

In the testing phase, there will be two maps in use to make sure that the time metrics are fair and comparable to the competing teams. One map will only be half of the entire environment, same as our competition and then a full map to see if the system can handle it. Map 1 will be half of the map and the dimensions are 89.4033 m x 91.1713 m. Map 2 will be the entire map with the dimensions. This way we make sure that we meet our time requirement for detecting the target.



Map 1 (Half of the map) PGM File

12.5.2 Scenario 1: Frontier Exploration

Since there are two ways to navigate in ROS, we have two scenarios that we have to test. Frontier exploration happens when the map of the environment is unknown. While using this method, the system can begin to draw a map autonomously. Each drone will start in a given corner of the map, and begin to run the necessary packages. The object detection system will run as well to try and spot the target on its exploration path. Optionally, if the drones fail to find the target and there is satisfactory coverage of the map, the individual maps can combine to create a fully rendered occupancy grid map.

The measure of success in the scenario is to make parts of a full map autonomously. The reason this is the primary goal is because that is the main point of frontier exploration is to make an unknown environment known. In addition to this, there is no ability to add swarm intelligence in this scenario since the flight paths are dictated by each drone and not a waypoint broadcaster.

12.5.3 Scenario 2: Swarm Exploration

In this scenario, the test provides a known map ideally created from the previous scenario. From the map we can generate waypoints, and dictate searchable regions of the map. Once waypoints are generated, the system attempts to divide the list among the *n* number of drones and have them search their assigned zones with minimal overlap. The motivation for this method was to reduce overall search time and maximize search coverage.

To measure the success of this scenario, drones should be able to receive their given waypoints, search the areas given. They should also be able to detect their target with a confidence at or exceeding 95% to trigger the SLAM kill switch. Note that once this kill switch is triggered, the system search timer stops. Once this switch is activated, all waypoints will be canceled and the approximate position of the drone that found BB8 will be broadcasted to the rest of the swarm. The drones will then converge on BB8 and orient their cameras to face the target and submit confidence scores to the system.

12.6 Test Plan

Level 1 Testing:

Test ID	Test Objectives	Test Description	Test Conditions	Expected Results
L1-01	Check that drone 1's API calls to all necessary topics, services are working properly.	Use rostest and gtest when appropriate to check that there are no issues with the flow of information in drone 1's nodes.	Drone 1 must be launched to begin the flow of information and rostest should be used mainly. The drone needs to be in the correct workspace as well.	API calls between nodes in the drone's package should not be throwing errors, or feeding incorrect messages.
L1-02	Check that drone 2's API calls to all necessary topics, services are working properly.	Use rostest and gtest when appropriate to check that there are no issues with the flow of information in drone 2's nodes.	Drone 2 must be launched to begin the flow of information and rostest should be used mainly. The drone needs to be in the correct workspace as well.	API calls between nodes in the drone's package should not be throwing errors, or feeding incorrect messages.
L1-03	Check that drone 3's API calls to all necessary topics, services are working properly.	Use rostest and gtest when appropriate to check that there are no issues with the flow of	Drone 3 must be launched to begin the flow of information and rostest should be used mainly. The	API calls between nodes in the drone's package should not be throwing errors, or feeding incorrect

		information in drone 3's nodes.	drone needs to be in the correct workspace as well.	messages.
L1-04	Check that drone 4's API calls to all necessary topics, services are working properly.	Use rostest and gtest when appropriate to check that there are no issues with the flow of information in drone 4's nodes.	Drone 4 must be launched to begin the flow of information and rostest should be used mainly. The drone needs to be in the correct workspace as well.	API calls between nodes in the drone's package should not be throwing errors, or feeding incorrect messages.

