Final Report
A12/3 - Pop A Red Balloon
(90 roter Luftballons)
Spring 2022

Eli Hubble
Gabriel Einstoss
Lauren Madey
William Jarratt

Github Link: Pop A Red Balloon

This document describes project A12 - Pop A Red Balloon. The goal was to autonomously pop a red balloon using a drone. This project was an exercise in object detection and its integration with anonymous systems. The team designed a system that consists of an onboard embedded computer (OEC) and video feed that works in concert with the drone's autopilot system to locate the red balloon in a field and then navigate to the balloon to pop it. The goal was achieved with the built system and this document describes the design and operation of the system built by the team.

Introduction

The project's objective was to pop a red balloon that was located at a random location in a 50 meter by 50 meter square located at the NCSU Animal Health Building (Google Map) using an autonomous drone. The balloon was placed a known distance, approximately 2 meters, above the ground. The problem that this project seeks to tackle is that of object detection based tasks, or having a system complete a task on its own that revolves around detecting real life objects, such as a red balloon.

The nature of this problem is important because in many instances an autonomous system must react with the natural world where there is no direct way to translate real properties into computer understandable data. What sensor can directly sense a red balloon? This problem goes beyond a red balloon and can apply to many other problems. In a direct application, being able to recognize a red circular object could allow detection of many objects if one attaches a red circle to it. In another scenario, a model for detecting people could replace the red balloon and then the drone would navigate to that person instead. Not with the goal of popping of course. This could be helpful for search and rescue drones.

The contribution of this project is a proof of concept demonstration that can be used as a lesson in what and what not to do for object detection based tasks. We provide a basic drone based system that can be used as a baseline for object based detection and approach applications.

Related Work

Two teams attempted this project, one in spring 2017 (S17) and the other in spring 2018 (S18). Our approach follows some of the methods used in previous semesters, but overall our design was different in key ways.

Spring 2017

The S17 team used a Raspberry Pi as their OEC and were able to successfully pop the balloon using the propellers. Their initial plan was to use a laser, but could not pop the balloon due to stability and environmental issues such as the reflections on the balloon. Their video processing was done using OpenCV where the balloon was detected by using HSV filtering and shape detection. They mentioned that they had issues detecting the balloon in different lighting conditions and talked about the changes they made to their video processing. The drone started at a waypoint near the balloon to start. When the balloon was detected, an XYZ vector was generated from the OpenCV analysis and used to navigate to the balloon. In the end, the drone located the balloon, navigated to slightly above the balloon then pitched forward and ran the propellers into the balloon.

Spring 2018

The S18 team used a dragonboard 410c in place of the Raspberry Pi as their OEC. The S17 team suggested that limited processing capabilities of the Raspberry Pi hampered their progress and the S18 team figured the dragonboard 410c would rectify this issue. They also utilized a gimbal in their solution to rectify the stabilization problem. Ultimately they were not able to achieve their goal though, due to issues throughout the project related to the drone configuration itself. The proposed solution was to sweep a grid pattern and then once the balloon was detected, to generate an XYZ vector, similar to the previous team to approach the balloon. Their video processing was similar to the previous team, utilizing OpenCV.

Approach

Our approach was similar to S18 in that we chose to start our search with a grid sweep and then once the red balloon was detected, use a separate algorithm for approaching the balloon. We decided to use the Nvidia Jetson TX2 instead of the Raspberry Pi due to concerns over performance issues. In the end, the TX2 was more than sufficient for the OpenCV application we implemented. The video feed was provided by a Logitech C930 webcam connected via USB to the TX2. The red balloon detection was based on the SimpleBlobDetector class in OpenCV, which provides shape detection on HSV filtered frames. We decided to forgo the laser due to the failure of each previous team to implement the laser popping. Additionally, we decided to use a rigid popping mechanism instead of the propellers. Our thought was that we could position the popping mechanism to correlate with the video feed and not rely on the pitch of the drone. Unlike the previous teams, we decided to not generate XYZ vectors based on video processing. Instead we decided to change our altitude, position, and yaw based on the location of the red balloon in a single frame.

