ECE 592: Introduction to Autonomous Systems, Spring '17

Final Report

Team A3, "Pop a Red Balloon"

I. Abstract

The UAV we developed was capable of independently navigating to waypoints, identifying targets (red balloons) in the vicinity of the waypoint, and popping any identified balloons before returning to the launch point. Our platform was comprised of a custom-built quadcopter armed with a pen laser, and controlled by a Beaglebone Black single-board computer running ArduPilot. Additionally, an on-board Raspberry Pi 3 companion computer was used to run Python scripts for autonomous flight/mission control, visual balloon identification, LIDAR interfacing and laser control. For the final demonstration, the LIDAR and laser were removed and the propellers used instead to pop the balloon.

II. Introduction

The task can be viewed as three related sub-problems: drone control, visual target identification, and balloon-popping. Flight control of the drone can be further decomposed into waypoint navigation, target scanning (with feedback from the computer vision system), target approach (with LIDAR distance data in addition to visual feedback), and fine stabilization for target engagement. Target identification requires detection of color and contour in visually noisy outdoor environments. Once the preceding problems are solved, the simplest task-popping the balloon-requires only the switching of the laser (or close approach with the propellers).

While identifying, approaching, and bursting balloons with an aerial drone may not be valuable on its own, a more general view of these capabilities can find numerous applications. Color and contour detection has established applications in agriculture, and our system might be used for the control of pests and unwanted plants by replacing the laser with a sprayer.

III. Related work

1. Balloon finder project with ardupilot.

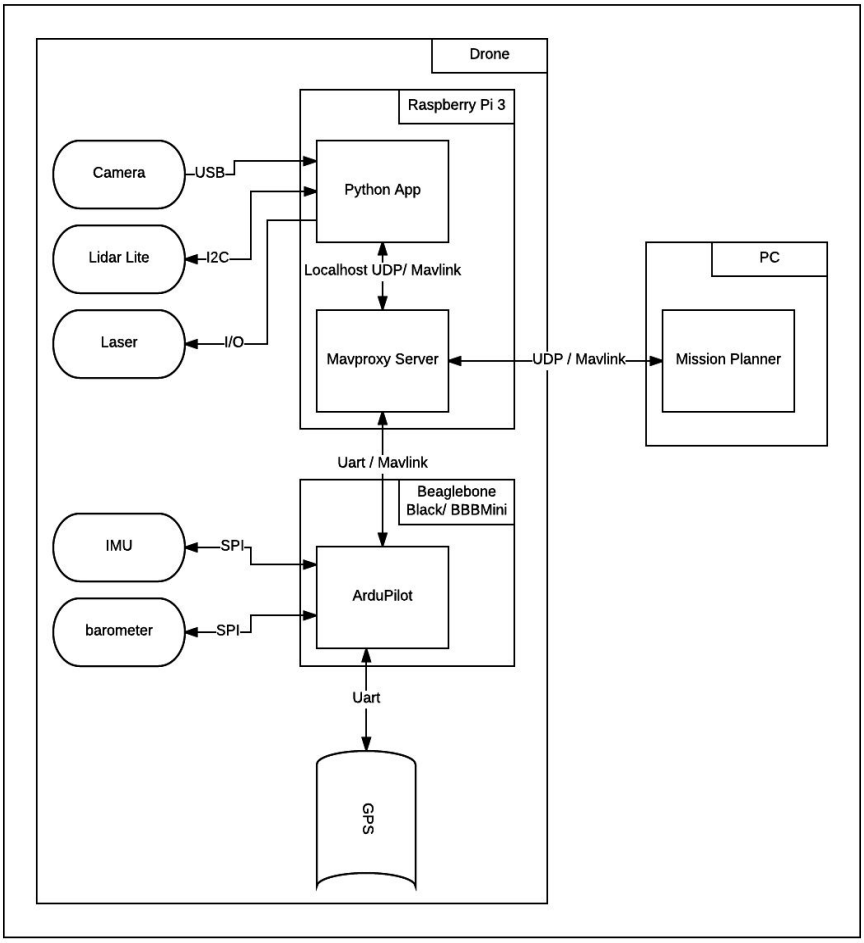
[Appendix 1] This project uses pixhawk as the autopilot and the ODROID-U3 as the companion computer. It uses the PID controller for the velocity vectors to approach the balloon and pops it using the propellers on the drone. In our project, we used Ardupilot on BBB Mini as the autopilot, Pi 3 as the companion computer. We used SolvePnP to find the vector to approach the balloon and pop it.