Level 2 Testing:

Test ID	Test Objectives	Test Description	Test Conditions	Expected Results
L2-01	Individual drone packages should be able to run without conflicting with one another.	Make sure that API calls to each other and the swarm controller don't create race conditions or deadlocks.	All drone packages must be in the correct workspace and they need to be launched.	No one drone should cause complications in another.
L2-02	Swarm controller should be able to communicate to each drone and relay information that doesn't create conflicts in the drone's operations.	Ascertain whether or not commands from the swarm controller cause issues with an individual drone's	The swarm controller needs to be in the correct workspace in order to communicate properly and create accurate	The swarm controller commands should not create any deadlocks or race conditions as it talks to each drone.

		operation.	results.	
L2-03	Ensure that all packages are able to communicate in their intended way.	Test that the entire system is correctly integrated so that drones and the swarm controller communicated as intended.	Both the swarm controller and all drones need to pass L3-01 and L3-02 to ensure that both forms of communication don't create issues.	System should be able to pass L3-01 and L3-02.

Level 3 Testing:

Test ID	**Test Objectives**	**Test Description**	**Test Conditions**	**Expected Results**
L3-01	All 4 drones are able to create a path plan and adapt if they encounter an obstacle or each other.	Run simulation to determine that the path planning setup and swarm controller are dictating flight paths correctly.	The simulation needs to be executed. Drones will need to fly the entire map without fatal collisions.	Drone's should be able to adapt and overcome complications in both static and dynamic obstacles.
L3-02	Drones have functioning cameras and camera feeds.	Run a simulation to check if each drone is able to produce a camera feed.	The simulation needs to be launched. Rviz should be running to confirm the	Each drone should have a camera feed being published to a corresponding

			existence of the feed.	ROS topic.
L3-03	A swarm behavior needs to be exhibited in the simulation.	Run the simulation and run the swarm controller to ensure that it is communicating a desired swarm behavior.	The simulation needs to be launched along with all 4s drones. The movement needs to be consistent with what was defined by the swarm controller.	Swarm controller should be able to communicate the correct flight pattern.
L3-04	The object detection system inside each drone should be able to determine that the target is in their view.	Deploy the target in a designated area and then position a drone in front of it to check if ODS recognizes the target.	The simulation needs to be launched. Target BB-8 needs to be deployed as well.	ODS should be able to recognize and identify BB-8 in the camera view.
L3-05	The object detection system should be able to identify the target with higher than 70% confidence.	Deploy the target in a designated area and then position a drone in front of it to check if ODS recognizes the target with >70% confidence.	The simulation needs to be launched. Target BB-8 needs to be deployed as well.	ODS should be able to recognize and identify BB-8 in the camera view with greater than 70% confidence.
L3-06	The object detection model shall have an	Deploy the target in a	The simulation needs to be	ODS should be able to

	intersection of union (IoU) metric of 0.5 or greater.	designated area and then position a drone in front of it to check if ODS recognizes the target with the correct IoU	launched. Target BB-8 needs to be deployed as well.	recognize and identify BB-8 in the camera view the correct IoU.
L3-07	The drone swarm shall take no longer than 3 minutes to find the BB8 target within the environment.	Deploy the target in a designated area and then run swarm flight pattern with a timer to check if the swarm can find the target in under 3 minutes.	The simulation needs to be launched. Target BB-8 needs to be deployed as well.	Swarm should be able to find the target within 3 minutes.
L3-08	Deploy the target in a designated area and then run swarm flight pattern with a timer to check if the swarm can find the target in under 3 minutes in unknown map.	Simply make sure that ROS and Gazebo are being used as tools in the simulation.	The simulation needs to be launched. Target BB-8 needs to be deployed as well.	Swarm should be able to find the target within 3 minutes.
L3-09	The software shall be implemented using some version and distribution of ROS and Gazebo	Simply make sure that ROS and Gazebo are being used as tools in the simulation.	Check the versions of ROS and Gazebo.	Simulation used ROS and Gazebo.