Operational Flow

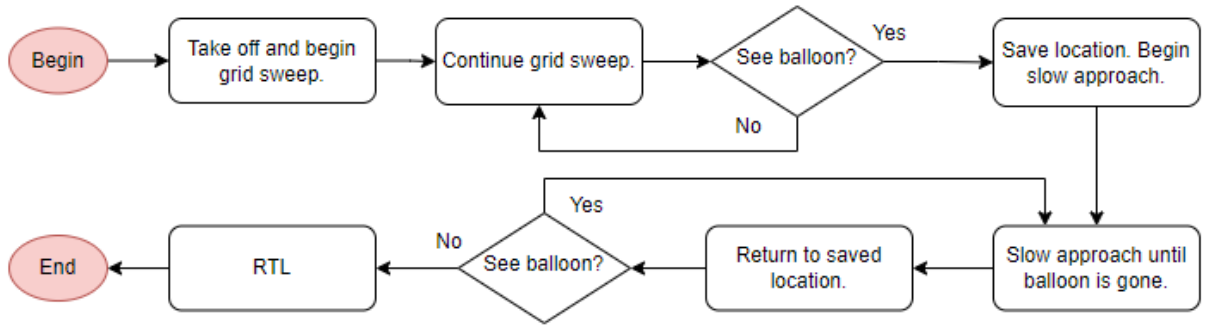

Figure 1: High Level Operational Flow

The drone can begin in any location as the grid is defined by relative GPS coordinates. The upper right corner of the 50 meter by 50 meter grid is specified by the user and is the first waypoint. The drone will move from left to right, advancing the grid by 5 meters each sweep. The drone has a fixed yaw and looks forward as it sweeps. In our demonstration, the drone yaw was north and it swept west/east, east/west. As the drone sweeps the grid, the video feed is being analyzed by the OpenCV application for a red object.

Once the red object is detected the drone will save the location that it saw the red object and move to that location. We assumed that no other red objects would be in the field.

The drone then initiates the slow-approach process. Here the (x,y) coordinates of the red object and its size in the camera feed are used as parameters for the drone to adjust its yaw, position, and altitude. The drone will adjust these three parameters to maintain the red object in the center of the frame. If the object is too far left or right, the drone will yaw to correct. If the object is too high in frame, it will move forward slightly, causing the object to drop in frame. If the object is too low, the drone will lower its altitude to raise the object in frame. The popping mechanism is positioned so that a slow-approach with the balloon centered in frame will cause the balloon to meet with the popping mechanism.

When the balloon pops, the leftover material is no longer a rough circular shape, the OpenCV application no longer detects the balloon and the drone returns to the saved location in the grid sweep where it first saw the red object. If the drone lost sight of the red object and it still exists, then it will re-attempt the slow-approach. Otherwise, if the balloon cannot be located, it must have been popped and the drone returns to launch.

Architecture

The design's architecture is shown in figure 2. The UAV was a SAM4 and was built and set up by the lab. The autopilot and power source was designed for us. We added the DC-DC downconverter, camera, gimbal, and TX2 OEC onto the UAV. The TX2 and gimbal require greater than 11V and the downconverter supplies 12 volts. The downconverter is supplied by the battery. The camera derives power from the TX2 USB port.

The TX2 receives data from the camera via USB and communicates with MAVLink commands over microUSB with the autopilot. The TX2 generated, received, and forwarded MAVLink packets. All MAVLink packets were routed using MAVProxy running in an Ubuntu terminal. The python application was responsible for sending drone commands through MAVProxy and also for receiving and analyzing video feed from the camera. Additionally MAVLink telemetry packets received from the autopilot were forwarded to the GCS.

The GCS used was QGroundControl and ran using Windows 10. The GCS was only used to monitor parameters such as position, altitude, speeds, yaw, and battery. Alongside the GCS, we also ran VNC Viewer to log in remotely to the TX2 desktop. This allowed us to start MAVProxy and the python application, and to view the video stream in real time. Unfortunately, the video stream was very laggy due to being streamed over Wi-Fi.

Further details about how the python application operates is described in other sections. To run our system we would first open three terminals. In the first terminal we would start MAVProxy, `mavproxy.py --master=/dev/ttyACM0 --out=udp:192.168.1.139:14550 --out=udp:127.0.0.1:14551`. In the next, we would run, `python2.7 main.py connect :14551`. In the third we would wait until at waypoint 0, then execute, `python3.7 videoproc.py`. The demonstration run had slightly different file names, but they match up to the most recent GitHub commit.

