Anti-Poaching Drone Control

CS 4624 Multimedia, Hypertext, and Information Access

Virginia Tech, Blacksburg, VA 24061

Professor: Dr. Edward A. Fox

Client: Dr. Francesco Ferretti

May 11, 2022

Authors:

Cory Bishop

Matthew Lyman

Matthew Hudson

Table of Contents

Table of Figures

Abstract

Our client, Dr. Francesco Ferretti, is head of the SeaQL Lab in Virginia Tech's Department of Fisheries and Wildlife Management. Dr. Ferretti coordinated with the Marine Management Organization (MMO) of the UK to design and develop an autonomous drone system to monitor the waters around the Chagos Archipelago in the British Indian Ocean Territory. The drones will fly predetermined flight paths to detect boats potentially poaching sharks in those waters. Given the large territory and limited resources, the autonomous drone detection system will decrease the cost, time, and complexity of monitoring poaching around the archipelago compared to live Coast Guard and boats. The ultimate goal is to preserve the sharks and larger ecosystem in these waters. Joining with the SeaQL Lab team, we were assigned individual tasks to help us get closer to building a final, working autonomous drone system.

Our team's primary work included calibrating and recording with cameras connected to the NVIDIA Jetson Nano computing module, finishing construction on a DJI Phantom 3 drone and hexacopter, and writing a script to automatically start the 4G cellular module to provide wireless Internet connectivity to the Jetson Nano. On recommendation from Dr. Chesi on the SeaQL Lab team, we often divided up the tasks they assigned for us. Matthew Lyman researched camera calibration and writing the scripts for the 4G module startup. Matthew Lyman and Cory Bishop focused on repairing the DJI Phantom 3 drone with hopes of performing test flights before the primary hexacopter kit was finished. We followed numerous forums and videos for troubleshooting issues. The drone's problems included two propellers not spinning and the remote controller not connecting after other SeaQL Lab members had rerouted power from the drone's battery. Next, we followed video tutorials to assist Zach Wendel and Katie Gilbert on the larger drone team in completing the hexacopter kit. This build-it-yourself Tarot FY680 drone kit will be our primary drone for flight testing the poaching boat detection models with the Jetson Nano. Lastly, the client had an issue where they must manually restart the Jetson Nano's 4G cellular module for connecting to the Internet. Matthew Lyman took charge on writing a script to startup the wireless module automatically when the Jetson Nano powers on, saving our clients time during flight testing. Without the 4G cellular connection, the Jetson Nano cannot remotely relay flight commands or video from the drone to the server.

The rest of this report outlines our work through: Deliverables, Requirements, Timeline of Work, Testing, User's Manual, Developer's Manual, Lessons Learned, and Future Work recommended for after the project.

Introduction

For the overall project scope, some broad tasks include: develop a machine-learning algorithm to detect potential poaching boats; build the drone and detection computer used for flight testing; and stream video feeds and detection alerts over a cellular connection to a server monitoring dashboard.

We were tasked with fleshing out functionality of the dashboard Coast Guard rangers would use to view video feeds and detection alerts from the drone swarm, as well as direct the swarm's flight path. However, the initial dashboard design was reliant on a third-party. As we haven't received the initial dashboard prototype, we've assisted in other small tasks in testing the cameras with the Jetson Nano[8] computing module to be attached to the drones, and assisted in making two different prototype drones operational.

Our first major tasks centered around calibrating the cameras to be used with the Jetson Nano computing module to actually receive video feed to be analyzed by the poaching boat detection algorithm. Two cameras are intended to be used: one wide angle and one long range. While a few issues arose regarding camera resolution, the calibration process[6] was easy to follow. The Robotic Operating System (ROS)[7] provides the main controls and video recording capabilities used on the Jetson Nano. The ROS Wiki[4][7] provided simple steps and tutorials for connecting both cameras and calibrating them.

After camera calibration was finished, we were asked to repair the "Frankenstein" drone for flying. This "Frankenstein" drone is a store-bought DJI Phantom 3[9] intended to perform flight testing before the hexacopter kit assembly was finished. However, we abandoned this task after running into several issues with some propellers no longer spinning and the drone not completing startup. We believe this was due to some power-routing to power the Jetson Nano from the drone's battery. We switched to help complete the main prototype drone, the hexacopter.

Zach Wendel and Katie Gilbert took responsibility for constructing the hexacopter drone[10] from a do-it-yourself kit, with occasional help from Jeremy Jenrette when issues arose. We attempted to pick up work where they left off by following video tutorials[11]. However, they still finished the bulk of the work since they had been building from the beginning, such as soldering wire connections and connecting radio modules. Some difficulties also arose when parts in the kit did not meet our needs, or the video tutorials strayed from the particular setup we were building. As of May 4, 2022 the hexacopter is not fully finished.

The sections below outline the steps we took to complete our work.

Deliverables/Requirements

Deliverables

Our deliverables will be a fully calibrated camera that fully functions with ROS wirelessly and is able to also communicate properly with a kit-built drone in-flight. We looked into the “Frankenstein” Drone to see if it could be repaired for flight where we deemed it as unable to fly. We will be assisting with building the Hexacopter drone for use with test flights. We will also research alternative cameras for the drone to use as well as allowing the drone to record with two cameras at the same time. We will deliver a full tutorial on how the setup is organized and utilized under the User Manual. Finally, we will also create a 4G module startup script that will be a bash script that will check if the module was plugged in, install drivers/updates as needed, enable the 4G module, and dial into the network and allocate an IP.

Material Requirements

The materials and software required for our project to meet these deliverables are hardware capable of running the ROS environment in Ubuntu Linux, a Runcam Camera[2] for calibration and output manipulation, and a drone for test flights.

Implementation

Phase 1 - February

1. Set up ROS (Robotic Operating System) environment on personal hardware and capture video and images through a Runcam Camera with ROS.
2. Store the recorded video file into a .bag file type using ROS commands.
3. View Runcam Camera output live on a remote connection.

Phase 2 - March

1. Disassemble the “Frankenstein” Drone (DJI Phantom 3) to see why it is not functioning properly.
2. Assemble the Hexacopter Drone as a replacement for the “Frankenstein” Drone.
3. Assemble tutorial for setting up and calibrating the Runcam Cameras with ROS.

Phase 3 – April (Current Phase)

1. Research camera alternatives to achieve a better resolution for the same price as the Runcam Camera.
2. Test the Hexacopter Drone to ensure it works properly.
3. Allow for multiple cameras to be run at the same time.
4. Create a 4G startup script.