The main differences between this project and our project was the use of the pixhawk (a high end custom auto-pilot). We decided not to go with this solution because the pixhawk was too expensive.

2. Drone Target Acquire

[Appendix 2] This project uses a pre-built drone and function for target detection. It uses the square object detection instead of ellipse in OpenCV. Contour properties are much complex for the ellipse than the square. This project is much simplified version of our project because of the pre-built drone and much easier contour properties.

IV. Approach



The Balloon popping algorithm is divided into two parts:

1. Computer Vision Algorithm
2. Path Finding Algorithm

Computer Vision Algorithm:

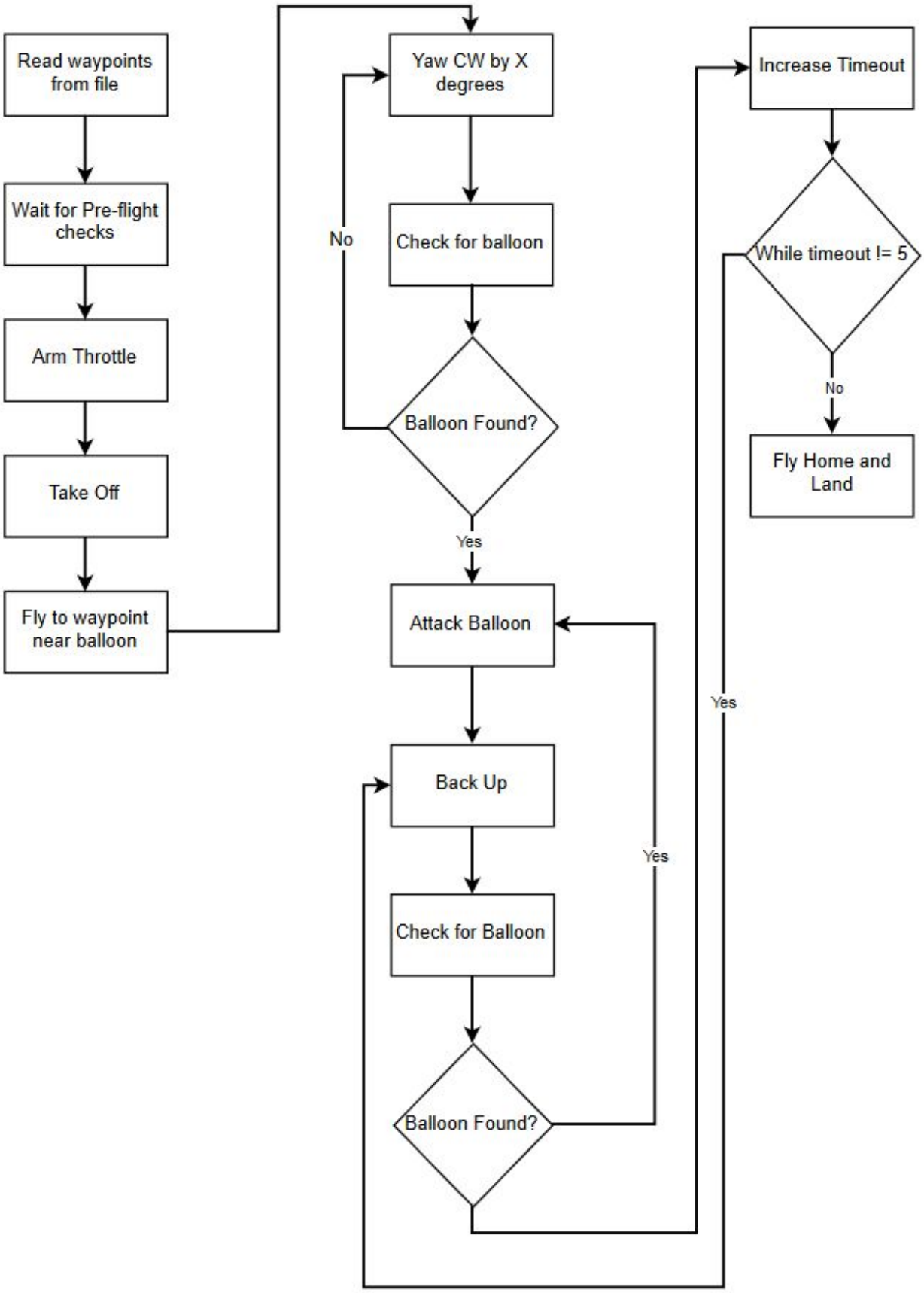
The image processing algorithm has two parts, identification and targeting. First the raw BGR image is run through a series of filters which convert it to HSV (hue-saturation-value) color space, as well as blurring it to sharpen the edges between objects, remove noise, and eventually change to a black and white image. Next, the cv2.findContours() function finds a list of all contours. The properties of each