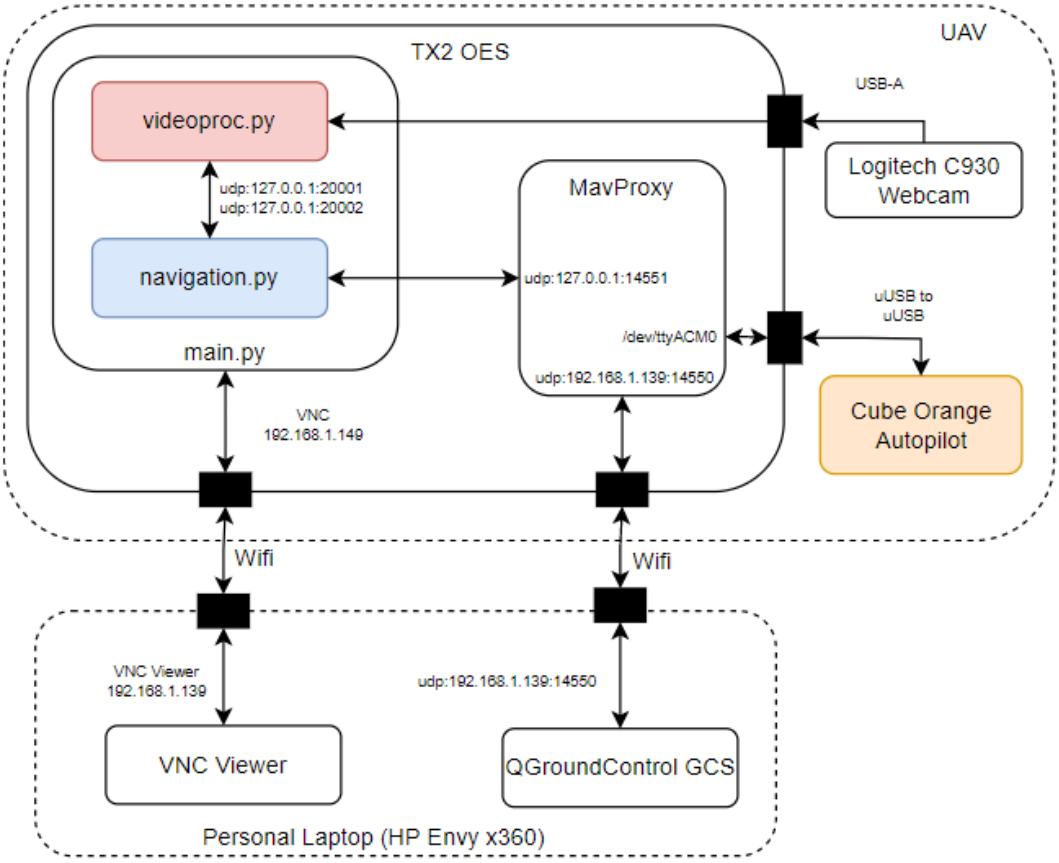

Figure 2: System architecture.

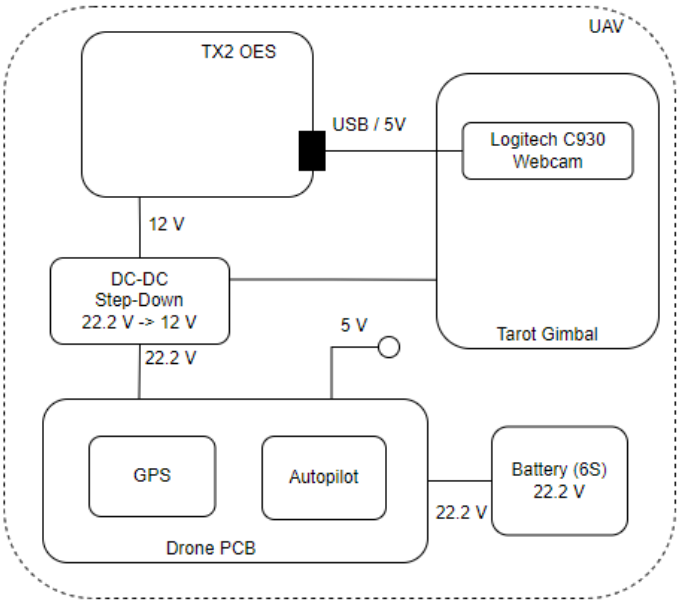

Figure 3: Power routing.

Software flow

The software consists of two python applications interacting with one another and the flowchart is shown in figure 4. Once the drone startup has been handled and the drone is at waypoint 0, the navigation and opencv applications takeover. The navigation and opencv application run in parallel. The navigation application has three parts: grid sweep, slow-approach, and pop detection. The opencv application simply runs non-stop, sending detections to the navigation script.