User's Manual

Frankenstein Drone

The Frankenstein drone was the initial test drone for the project. It consisted of a DJI Phantom 3 drone with a Jetson Nano module and a camera attached to it. We used this drone to initially test the cameras. Something on the drone got messed up when work was done to power the Jetson Nano module from the drone's battery. This led to the drone no longer working. Time was spent trying to troubleshoot the drone, but it was ultimately abandoned, at request from the client, so that we could assist with the assembly of the hexacopter.

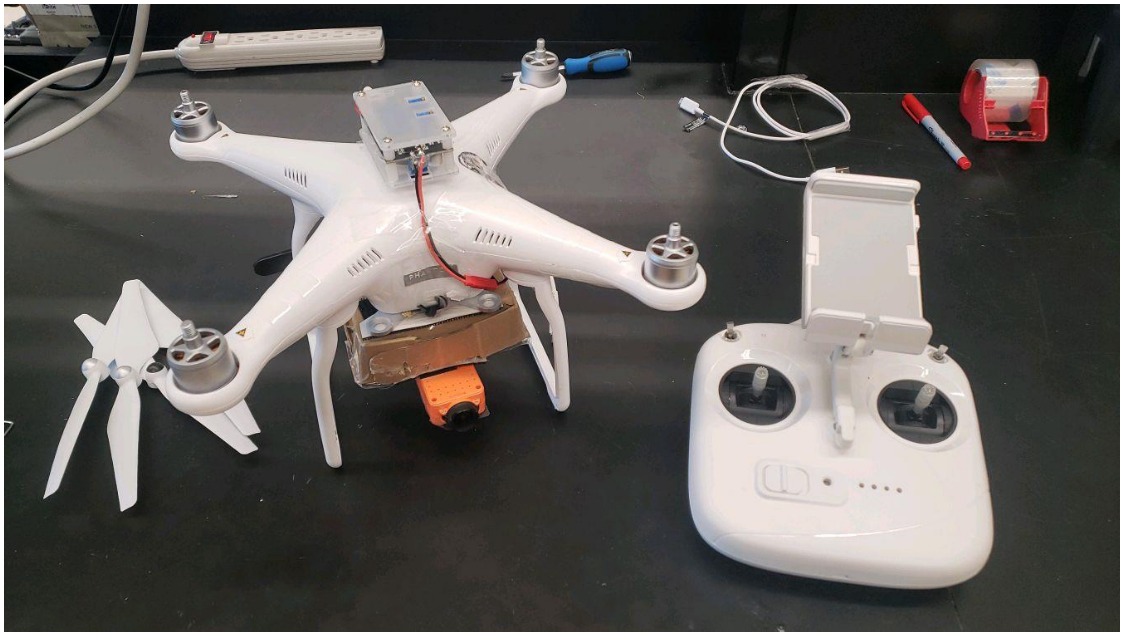


Figure 1: Frankenstein Drone

Hexacopter

The hexacopter was built from a drone kit that was purchased on Amazon[10]. The kit came with all the components needed for an autonomous drone. To complete the kit, work was needed on the assembly and wiring of the drone components. We assisted Zach Wendel and Katie Gilbert with the build by following along with a YouTube tutorial[11]. Once built, the drone looked like the figure below.

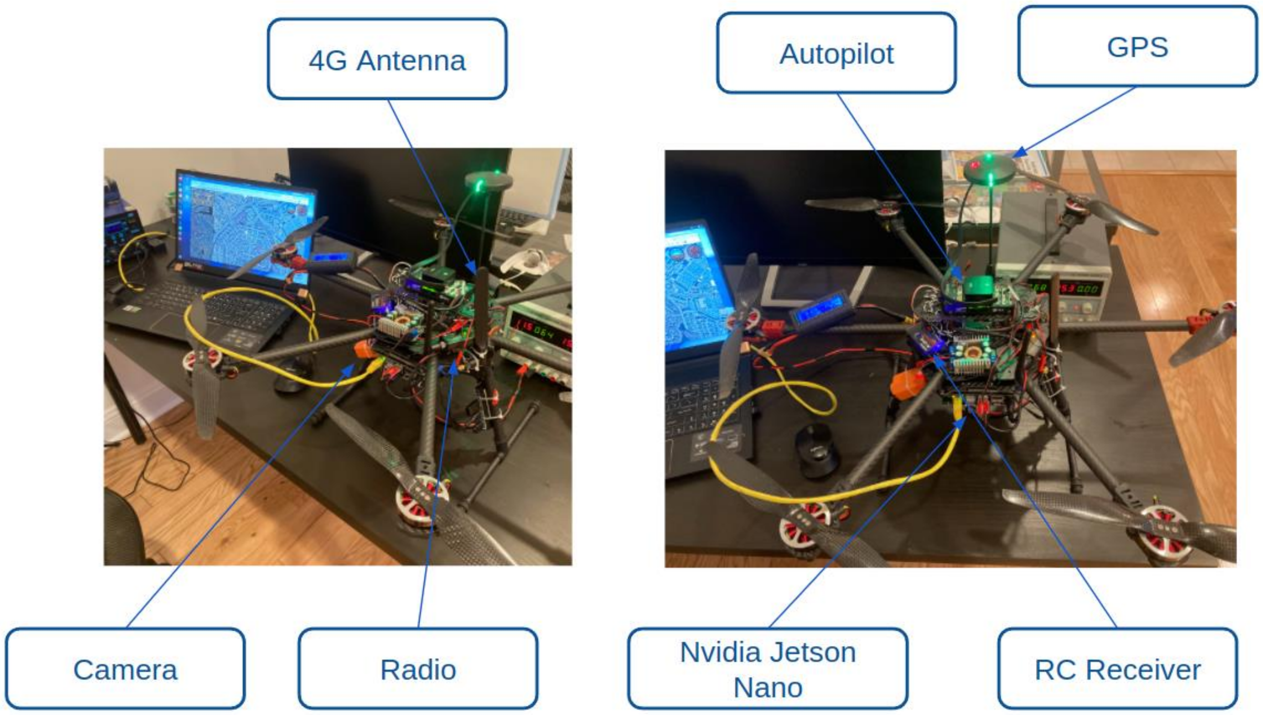


Figure 2: Hexacopter Drone Hardware

The drone consists of a 4G antenna, autopilot, GPS, camera, radio, RC receiver, and a Jetson Nano. The Jetson Nano is the computing module that ties everything together.

Camera Calibration Procedure

1. Print out the 8x6 checkerboard with 108mm squares that can be found on the calibration tutorial page in the ROS wiki[6].