contour are examined and compared to what we would expect the contour of a balloon to look like. For instance, the ‘bounding ellipse’ is drawn around the contour and their areas are compared (for a truly elliptical object, this ratio should be close to 1). We also looked at the ‘solidity’ of the object, drawing the convex full and checking for ‘empty’ space between them. This method proved to be fairly consistent at identifying the balloon in good to poor lighting conditions, and only failed to work when very bright reflections of sunlight appeared to make the balloon look white.

Once the list of all contours that are likely to be balloons are identified, we picked the one that most likely to be an actual balloon by sorting by size (the most common false positives were from small specks). Once this contour was identified, we found the ‘extreme’ points, the topmost, leftmost, rightmost, and bottom points and compared them to the pre-measured dimensions using solvePnP to find the ‘pose’ of the balloon. Because we only gave the 2D dimensions of the balloons outline, most of the pose matrix is garbage data but an accurate XYZ vector can be calculated, which can be used to create a waypoint.

Path Finding Algorithm :

The UML diagram below shows the high level overview of the navigation code. All of the code is available on the GitHub repository. We used Dronekit to send MAVlink command via the Command Long message (#76). We tried to write the entire app in GUIDED mode, but yawing was not working in GUIDED mode. We were able to get around this by setting a mission to loiter for an unlimited amount of time in AUTO mode. So, before we yaw, we switch into AUTO mode.The OpenCV code gives us an XYZ vector. This vector is given to the "SET_POSITION_TARGET_LOCAL_NED” command in MAVlink. The important thing to note here is that the “MAV_FRAME_BODY_OFFSET_NED” was sent as the 4th parameter in the command. This sets the drone itself as the origin for the North, East, Down coordinate. If you don’t use this parameter, the home location is used as the origin. As stated before, we ran MAVproxy on the Raspberry Pi. In order to get Dronekit to run on startup in MAVproxy, you need to change one of the MAVproxy start-up scripts. All of that information can be found on the “Getting Started” wiki on Dronekit [Appendix 4].


Read waypoints from file
Wait for Pre-flight checks
Arm Throttle
Take Off
Fly to waypoint near balloon
Yaw CW by X degrees
Check for balloon
No
Balloon Found?
Yes
Attack Balloon
Back Up
Check for Balloon
Balloon Found?
Yes
Yes
Increase Timeout
While timeout != 5
No
Fly Home and Land

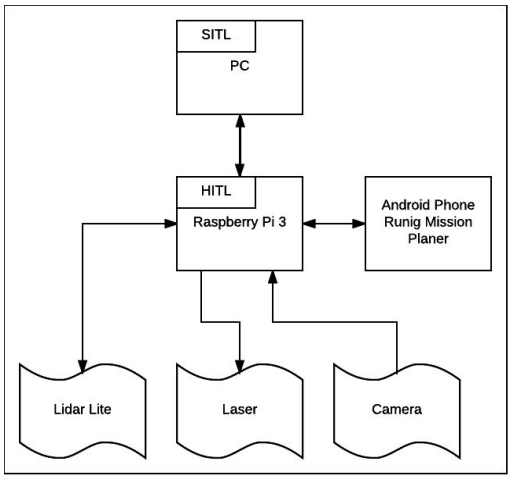
Laser Control:

On completion of approach, and with a sufficiently stable target image, the laser is controlled by switching a series low-side FET with a GPIO pin on the Raspberry Pi. The Python code for triggering this pin includes a user-adjustable timeout period to prevent extended laser activation. Triggering is in turn controlled by target tracking software with feedback from the LIDAR, and was tested successfully in lab conditions. As mentioned previously, this system was omitted for the final demonstration in favor of a "direct attack" using the propellers, due to flight stability problems.