13.Test Results

13.1 Level 1 Results

Test ID	Test Description	Test conductor	Date tested	Results
L1-01	Check that there are no issues with the flow of information in drone 1's nodes.	Keegan Kerns, Akash Samlal, JJ Chan	4/6/2022	Pass
L1-02	Check that there are no issues with the flow of information in drone 2's nodes.	Keegan Kerns, Akash Samlal, JJ Chan	4/6/2022	Pass
L1-03	Check that there are no issues with the flow of information in drone 3's nodes.	Keegan Kerns	4/6/2022	Pass
L1-04	Check that there are no issues with the flow of information in drone 4's nodes.	Keegan Kerns	4/6/2022	Pass

13.2 Level 2 Results

Test ID	Test Description	Test conductor	Date tested	Results
L2-01	Make sure that API calls to each other and the swarm controller don't create race conditions or deadlocks.	Akash Samlal, Keegan Kerns	4/13/2022	Pass
L2-02	Ascertain whether or not commands from the swarm controller	Akash Samlal, Keegan Kerns	4/13/2022	Pass

	cause issues with an individual drone's operation.			
L2-03	Test that the entire system is correctly integrated so that drones and the swarm controller communicated as intended.	Akash Samlal, Keegan Kerns	4/13/2022	Pass

13.2 Level 3 Results

Test ID	Test Description	Test conductor	Date tested	Results
L3-01	Run simulation to determine that the path planning setup and swarm controller are dictating flight paths correctly.	JJ Chan, Caleb McCown, Akash Samlal, Keegan Kerns, Jakob Germann	4/19/2022	Pass
L3-02	Run a simulation to check if each drone is able to produce a camera feed.	JJ Chan, Caleb McCown, Akash Samlal, Keegan Kerns, Jakob Germann	4/11/2022	Pass
L3-03	Run the simulation and run the swarm controller to ensure that it is communicating a	JJ Chan, Caleb McCown, Akash Samlal, Keegan Kerns, Jakob	4/17/2022	Pass

	desired swarm behavior.	Germann		
L3-04	Deploy the target in a designated area and then position a drone in front of it to check if ODS recognizes the target.	JJ Chan, Caleb McCown, Akash Samlal, Keegan Kerns, Jakob Germann	4/11/2022	Pass
L3-05	Deploy the target in a designated area and then position a drone in front of it to check if ODS recognizes the target with >70% confidence.	JJ Chan, Caleb McCown, Akash Samlal, Keegan Kerns, Jakob Germann	4/11/2022	Pass
L3-06	Deploy the target in a designated area and then position a drone in front of it to check if ODS recognizes the target with the correct IoU	JJ Chan, Caleb McCown, Akash Samlal, Keegan Kerns, Jakob Germann	4/11/2022	Pass
L3-07	Deploy the target in a designated area and then run swarm flight pattern with a timer to check if the swarm can find the target in under 3 minutes in known map.	JJ Chan, Caleb McCown, Akash Samlal, Keegan Kerns, Jakob Germann	4/19/2022	Pass
L3-08	Deploy the target in a designated area and then run swarm flight pattern with a timer to check if the swarm can find the target in under 3 minutes in unknown map.	JJ Chan, Caleb McCown, Akash Samlal, Keegan Kerns, Jakob Germann	4/11/2022	Pass
L3-09	Ensure that ROS and	JJ Chan, Caleb	4/11/2022	Pass

	Gazebo are being used as tools in the simulation.	McCown, Akash Samlal, Keegan Kerns, Jakob Germann		

13. Stretch Goals & Advice for the Future

Below is a compiled list of stretch goals that the team was unable to complete due to time and hardware constraints. We believe that these would enhance the efficiency of the search mechanism of the swarm.