2. Get the dependencies and compile the driver by running the following command:

   ```
   rosdep install camera_calibration
   ```

3. Run the camera (you can follow the steps in the developer manual to do this) and make sure it is publishing the topic. To do this run:

   ```
   rostopic list
   ```

   Check to see that the following topics are there:
   ```
   /camera/camera_info
   /camera/image_raw
   ```

4. Run the camera calibration module with the following command:
   ```
   rosrun camera_calibration cameracalibrator.py --size 8x6 --square
   0.108 image:=/camera/image_raw camera:=/camera
   ```

   This should open up the calibration window as shown in Figure 1. If it does not, try adding `--no-service-check` to the end of the previous command.

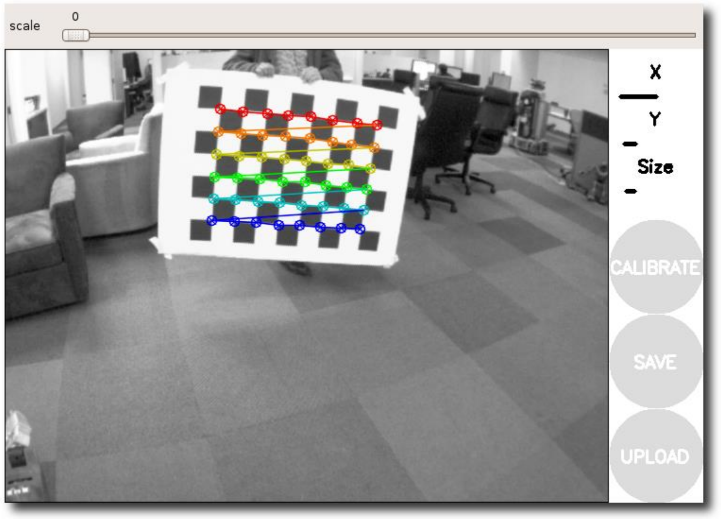


Figure 3: Camera calibration window

5. Move the checkerboard around in the frame so that it is in the left and right field of view, top and bottom field of view, and move it closer and further from the camera. An example of this is in Figure 2.

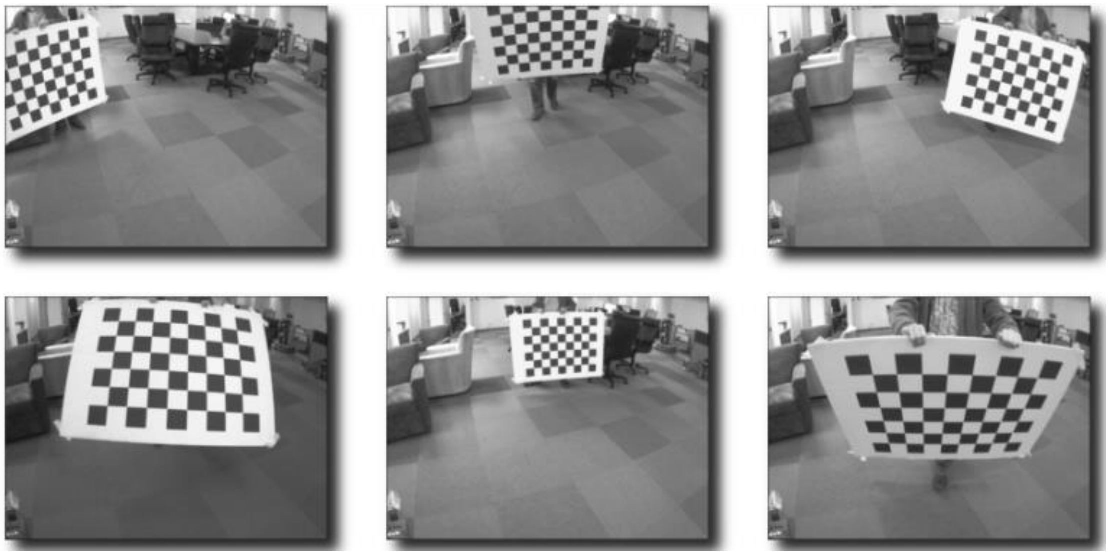


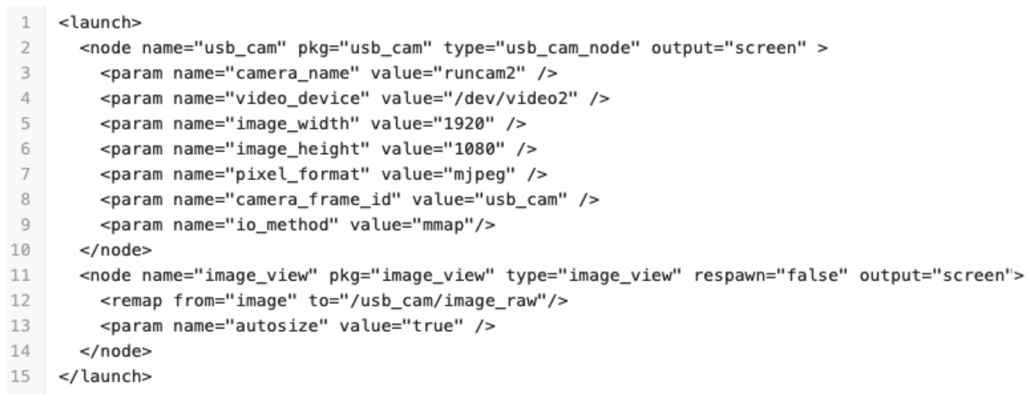
Figure 4: Example of moving the checkerboard around the camera field of view.

6. Keep moving the checkerboard around until the "Calibrate" button in the calibration window fills in and becomes clickable. This means enough data points have been collected for calibration. Click the calibrate button and wait for the calibration to complete.

7. Once complete, hit the "Upload" button and the yml calibration file will be stored for that camera. If you would also like to save the file, hit the "Save" button. You have now successfully calibrated the camera for ROS.

