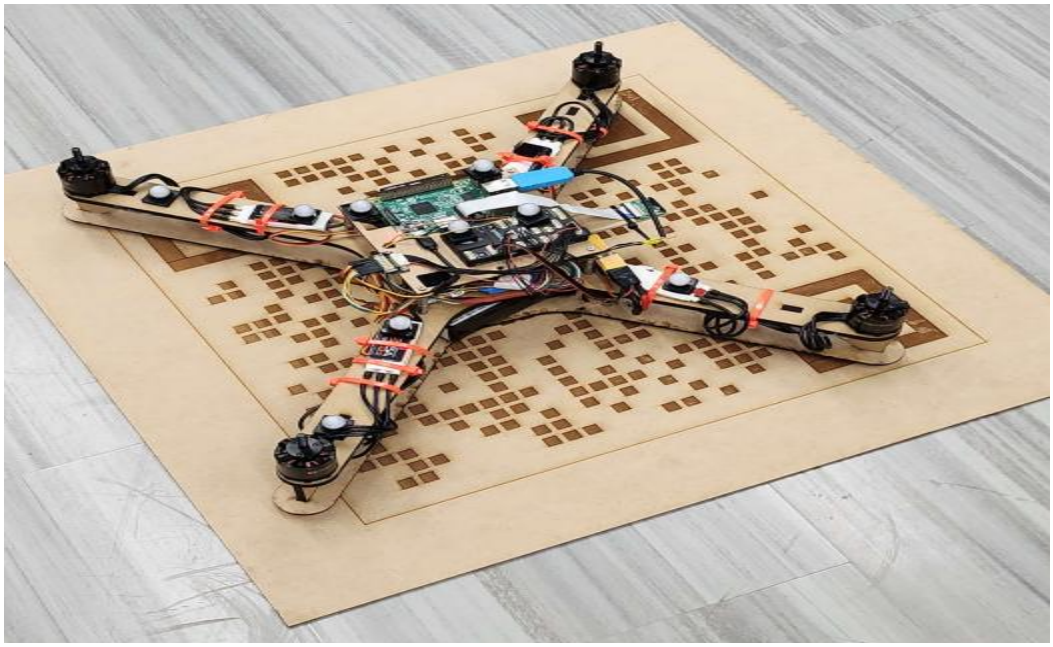
Vision Targeted Landing for a Transformable Solar Unmanned Aerial Vehicle (UAV)

Final Report

Fall Semester 2022



Andy Zheng

Archit Kalla

Daniel Carvalho

Grant Eichenberger

Kunal Rastogi

Advised by: Professor Nikos Papanikolopoulos and Travis Henderson

Abstract

The Solar UAV in development at the Minnesota Robotics Institute lacks critical components to enable autonomous flight. This project adds camera-assisted functionality in the form of an automated vision targeted landing system. The hardware and camera components work in a fully modulated project to identify a predetermined marker, calculate its real-world position, navigate and land. This project implemented all features onto a laser cut UAV produced in-house. Large targets were laser cut into our desired shape and laser etched with our targeting features. In house testing at the Center for Distributed Robotics demonstrated our ability to autonomously deploy, search for a visible target, navigate to the target, and land.

Introduction

Unmanned Aerial Vehicles, commonly known as UAVs are remotely operated aircraft responsible for an increasingly broad range of activities. Most modern UAVs fall into one of two wing-configuration categories: fixed wing aircraft and multi-rotor aircraft. The Solar UAV developed by Professor Nikos is of the novel transformer configuration. A transformer UAV seeks to take advantage of both predominant wing-configurations. Our goal in this project was to create a potential landing algorithm for this novel transforming UAV. In doing so, we focused and tested development on a barebones laser-cut quadcopter that would represent the multi-rotor configuration of the transformer.

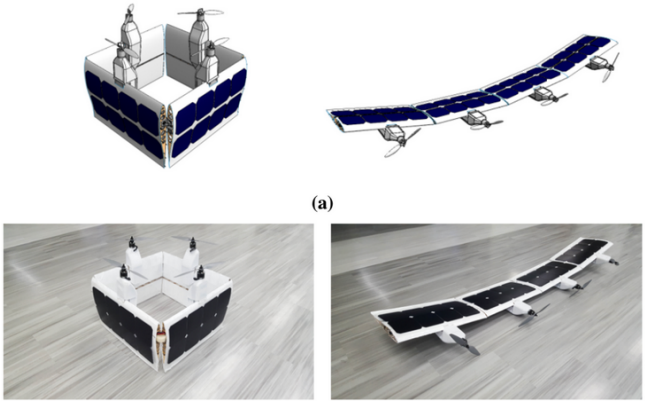


Figure 1: The Solar UAV prototype in multi-rotor (left) and fixed-wing (right) configurations

This report will present the reader with a comprehensive overview of the project, explaining the problem, design process, and the current state of affairs. This will be followed by a design evaluation and suggestions for future improvements to be made by a following group. All of this will be presented in a clear, concise fashion, allowing the reader to comprehend all aspects of our project in an expedient manner.