Goal Description	Justification
Utilize a Gimball camera to allow for enhanced environment search.	This would remove the need for full drone transformations during navigation, reducing errors and can reduce the time to target.
Utilize a 3d Navigation system like Octomap 3d to allow navigation paths along the z-axis.	Increased dynamic path planning and obstacle avoidance. Possible reduction in time to target metric.
Implement a moving target for the drones to continuously track & follow once found.	Provides effective swarm intelligence & flocking behavior. Would provide a more practical situation for the Boid algorithm to be fully used.

13.2 Hardware Application & Recommendations

Lockheed Martin has discussed the possibility of implementing this project using live drones. If such a project were to be conducted, the team recommends that that team use either the Ardupilot or PX4 software. Both systems are flight control software that can be used to interface with and control most autonomous vehicles including Fixed-Wing, VTOL, and Rotary UAVs, UGVs, USVs, UUVs, and small unmanned space vehicles. The major differences between Ardupilot and PX4 are outlined below:

Ardupilot	PX4
Uses a GPL license, meaning people modifying and then selling systems using Ardupilot are obligated to make their modifications open.	Uses a BSD license, meaning teams modifying it aren't obligated to publish said modifications.
Can function on various software systems, including Linux for easy ROS integration.	Current implementation does not work on Linux.
Has been in continuous development since 2009 and is more mature because of it.	Has been in development since 2012, so less functionality.
According to the development team, testing is extremely thorough and modifications are tested in live flight tests.	Tests of modifications are mostly unknown and releases have been known to have been made almost daily without much published testing.
Highly advanced algorithms allow the controller to be used in various environments.	Algorithms are much more simplistic which limits the controller to mostly indoor test flights.
Advanced algorithms make it difficult to use and implement.	Simplistic algorithms allow it to be very easily integrated and used.

It is also worth noting that Ardupilot has Gazebo simulation capability using a variety of systems and worlds for easy testing. For details on setup and usage, we recommend members reference the Intelligent Quads github repository or youtube channel. Both links are provided below:

- Github: https://github.com/Intelligent-Quads
- Youtube: https://www.youtube.com/channel/UCuZy0c-uvSJglnZfQC0-uaQ

14. References

- [1] AlexeyAB. "Repo Claims to be YOLOv5." GitHub. 2020. https://github.com/AlexeyAB/darknet/issues/5920#issuecomment-642213028
- [2] Jonathan Hui. "mAP (mean Average Precision) for Object Detection." Medium. 2018. https://jonathan-hui.medium.com/map-mean-average-precision-for-object-detection-45c121a31173

- [3] Vidushi Meel. “What is the COCO Dataset? What You Need to Know in 2021.” Viso. 2021. https://viso.ai/computer-vision/coco-dataset/
- [4] Glenn Jocher. GitHub. 2021. https://github.com/ultralytics/yolov5
- [5] “YOLOv5 Tutorial”. Google Colab. https://colab.research.google.com/github/ultralytics/yolov5/blob/master/tutorial.ipynb#scrollTo=ZY2VXXXu74w5
- [6] Glenn Jocher. “Tips for Best Training Results.” GitHub. 2021. https://github.com/ultralytics/yolov5/wiki/Tips-for-Best-Training-Results
- [7] Jojo Moolayil. “A Layman’s Guide to Deep Neural Networks”. Towards Data Science. 2019. https://towardsdatascience.com/a-laymans-guide-to-deep-neural-networks-ddcea24847fb
- [8] Gaudenz Boesch. “Deep Neural Network: The 3 Popular Types (MLP, CNN, and RNN)”. Viso. 2021. https://viso.ai/deep-learning/deep-neural-network-three-popular-types/
- [9] Michael Phi. “Illustrated Guide to Recurrent Neural Networks.” Towards Data Science. 2018. https://towardsdatascience.com/illustrated-guide-to-recurrent-neural-networks-79e5eb8049c9
- [10] Krut Patel. “Convolutional Neural Networks – A Beginner’s Guide.” Towards Data Science. 2019. https://towardsdatascience.com/convolution-neural-networks-a-beginners-guide-implementing-a-mnist-hand-written-digit-8aa60330d022
- [11] Divakar Kapil. “YOLO v1: Part 1.” Medium. 2018. https://medium.com/adventures-with-deep-learning/yolo-v1-part-1-cfb47135f81f
- [12] Glenn Jocher. GitHub. 2021. https://github.com/ultralytics/yolov5/issues/1333
- [13] Kaiming He et al. “Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition.” Papers With Code. 2014. https://paperswithcode.com/method/spatial-pyramid-pooling
- [14] Surya Gutta. “Object Detection Algorithm – YOLO v5 Architecture.” Medium. 2021. https://medium.com/analytics-vidhya/object-detection-algorithm-yolo-v5-architecture-89e0a35472ef
- [15] “Converting Between ROS Images and OpenCV Images (Python).” ROS Wiki. http://wiki.ros.org/cv_bridge/Tutorials/ConvertingBetweenROSImagesAndOpenCVImagesPython
- [16]“Learn Robotics from Zero - Robotics & Ros Online Courses.” Learn Robotics from Zero - Robotics & ROS Online Courses, https://app.theconstructsim.com/.