Modifying Camera Launch Files

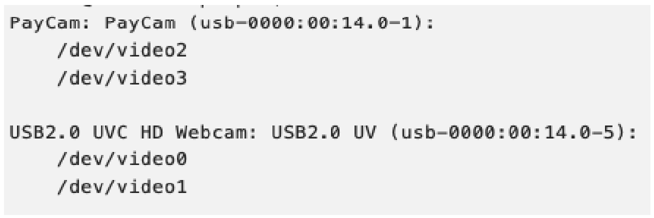
The usb_cam module in ROS comes with a test launch file but if you want to be able to use multiple cameras or change the resolution of the image you are going to need to modify the launch file. An example of a launch file is in Figure 5.



```
 1  <launch>
 2    <node name="usb_cam" pkg="usb_cam" type="usb_cam_node" output="screen" >
 3      <param name="camera_name" value="runcam2" />
 4      <param name="video_device" value="/dev/video2" />
 5      <param name="image_width" value="1920" />
 6      <param name="image_height" value="1080" />
 7      <param name="pixel_format" value="mjpeg" />
 8      <param name="camera_frame_id" value="usb_cam" />
 9      <param name="io_method" value="mmap"/>
10    </node>
11    <node name="image_view" pkg="image_view" type="image_view" respawn="false" output="screen">
12      <remap from="image" to="/usb_cam/image_raw"/>
13      <param name="autosize" value="true" />
14    </node>
15  </launch>
```

Figure 5: Example usb_cam launch file

The first important line of code is line 3. Line 3 sets the camera name for the camera. This is important because the calibration file is saved for each camera by its name, so for the calibration file to be successfully loaded, the camera name needs to be correct. To change this just change what is set as value. The next important line is line 4. Line 4 changes the device usb_cam will use for video. By default, the first plugged in camera will be /dev/video0 and the second will be /dev/video2 and so on. You can list your cameras by running “v4l2-ctl --list-devices” if you have v4l-utils installed. If you don’t have it installed run “sudo apt-get install v4l-utils” to install the package. An example output is given in Figure 6.

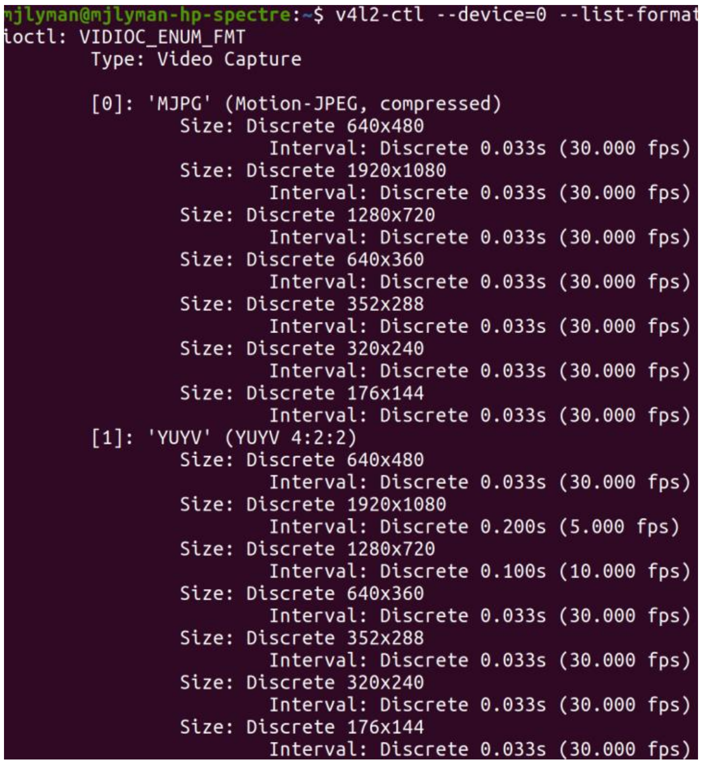


```
PayCam: PayCam (usb-0000:00:14.0-1):
	/dev/video2
	/dev/video3

USB2.0 UVC HD Webcam: USB2.0 UV (usb-0000:00:14.0-5):
	/dev/video0
	/dev/video1
```

Figure 6: Example v4l-utils output

This example shows two cameras plugged into this computer. To get video from the PayCam device you would set /dev/video2 as the video device in the launch file. To get video from the UVC HD Webcam you would set /dev/video0 as the video device in the launch file. Lines 5 and 6 allow you to change the resolution of the camera. This must be set as something that the camera supports. To see what pixel formats and resolution your camera supports you can run “v4l2-ctl --device=0 --list-formats-ext” and change the device number to whatever number is after video. An example of this is given in Figure 7.



```
mjlyman@mjlyman-hp-spectre:~$ v4l2-ctl --device=0 --list-format
ioctl: VIDIOC_ENUM_FMT
        Type: Video Capture

        [0]: 'MJPG' (Motion-JPEG, compressed)
                Size: Discrete 640x480
                        Interval: Discrete 0.033s (30.000 fps)
                Size: Discrete 1920x1080
                        Interval: Discrete 0.033s (30.000 fps)
                Size: Discrete 1280x720
                        Interval: Discrete 0.033s (30.000 fps)
                Size: Discrete 640x360
                        Interval: Discrete 0.033s (30.000 fps)
                Size: Discrete 352x288
                        Interval: Discrete 0.033s (30.000 fps)
                Size: Discrete 320x240
                        Interval: Discrete 0.033s (30.000 fps)
                Size: Discrete 176x144
                        Interval: Discrete 0.033s (30.000 fps)
        [1]: 'YUYV' (YUYV 4:2:2)
                Size: Discrete 640x480
                        Interval: Discrete 0.033s (30.000 fps)
                Size: Discrete 1920x1080
                        Interval: Discrete 0.200s (5.000 fps)
                Size: Discrete 1280x720
                        Interval: Discrete 0.100s (10.000 fps)
                Size: Discrete 640x360
                        Interval: Discrete 0.033s (30.000 fps)
                Size: Discrete 352x288
                        Interval: Discrete 0.033s (30.000 fps)
                Size: Discrete 320x240
                        Interval: Discrete 0.033s (30.000 fps)
                Size: Discrete 176x144
                        Interval: Discrete 0.033s (30.000 fps)
```

Figure 7: Example "v4l2-ctl --device=0 --list-formats-ext" command output

If you look at the output you can see the video device supports both the MJPG and YUYV pixel format. For each of these formats it supports different camera resolutions. Those are the basics of modifying launch files.

Developer's Manual

For the current phases of our project, we are still working to develop the full product. The main user experience would come from interacting with the drones through a dashboard on a server. As such, our current work serves more to guide continued development than users.

Time Spent on Major Tasks

Below is a table representing the estimated hours spent on the major tasks of the project. The estimation is including hours per person; for example, if two people worked on a task for 4 hours together, the total would be 8 hours and not just 4.

Task	**Details**	**Estimated Hours**
Frankenstein Drone	Familiarization, Initial testing, troubleshooting	40
Hexacopter	Helping with build, flight test	30
Cameras	Learning usb_cam, launch files, camera calibration, recording, new camera research	50
Startup Script	Shell script research, coding, testing	8

Inventory

We keep all our files in a shared Google drive which the client has access to. The Google drive contains our presentations, report, and any other docs or presentations for the CS 4624 class. The file structure for our Google drive is as follows:

/Presentations	This folder contains our presentations for the class.
/Final Report	This folder contains the work for our final report.
/Miscellaneous Work	This folder contains other work such as writeups for new camera recommendations.