An additional safety feature (planned but not implemented) is the inclusion of an RC switch between the 5V supply and the laser, which would prevent activation - regardless of GPIO state - without an established radio link and manual switch activation.

Software Testing:

Ardupilot has tools to help test software and hardware. The two main tools are Software in the loop (SITL) and hardware in the loop (HITL). SITL was used to simulate the real world drone. HITL was used to introduce real world inputs into the the system. HITL includes a Raspberry Pi 3, USB Camera, LiDAR Lite, Laser. See Figure. With the inputs and outputs, the path finding algorithm was tested to see if there was any faults in the logic. A red balloon was used to trigger the code in different states to get the desired outputs.



V. Analysis & Results

While the drone performed as expected in simulation, it demonstrated several shortcomings in the field, the greatest of which was seen in loiter mode. The loiter mode was not able to maintain an

accurate position. It would drift randomly within a one meter bubble. This error came both from instability in the hardware of the drone, as well as in the drone sensors. The drone both didn't seem to accurately know where it was and what its orientation was, and also had trouble maintaining what its current position was. Compasses are inherently unreliable, but even integrating the gyroscope readings frequently gave bad data.

Calibrating the image processing was also extremely difficult. Calculating a vector to the balloon was accurate within ~2 inches when running indoors, or from still images, but being able to find the balloon in broad daylight was much more difficult. In enough light, parts of the balloon appear to be completely white, and it seems that accounting for that much variance would require much more advanced image processing techniques likely incorporating machine learning.

In lab conditions (i.e. indoor lighting with a fixed target and laser platform), the laser exceeded expectations. At the experimentally selected target range of 1.6m, only 0.2s of laser activation was required to consistently burst balloons. It should be noted that laser effectiveness can vary with balloon manufacturer, color, and inflation level. While testing one package of balloons, the laser would melt through only to the point where the latex would become sufficiently translucent as to pass the rest of the light. Over-inflated balloons tended to be too translucent from the start, and would pass the laser light without any observable melting. The red balloons used for this project in the end were just right, and exhibited none of the aforementioned problems. However, depending flight conditions, the drone wasn't always able to hold itself steady enough for the laser to be able to pop the balloon.

In the end, we were able to pop the balloon with the propellers of the drone. As stated above, the loiter issues that we faced made us change to running into the balloon instead of popping it with the laser. Another issue that we had is that the drone lost altitude when it flies. This is due to the increased airflow over the barometer. Taping some paper towel over the barometer seemed to help, but it still lost altitude while flying to a waypoint or vector. It was still able to pop the balloon once it corrected its altitude.

VI. Conclusion

Of the three major sub-problems to the project task, our team managed to achieve all of them, with some caveats. Our drone has successfully and autonomously navigated waypoints, scanned and identified targets, popped a red balloon with the propellers, and was even able to consistently tell the difference between balloons and other red objects.. Our biggest obstacle was with how we pursued the actual control strategy. Writing a ground-station Python app was easy, but we knew that to be able to do any kind of meaningful feedback control we would need a lower latency and so we wasted a lot of time trying to figure out how to modify Ardupilot. Even after we switched to using a mavlink server, we found that it was not fully supported which made fine-tuned movements difficult. Another big obstacle was making the image processing robust enough to handle inconsistent lighting conditions. We accounted for this as best we could, but as discussed above bright light can easily blind a drone. During test flights, our drone had issues maintaining stability, and at least one flight ended in a crash that appeared to be a result of sensor malfunction or noise.

In future semesters, we think that the course material should start with the drone core components and then build up quickly to learning MAVlink, SITL, and HITL. These tools were critical in the success of our project. We would also recommend exploring machine learning options for image processing.This would likely help solve a lot of the problems caused by cheap and unreliable hardware, and help the drone deal with its inability to handle different light conditions. In the end, we still completed the task.

A. Appendices

1. https://github.com/rmackay9/ardupilot-balloon-finder/tree/master/scripts

2.http://www.pyimagesearch.com/2015/05/04/target-acquired-finding-targets-in-drone-and-quadcopter-video-streams-using-python-and-opencv/

3. https://github.com/Star-Wars-Drone/Python-App

4. http://python.dronekit.io/1.5.0/guide/getting_started.html