- [17]”How to load a world file into Gazebo-ROS2.” Addison Sears-Collins. September 15, 2021. https://automaticaddison.com/how-to-load-a-world-file-into-gazebo-ros-2/.
- [18]”Spawn model with ROS2 gazebo” ROS answers forum. https://answers.ros.org/question/314607/spawn-model-with-ros2-gazebo/
- [19]”navigation/ Tutorials.” ROS Wiki Page. 2019. http://wiki.ros.org/navigation/Tutorials.
- [20]”Python Unit Testing” ROS Wiki Page. 2018. http://wiki.ros.org/unittest.
- [21]”Automatic Testing with ROS.” ROS wiki Page. 2019. http://wiki.ros.org/Quality/Tutorials/UnitTesting.
- [22]”Dijkstra’s shortest path algorithm | greedy algo 7” 2021. https://www.geeksforgeeks.org/dijkstras-shortest-path-algorithm-greedy-algo-7.
- [23]”Graphs in python: Dijkstra’s Algorithm.” Mila Lukic. 2021. https://stackabuse.com/dijkstras-algorithm-in-python/.
- [24]”dwa_local_planner.” ROS Wiki Page, 2020. http://wiki.ros.org/dwa_local_planner.
- [25]”costmap_2d.” ROS Wiki Page, 2018. http://wiki.ros.org/costmap_2d.
- [26] “amcl.” ROS Wiki Page, 2020. http://wiki.ros.org/amcl.
- [27]”Introduction to tf2.” ROS Wiki Page, 2017. http://wiki.ros.org/tf2/Tutorials/Introduction%20to%20tf2#Using_view_frames.
- [28] “Cartographer ROS: Algorithm walkthrough for tuning.” Cartographer ROS HTML page. 2021. https://google-cartographer-ros.readthedocs.io/en/latest/algo_walkthrough.html.
- [29]”rqt graph – Visualize and Debug Your ROS Graph.” Robotics Backend, 2021. https://roboticsbackend.com/rqt-graph-visualize-and-debug-your-ros-graph/.
- [30]”Unit Testing Framework.” Python documentation. 2021. https://docs.python.org/3/library/unittest.html.
- [31] ”gmapping.” ROS Wiki Page, 2019. http://wiki.ros.org/gmapping.
- [32] ”Ubuntu Install of ROS Noetic.” ROS Wiki Page, 2021. http://wiki.ros.org/noetic/Installation/Ubuntu.
- [33] ”Spawning Multiple Robots.” Autonomous Robotics Lab Notebook, Belle Scott, 2021. https://campus-rover.gitbook.io/lab-notebook/faq/spawn_multiple_robots.
- [34] ”Hector Quadrotor” Technische Universität Darmstadt ROS Packages, 2014. https://github.com/tu-darmstadt-ros-pkg/hector_quadrotor.
- [35] “YOLO-Fine: One-Stage Detector of Small Objects Under Various Backgrounds in Remote Sensing Images.” ResearchGate article, Minh-Tan Pham, 2020.