Our other files are kept in a GitLab repository which is hosted by our client. The entire VT Drone team uses this repository, but we had our own branch where we have stored our documentation for what we have done, photos, and calibration files for the cameras. For protection of the team's work, we will not be sharing the link to the repository. The file structure for the repository is as follows:

/4g_startup_script	This folder holds the 4G startup script
/camera_calibration_files	This folder contains our yml camera calibration files for the Runcam and the Scopecam[13].
/camera_calibration_photos	This folder contains photos of us calibrating the cameras that we used in our presentations.
/camera_launch_files	This folder contains the launch files for the different cameras and camera setups.
/docs	This folder contains docs on how to calibrate the cameras, operate the cameras, and connect to the remote server.
Camera_Calibration_Progress_Presentation.pptx	This is our PowerPoint presentation we sent to our client as an update of our progress on camera calibration.
README.md	This is a "readme" for the repository that explains what the repository is for and the directory structure.

Running a Single Camera

To operate either of the cameras when connected, you can run the following command:

```
roslaunch usb_cam usb_cam-test.launch
```

This command will launch the usb_cam module and stream the data of the camera of your machine to usb_cam/image_raw. See Figure 8.

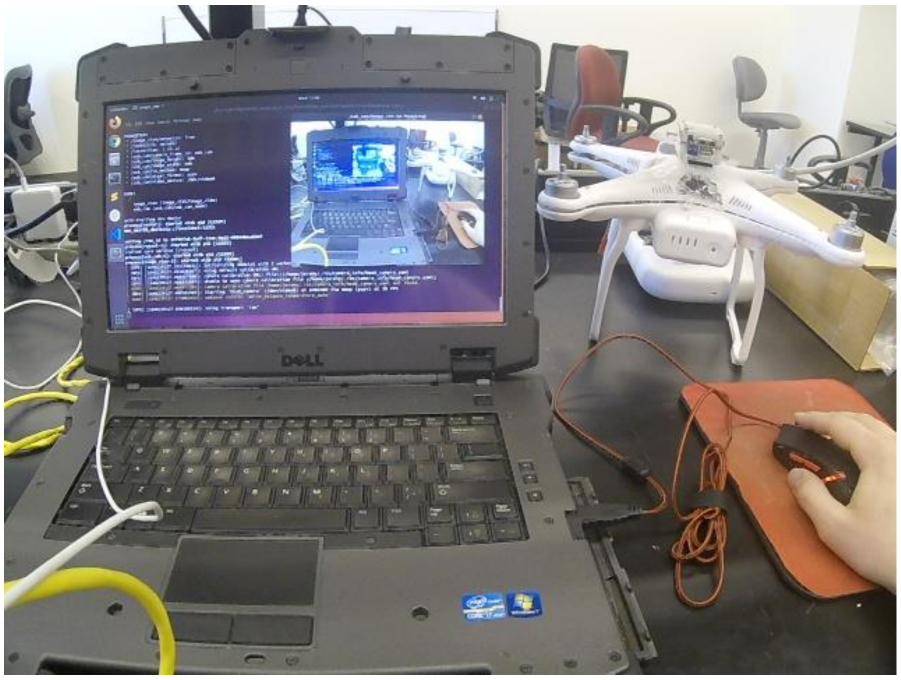
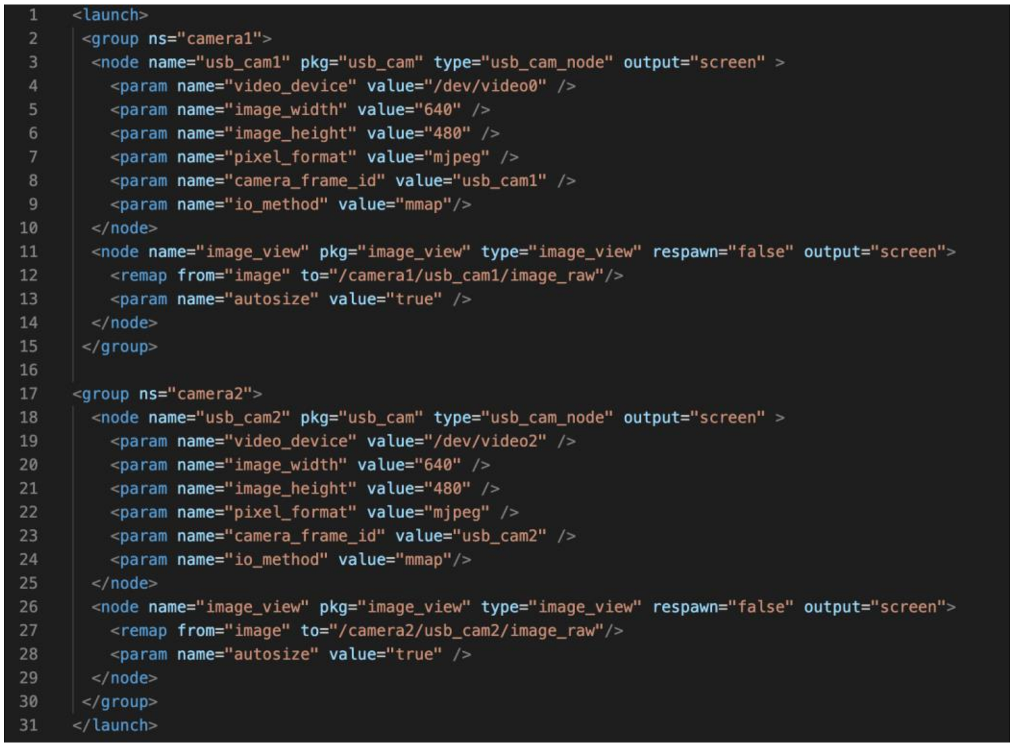


Figure 8: Running usb_cam and image_view to view camera output

Running Two Cameras

To run two cameras, we need a custom launch file. Your launch file will look something like what is shown in Figure 9.



```
<launch>
 <group ns="camera1">
  <node name="usb_cam1" pkg="usb_cam" type="usb_cam_node" output="screen" >
    <param name="video_device" value="/dev/video0" />
    <param name="image_width" value="640" />
    <param name="image_height" value="480" />
    <param name="pixel_format" value="mjpeg" />
    <param name="camera_frame_id" value="usb_cam1" />
    <param name="io_method" value="mmap"/>
  </node>
  <node name="image_view" pkg="image_view" type="image_view" respawn="false" output="screen">
    <remap from="image" to="/camera1/usb_cam1/image_raw"/>
    <param name="autosize" value="true" />
  </node>
 </group>

<group ns="camera2">
  <node name="usb_cam2" pkg="usb_cam" type="usb_cam_node" output="screen" >
    <param name="video_device" value="/dev/video2" />
    <param name="image_width" value="640" />
    <param name="image_height" value="480" />
    <param name="pixel_format" value="mjpeg" />
    <param name="camera_frame_id" value="usb_cam2" />
    <param name="io_method" value="mmap"/>
  </node>
  <node name="image_view" pkg="image_view" type="image_view" respawn="false" output="screen">
    <remap from="image" to="/camera2/usb_cam2/image_raw"/>
    <param name="autosize" value="true" />
  </node>
 </group>
</launch>
```

Figure 9: Running two cameras launch file

Looking at the launch file, we have a group for camera 1, and a group for camera 2. This launch file will first launch camera 1, which is linked to video device /dev/video0, and then will launch camera 2, which is linked to video device /dev/video2. See Page 12 for information on modifying launch files.