First main.py is called, and it's only purpose is to handle startup, arming, getting to waypoint 0, returning to launch, disarming, and exiting. It calls forth navigation.py, which handles drone operations, after reaching waypoint 0. When navigation.py returns it returns to launch and waits for the disarm.

Inside navigation.py, the drone begins by starting videoproc.py to start the video feed. Then it has the drone visit each waypoint in the grid. As the drone visits each waypoint, the application is listening for messages from videoproc.py, which at this time is simply analyzing the video. It sends a heart beat periodically and a message with object detection data. The detection data is (x position, y position, size). Heart beats are characterized by (65535, 65535, 65535). If a detection data packet is never received, the drone will return to launch at the end of the sweep.

Heart beats do nothing during grid sweep. However, receiving a packet with detection data causes the drone to save its location at the time it receives the detection data, and breaks the grid sweep. The drone is now performing the slow approach and is attempting to go to the balloon itself. During the slow-approach heart beats have a purpose.

While in slow-approach, each time detection data is received, the drone will yaw, decrease altitude, or move forward to center the balloon in frame. Additionally, the GPS location and yaw is saved. If the balloon size is too small it will move forward to advance, if the balloon size is big enough it knows that it can jog itself forward to pop the balloon. However, if the balloon leaves the frame, no detection data will come through the socket. At this point, heart beats will begin to accumulate on a heart beat counter. If the heart beat counter increases too much, the balloon will go into pop detection. Otherwise, a new detection data packet will reset the counter.

For pop detection, the drone goes back to the last location it saw a red balloon. If it does not see a balloon there, the heart beat counter will continue to accumulate. Next it goes to the position on the grid sweep where it saw the balloon. If the heart beat counter accumulates again, then the balloon is likely popped and navigation.py returns. Outside in main.py return to launch is started.

The videoproc.py application is running this entire time and will not close until ‘q’ is depressed while the video stream is the active window in VNC.