https://www.researchgate.net/figure/Overview-of-the-state-of-the-art-You-Only-Look-Once-YOLO-family-for-one-stage-object_fig1_343459192.
- [36] “Bounding box object detectors: understanding YOLO, You Look Only Once.” Personal blog, Christopher Bourez, 2017. https://christopher5106.github.io/object/detectors/2017/08/10/bounding-box-object-detectors-understanding-yolo.html.
- [37] “Introduction to YOLO Algorithm for Object Detection.” Section.io blog, Grace Karimi, 2021. https://www.section.io/engineering-education/introduction-to-yolo-algorithm-for-object-detection/.
- [38] “YOLOv5 Documentation.” YOLOv5 documentation, Glenn Jocher, 2021. https://docs.ultralytics.com/.
- [39] “OpenCV Documentation.” OpenCV documentation, Julia Bareeva, 2021. https://docs.opencv.org/3.4/da/d97/tutorial_threshold_inRange.html.
- [40] “Canny edge detector.” Canny Edge wiki, 2021. https://en.wikipedia.org/wiki/Canny_edge_detector.
- [41] “Image Processing Learning Resources.” Morphological transformations, R. Fisher, S. Perkins, A. Walker and E. Wolfart, 2003. https://homepages.inf.ed.ac.uk/rbf/HIPR2/morops.htm.
- [42] Vision Online Marketing Team. “What Is Visual Slam Technology and What Is It Used for?” *Automate*, A3 Association for Advancing Automation, 15 May 2018, https://www.automate.org/blogs/what-is-visual-slam-technology-and-what-is-it-used-for. [43] Onit. “What Is Visual Slam? What Does Visual Slam Mean?” *Dragonfly by Onit*, 14 Apr. 2021, https://dragonflycv.com/what-is-visual-slam/.
- [44] Mur-Artal, Raúl, et al. “Orb-Slam.” *Webdiis.Unizar*, Oct. 2016, https://webdiis.unizar.es/~raulmur/orbslam/.
- [45] Hosseinzadeh, Mehdi, et al. “Real-Time Monocular Object-Model Aware Sparse SLAM.” *ArXiv.org*, 6 Mar. 2019, https://arxiv.org/abs/1809.09149.
- [46] Klein, Georg, and David Murray. “Parallel Tracking and Mapping for Small AR Workspaces.” *ResearchGate*, Dec. 2007, https://www.researchgate.net/publication/4334429_Parallel_Tracking_and_Mapping_for_Small_AR_Workspaces.
- [47] Pire, Taihu, et al. “Stereo Parallel Tracking and Mapping - Webdiis.unizar.es.” *Stereo Parallel Tracking and Mapping for Robot Localization*, https://webdiis.unizar.es/~jcivera/papers/pire_etal_iros15.pdf.
- [48] Newcombe, Richard A., et al. *DTAM: Dense Tracking and Mapping in Real-Time*. Department of Computing, Imperial College London, UK, https://www.doc.ic.ac.uk/~ajd/Publications/newcombe_etal_iccv2011.pdf.
- [49] Direct Visual Slam.” *Kudan*, Kudan, 16 Sept. 2020, https://www.kudan.io/archives/508.