Recording Video from the Camera

To record video from the camera we use the following command:

```
rosbag record usb_cam/image_raw
```

This command uses the rosbag module[5] and will start recording whatever data is streamed to usb_cam/image_raw. Once you stop the recording the data is stored in a .bag file in the folder where the command was initiated.

Video Playback

To play back video from a .bag file, first make sure the usb_cam module is no longer running and streaming data to usb_cam/image_raw. Once that is done run the following commands:

```
roscore
rosrun image_view image_view image:=/usb_cam/image_raw
rosbag play <bag file>
```

The first command starts the core of ROS. The second command opens up the image_view module. It waits for data to stream to usb_cam/image_raw. The third command starts playing the .bag file. Since we recorded the data from usb_cam/image_raw, that data is streamed out and the image_view module will display it.

Camera Calibration

The cameras on the drones need to be calibrated so that real-world straight lines appear straight to the camera. This camera calibration procedure enabled the drone to improve the confidence and quality of the detected objects. The calibration of the cameras was done using a ROS module called camera_calibration[6]. To calibrate the cameras, we followed along with the Monocular Camera Calibration Tutorial on the ROS Wiki[6]. The procedure uses a printed-out checkerboard for the calibration. Once you run the module, you put the checkerboard in the frame of the camera and move it around until enough data points have been gathered and the calibration algorithm can be run. After the algorithm has run, you can save and commit the calibration files so that they can be loaded up whenever the camera is run in the future.

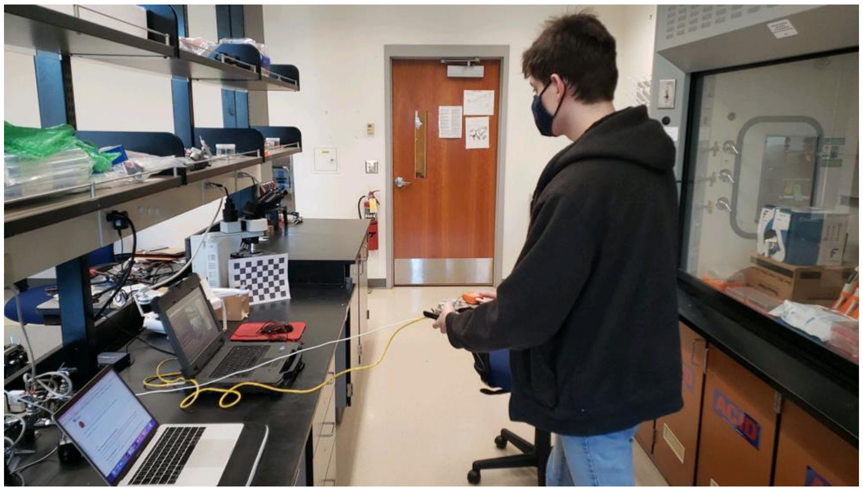


Figure 10: Calibrating the camera using the checkerboard pattern.

Connecting Remotely to the Drone

An AWS server has been set up that allows us to remotely connect to the Jetson Nano modules on the drones and run commands from anywhere in the world. This AWS server was set up by other members of the SeaQL Lab team. The Jetson Nano automatically connects to this server anytime it boots up and has an internet connection. We SSH into the AWS server (ssh -X ubuntu@<serverIP>) and then can SSH into the drone's Jetson Nano (ssh <droneName>@<droneIP>). This functionality means we can see what the drone's cameras are looking at from anywhere we have an internet connection.

4G Module Startup Script

The drones will primarily connect to the cloud server via a 4G connection.

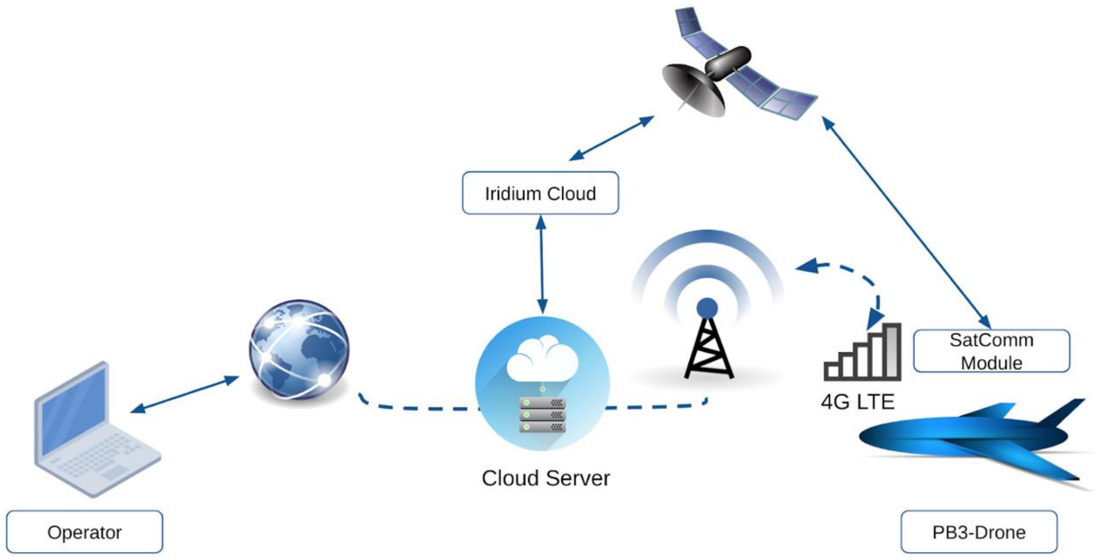


Figure 11: Network architecture

We needed to create a startup script to initialize the 4G module on startup to ensure the drone could connect to the server. The script was created using the 4G module's wiki documentation[14]. To run the script, the script needs to be added to the cron task list. To do this, run (crontab -e) and then add in the line (@reboot <path to script>). The 4G module startup script is not fully complete and tested as of writing this. The script is in the Gitlab repository.