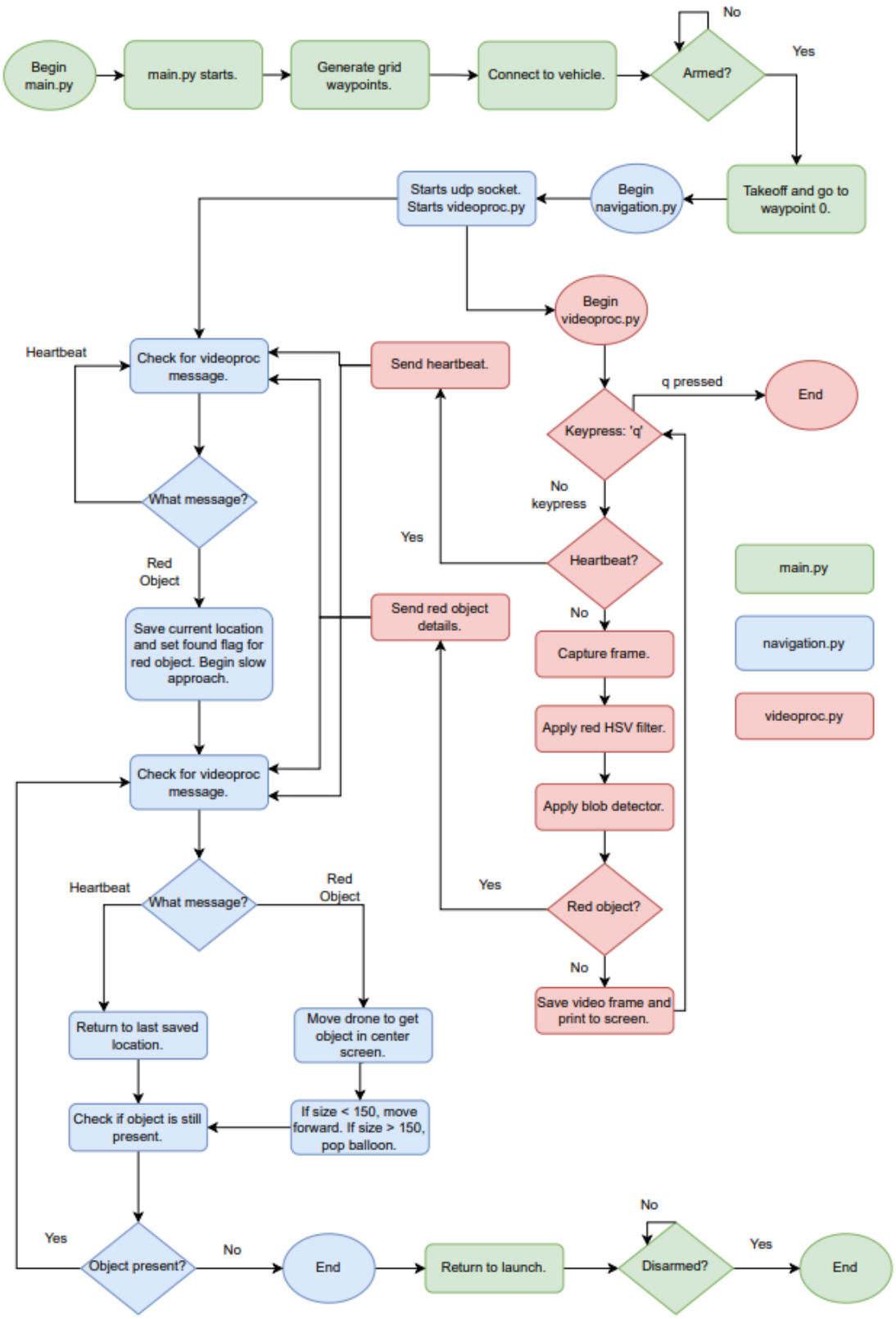

Figure 4: Software Flowchart.

Experimental Results and Analysis

There were multiple pieces of the project that had to come together to achieve the project's objective. Two parts of the project required tuning and are still the focus for improvements in this particular project. The other parts, such as the grid sweep, are solid and work exactly as expected. The two parts that will be discussed are the slow-approach and the object detection.

Slow-approach

The drone successfully popped the balloon. On the successful run, when the drone first spots the balloon it begins implementing the slow approach and slowly adjusts itself to keep the balloon near the center of the frame.The approach has built in fail safes for if the balloon is lost which is going back to the position where the balloon was last seen, and if that fails it goes back to the first location where the balloon was seen. After that it returns to launch. The pop detection worked well. However, the drone had to rely on these failsafes a few times before it successfully popped the balloon.

One reason for this is due to the lack of fine adjustment of the yaw, altitude, and forward position. Some of this is simply due to not having the accuracy required. The other bit results, which is probably most of the reason, from our algorithm. We yaw at 5 degree increments, 0.4 meter altitude increments and 2 ft increments. These values are fine for when the balloon is a few meters away, but not when it gets close. A 5 degree yaw too close to the balloon can cause the detection algorithm to lose the balloon. When this happens a failsafe is initiated.

If we had begun earlier with testing the slow-approach we may have been able to implement a more complex slow-approach. Ideally this would involve using adjusted values based on the balloon's position and size. For example, scaling the movements based on how far off from the center of the frame the object is and scaling down movement when the object becomes larger.

Another reason is that wind and altitude drift caused the balloon to leave the frame. This is what we had in mind when we thought of the failsafe. The video is processed fast enough that if the drone drifts due to wind, it will yaw to keep the balloon in frame. It is only in winds of a certain strength that the balloon loses frame. The autopilot does not automatically correct for wind, but there may be a better set of dronekit functions to use for engaging this aspect of the autopilot.

Figure 5: Drone popping the balloon

Object Detection

Detecting the red balloon involves placing a HSV filter over the video stream, ignoring colors not in the preset range of red colors, and then detecting a blob of specific shape. This method works to detect the balloon but the color range may need to be adjusted depending on weather conditions. Additionally, the detection method we used requires that the blob be fully on frame. We did not have time to probe this issue, but if the balloon touches the edge of the screen, it is lost. Therefore when we say the balloon leaves the frame, we actually mean that it has either fully disappeared, or has reached the edge. This was not a major roadblock, as the balloon was reachable by the popping mechanism before outsizing the frame.

Another issue we encountered was that small red blobs were detected in the green field and changed based on weather conditions. This was not looked into. However, a future remedy might be to acquire flight footage on many days of differing weather conditions and run the analysis to fine tune your parameters for all situations.

Figure 6: Red balloon detection

Conclusion

Our objective was to have the drone go off and pop the balloon on its own, and it did just that. Our design was able to leverage video processing from the on board video camera to detect and approach the red balloon. Although our algorithms did not provide the smoothest solution, they are a solution. There are many areas for optimization and re-organization and if provided more time we would pursue them. We would look into other video processing techniques to provide a more robust detection method and we would tune and upgrade the slow-approach algorithm.

We hope our results can be a jumping off point for future teams and for specific details pertaining to the organization of the project, check out the projects webpage.