- [50] Engel, Jakob. “LSD-SLAM: Large-Scale Direct Monocular SLAM.” *Computer Vision Group - Visual SLAM - LSD-SLAM: Large-Scale Direct Monocular SLAM*, 8 Mar. 2016, https://vision.in.tum.de/research/vslam/lsdslam.
- [51] Engel, Jakob, et al. *Direct Sparse Odometry*. https://jakobengel.github.io/pdf/DSO.pdf.
- [52] Choset, Howie. “Robotic Motion Planning: A* and D* Search.” *CMU*, https://www.cs.cmu.edu/~motionplanning/lecture/AppH-astar-dstar_howie.pdf.
- [53] Fernando, Dorado.“Fernandodorado/A-Star-Search-Algorithm-Implemented-in-Ros: A* Is a Computer Algorithm That Is Widely Used in Pathfinding and Graph Traversal, Which Is the Process of Finding a Path between Multiple Points, Called ‘Nodes’. It Enjoys Widespread Use Due to Its Performance and Accuracy. However, in Practical Travel-Routing Systems, It Is Generally Outperformed by Algorithms Which Can Pre-Process the Graph to Attain Better Performance, Although Other Work Has Found A* to Be Superior to Other Approaches.” *GitHub*, 29 Apr. 2019, https://github.com/FernandoDorado/A-STAR-search-algorithm-implemented-in-ROS.
- [54] “DOCKER VS SINGULARITY VS SHIFTER VS UGE CONTAINER EDITION.” *Docker vs Singularity vs Shifter vs Uge Container Edition*, https://tin6150.github.io/psg/blogger_container_hpc.html.
- [56] Reynolds, Craig (1987). *Flocks, herds and schools: A distributed behavioral model*. *SIGGRAPH '87: Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques*.
- [57] M. Dorigo, V. Maniezzo, et A. Colorni, *Ant system: optimization by a colony of cooperating agents*, IEEE Transactions on Systems, Man, and Cybernetics
- [58] Mirjalili, Ali. *Learn Particle Swarm Optimization (PSO) in 20 Minutes*. YouTube, 30 Mar. 2018, https://www.youtube.com/watch?v=JhgDMAm-imI.
- [59] lawtomated. 2021. *4 things you need to know about AI: accuracy, precision, recall and F1 scores*.
- [60] Lee, K., Kim, Y. and Hong, Y., 2021. *Real-Time Swarm Search Method for Real-World Quadcopter Drones*. Kwangwoon University.
- [61] Fefferman H. N., Starks T. P., 2005. *A Modeling Approach to Swarming in Honey Bees (Apis Mellifera)*. Tufts University.
- [62] En.wikipedia.org. 2021. *Swarm intelligence - Wikipedia*. [online] Available at: <https://en.wikipedia.org/wiki/Swarm_intelligence> [Accessed 25 November 2021].
- [63] En.wikipedia.org. 2021. *Ant colony optimization algorithms - Wikipedia*. [online] Available at: <https://en.wikipedia.org/wiki/Ant_colony_optimization_algorithms> [Accessed 25 November 2021].

- [64] En.wikipedia.org. 2021. *Particle swarm optimization - Wikipedia*. [online] Available at: <https://en.wikipedia.org/wiki/Particle_swarm_optimization> [Accessed 25 November 2021].
- [65] Docs.ros.org. 2021. *Understanding ROS 2 topics — ROS 2 Documentation: Foxy documentation*. [online] Available at: <https://docs.ros.org/en/foxy/Tutorials/Topics/Understanding-ROS2-Topics.html> [Accessed 5 November 2021].
- [66] Docs.ros.org. 2021. *Understanding ROS 2 services — ROS 2 Documentation: Foxy documentation*. [online] Available at: <https://docs.ros.org/en/foxy/Tutorials/Services/Understanding-ROS2-Services.html> [Accessed 5 November 2021].
- [67] Docs.ros.org. 2021. *Understanding ROS 2 actions — ROS 2 Documentation: Foxy documentation*. [online] Available at: <https://docs.ros.org/en/foxy/Tutorials/Understanding-ROS2-Actions.html> [Accessed 1 November 2021].
- [68] “Concepts.” *Brand*, https://moveit.ros.org/documentation/concepts/.
- [69] ”explorer_lite.” Ros Wiki Page. 2022. http://wiki.ros.org/explore_lite