Lessons Learned

Clarify Deadlines Ahead of Time

We joined an ongoing project with certain deadlines relevant to the larger project team's own client, the Marine Management Organization of the UK (MMO). Many of the deadlines we were aware of initially were arbitrary deadlines for our small tasks. However, we often had to rush to meet larger project deadlines for the MMO that we were unaware of initially.

For example, we were tasked at one point with repairing the "Frankenstein" DJI Phantom 3 drone to perform flight testing. However, our client had some deadlines for a progress report they wrote for the MMO. Without original knowledge of this deadline, our client asked us to abandon this drone and switch focus to helping Zach Wendel and Katie Gilbert on the SeaQL Lab team finish building the Tarot FY680 hexacopter kit. This was important to show progress to the MMO toward reaching the goal of flying the main drones the SeaQL Lab team plans to use for flight testing in the future. If our team was aware of these other deadlines ahead of time, we could have discussed shifting the focus of our team's efforts earlier to reduce our client's stress and make more progress by the deadline.

We learned when joining an ongoing project to clarify what all of the deadlines for the project are. This could have prevented rushing and re-arranging the schedule, that was required for completing different tasks.

Ask for Help

When important deadlines approach, it's better to ask for help quickly instead of always trying to troubleshoot on your own. There is virtue in troubleshooting on your own to solve problems instead of always deferring to other team members. However, when the work must be completed quickly, it's good to ask for help after searching for a solution on your own for a little bit of time. This is especially true when other team members are much more knowledgeable about the technology, limitations, and project goals than you are.

When first joining the team, Dr. Chesi tasked us with setting up a model ROS environment for eventually calibrating the cameras. Initially we started development on our personal laptops without help. However, this process was slow going. After mentioning some issues to Jeremy Jenrette on the SeaQL Lab team, he quickly recommended we use the ROS environment he'd already setup on the Jetson Nano. Other tasks originally assigned to us, such as remotely accessing the Jetson Nano and viewing the cameras, he already knew how to do. More than that, he showed us how to perform these tasks in a matter of minutes. Online resources can be valuable, but your team members can often provide the best, quickest help.

Abandon Failed Implementations Quickly

Likewise, a project with quick deadlines and lots of problems requires moving past the problems as fast as possible. While troubleshooting is necessary, too much time spent on one task could set back the entire project. Thus, if one proposed implementation step doesn’t work as intended, quickly proposing a new solution can be critical to staying on track.

After completing the camera calibration process, we were tasked with repairing the DJI Phantom 3 “Frankenstein” drone. This off-the-shelf drone was originally intended to connect to the Jetson Nano to perform flight testing of the poaching boat detection algorithms (not yet developed). This planning would allow flight tests to occur while the future testing drones were being built, from the Tarot FY680 build-it-yourself kit. Some work had been done previously on the Frankenstein before we took over, such as rerouting power from the drone’s battery to a power adapter. This power adapter would power the Jetson Nano attached to the drone. However, this method seems to have brought other issues. Two of the drone’s propellers would not spin and the remote controller would not connect with the drone. We were tasked with fixing these issues to perform flight tests.

Despite two weeks of troubleshooting, we were unable to progress toward repairing the drone. Approaching deadlines for the SeaQL Lab team led us to switch focus to help other members complete building of the Tarot FY680 kit instead of repairing the DJI Phantom 3. This was a hard but important decision for our client. The Frankenstein drone’s flight tests were meant to save the team time in the long run. This would mean they could test the boat detection algorithms before the final testing hexacopter drones were finished. When the Frankenstein drone proved too difficult to repair, the issues caused the opposite effect, taking up more of the team’s time than what it was worth. This prospect increased as the hexacopter drone moved nearer to completion.

In situations like these, we learned despite the value of persevering in the face of problems, knowing when to abandon one aspect of your plan and pivot to find another solution is also just as valuable. The end goal is the most important aspect of any project; the particular details in between are often interchangeable. As such, you shouldn’t be tied to any particular implementation choice if it costs the team too much, in terms of work, time, money, etc.

Plan Work to Progress During Roadblocks

At many points during project work, we ran into roadblocks preventing us from progressing on certain tasks.

Several issues caused this, such as:

- Discovering we didn't have the right parts.
- Parts breaking.
- Waiting for parts to arrive.
- Tasks dependent on work of other team members.
- Troubleshooting issues.

As such, it's good to have a plan of what other tasks can be worked on next when the most urgent ones come to a stand-still. Even if other tasks cannot progress, there's potential to research how future steps should be implemented once the team is able to.

This also could have been vital to the overall schedule of the project. Before we joined the larger team, their work was delayed by five months, dramatically increasing the speed needed to finish some milestones.

Learn How to Learn and Troubleshoot on Your Own

Many problems have come up while developing and building the drones and Jetson Nano computing system. Initially, our group members were unfamiliar with much of the technology needed for development. But nonetheless, our project clients had confidence in our abilities to complete the work, even as our initial work scope changed.

As with any project, learning from all the available resources[7][11] to understand how the entire scope of the system works, and how to troubleshoot issues, is important. Few projects are undertaken where the designer knows all of the required information ahead of time. And problems are always expected. Skills with problem solving and project management are especially crucial when different team members take responsibility for their own individual tasks. Yet each team member's tasks often define the schedule for the tasks of others on the team member's tasks.

Clarify Intentions for Project Tasks

Since we joined the larger project team after work had already begun, many design decisions had been made without our knowledge. In general, this didn't directly affect our work, as we learned how to implement the tasks we were given regardless of the reasoning for them.

However, we realized after becoming stuck on and abandoning one aspect of the project, it might’ve been helpful to provide our own input into the design decisions. This could have saved us some time, trouble, and money.

Initially, we were developing two drones at once: a “Frankenstein” drone that was store-bought and a hexacopter being built from scratch by other team members. The hexacopter is the full development drone that will be used for further testing and production, but building from scratch has taken much longer than planned and as such, testing is delayed till Summer 2022. The Frankenstein drone was intended to be a proof-of-concept for testing the camera and detection algorithms before the full hexacopter was finished. After the hexacopter was finished, the Frankenstein drone would not be used.

In hindsight, instead of tasking us to work on a separate drone in tandem not meant for long-term use, it may have saved time to task our group with helping the other members to finish the hexacopter more quickly. This is what happened anyway, after the Frankenstein drone posed problems we weren’t able to resolve quickly.

One other prior design choice was to route power from the drone’s power system to run the Jetson Nano computing module. While a good idea in theory, these modifications are suspected to have caused many of the problems preventing us from flying the Frankenstein drone. This was a hard task for a team not experienced with electronics. We also realized a simple solution of using a portable power bank would have worked well, decreasing both complexity and risk of failure for powering the computer.

Future Work

While our team ends our work on this project, the larger SeaQL Lab continues development on this project. After some final troubleshooting with the hexacopter drone, the team will be able to perform test flights. These initial flights will test the functionality of controlling the drone remotely using a computer connected to a server. This server will send commands through the Internet to the Jetson Nano onboard the drone. These commands in turn determine the drone's flight path.

Once this testing is done, a poaching boat detection algorithm can be installed on the Jetson Nano. This algorithm will use camera footage to detect boats on the water with potential for poaching hazard. The Jetson Nano will use cameras such as those our team calibrated for capturing this video footage. In the interim, flight testing of the algorithm can be tested using small boats in lakes or cars on the road. Testing the streaming of this video footage and detection alerts also remain for future work. Finally, the team can secure the Jetson Nano to longer-range drones for testing the algorithm over open water similar to the final intended environment.

Acknowledgements

The Drone Team

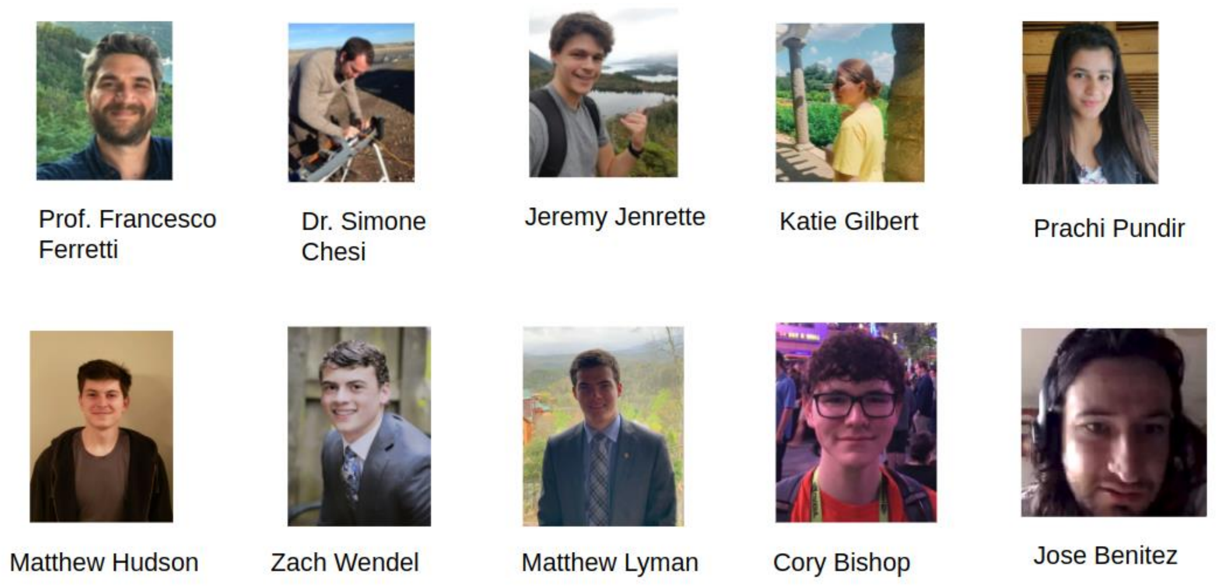


Figure 12: VT Drone team members

Marine Management Organization of the UK[1]

SeaQL Lab[2]

References

[1]Marine Management Organisation, "Marine Management Organisation," GOV.UK, 2022. [Online]. Available: https://www.gov.uk/government/organisations/marine-management-organisation. [Accessed: Apr. 28, 2022]

[2]SeaQL Lab, "SeaQL Lab Homepage," 2022. [Online]. Available: http://35.245.242.176/seaql/. [Accessed: Apr. 28, 2022]

[3]ROS Wiki, "usb_cam Package Summary," ROS Wiki, Nov. 19, 2019. [Online]. Available: https://wiki.ros.org/usb_cam. [Accessed: Apr. 28, 2022]

[4]ROS Wiki, "ROS/Tutorials," ROS Wiki, Jun. 03, 2021. [Online]. Available: https://wiki.ros.org/ROS/Tutorials. [Accessed: Apr. 28, 2022]

[5]ROS Wiki, "Recording and playing back data," ROS Wiki, Jun. 23, 2020. [Online]. Available: https://wiki.ros.org/rosbag/Tutorials/Recording%20and%20playing%20back%20data. [Accessed: Apr. 28, 2022]

[6]ROS Wiki, "How to Calibrate a Monocular Camera," ROS Wiki, Sep. 19, 2019. [Online]. Available: https://wiki.ros.org/camera_calibration/Tutorials/MonocularCalibration. [Accessed: Apr. 28, 2022]

[7]Open Robotics, "ROS: Home," 2021. [Online]. Available: https://www.ros.org/. [Accessed: Apr. 28, 2022]

[8]NVIDIA, "NVIDIA Jetson Nano For Edge AI Applications and Education," NVIDIA, 2022. [Online]. Available: https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-nano/. [Accessed: May 04, 2022]

[9] DJI, "Phantom 3 Standard - DJI," DJI Official, 2022. [Online]. Available: https://www.dji.com/phantom-3-standard. [Accessed: May 04, 2022]

[10] TAROT, "Amazon.com: TAROT FY680 3k Carbon Fiber Full Folding Hexacopter 680mm FPV Aircraft UFO Frame 6-Axis DIY Drone Airframe Kit TL68B01 : Toys & Games," Amazon, 2022. [Online]. Available: https://www.amazon.com/Carbon-Folding-Hexacopter-Aircraft-Copter/dp/B00O1WIYCY. [Accessed: May 04, 2022]

[11]R. Roux, Hexacopter Build Part 1 - Tarot FY690S Frame and its Parts, vol. 1, 12 vols. (Nov. 12, 2015) [Online]. Available: https://www.youtube.com/watch?v=hQ7XcUWutoE. [Accessed: May 04, 2022]

[12]RunCam, "RunCam 2," RunCam Store, 2022. [Online]. Available: https://shop.runcam.com/runcam2/. [Accessed: May 04, 2022]

[13]RunCam, "RunCam Scope Cam Lite," RunCam Store, 2022. [Online]. Available: https://shop.runcam.com/runcam-scope-cam-lite/. [Accessed: May 04, 2022]

[14]Waveshare, “SIM7600CE-T/E-H/A-H/G-H 4G Modules - Waveshare Wiki,” 2022. [Online]. Available: https://www.waveshare.com/wiki/SIM7600G-H_4G_for_Jetson_Nano. [Accessed: May 04, 2022]