Inspiration and Problem

Every vehicle capable of flight has two sequences that are of the utmost importance: takeoff and landing. Without takeoff, a UAV would never achieve autonomous flight, and without landing, it would never fly again. Both are critical procedures for any UAV that seeks to reach commercial viability. This project seeks to focus on the latter, more valuable of the two algorithms.

By providing a safe, efficient landing procedure with an emphasis on low power consumption, we can be assured that our UAV will be able to land in both regular use situations and emergency low power situations. This heavily reduces the cost of operation, as a safe landing ensures there is a flight capable drone ready for recovery and further missions.

To begin our project, we knew that we would need to have some form of understanding the environment a drone is in. We discussed our options, from attaching a LiDAR module to the UAV to mounting a camera setup onto the UAV. We settled on using a stereo camera module as that would fulfill the following conditions:

1. Considerably lightweight, weighing less than 100 grams
2. Power-efficient, using less than 10 watts of power
3. Be able to provide suitable depth information for UAV landing

Early Hardware Decisions

Several requirements for the camera module were set in place. Compatibility with the onboard single-board processor was a must. The camera must interface with a Raspberry Pi 3B. We wanted to provide a solution that would resolve not only the location of unmarked landing spaces, but also be capable of resolving depth information of the landing spaces themselves to ensure a selected space was within levelness requirements. After looking through our options, we determined that a pair of IMX477 high resolution cameras as well as the ArduCam Camarray

HAT would fit our requirements. The Camarray HAT is necessary to splice and synchronize both camera feeds into a singular camera feed for the Raspberry Pi. After purchasing the module, the team split into two groups, one for beginning hardware fabrication, and the other for beginning development on the autonomous flight algorithm. A pair of 3D camera mounts to house and affix the individual camera modules into the bottom plate of the UAV were designed, and they are pictured below. Need for specific sizing of landing pads led to the design and fabrication of several laser cut wooden rectangles of various sizes to be used as landing pads.

For flight control, the prototype multi-rotor employed an on-board Pixhawk 4 autopilot controller board, running the px4 firmware as the flight controller.

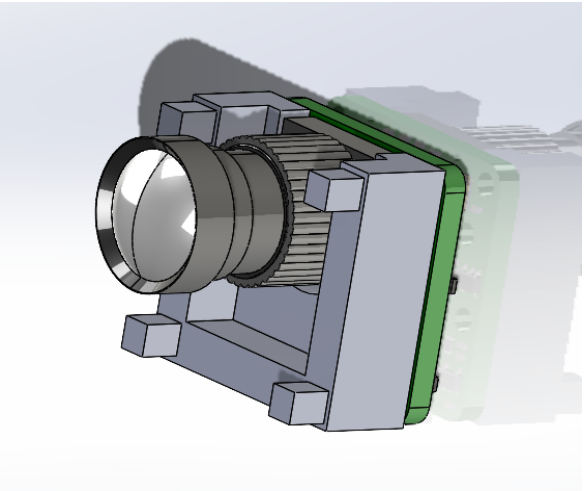


Figure 2: Camera Mount

Stereo Vision and Autonomous Flight Development

The first order of business for our project was to enable the multi-rotor prototype for autonomous flight. For this purpose we used a ROS package called MAVROS, which enabled our drone's onboard px4 controller to communicate with a companion computer. MAVROS has a function known as Offboard Mode, which allows us to run a ROS node on the multi-rotor that enables the drone for autonomous flight, including takeoff and landing. Several problems were immediately encountered in development, beginning with problems with entering 'Offboard Mode' on the PixHawk 4 Autopilot responsible for the actual flight of the UAV. We determined the issue was caused by some outdated firmware and incorrect configuration parameters in the QGroundControl software, the command center software that was running on the companion

computer. This was eventually resolved by updating the px4 firmware to version 1.13, and changing those QGroundControl parameters, and after that a simple takeoff into immediate landing procedure was completed. Complications then came up with regard to the camera module selected when integrating the module into the existing system. First, the camera module anticipated a more updated version of the Raspberry Pi OS. The module was designed for Ubuntu 20.04, with supported documentation for Ubuntu 18.04. However, the onboard Pi was running Ubuntu 16.04, a heavily outdated version of the OS that had significant performance issues relative to newer versions of Linux. After significant debugging, the team managed to get the module to function on Ubuntu 20.04. However, we were subsequently forced to downgrade to Ubuntu 18.04 in order to maintain compatibility within the ROS Melodic ecosystem currently in place on the UAV. As we downgraded, we noticed a significant performance degradation relative to Ubuntu 20.04. Once downgraded, the camera module ran into further issues once again, however the debugging process was much faster this time. After configuring the camera to work properly using the MIPI_Camera driver, we mounted our module onto the UAV for a live test. The initial camera images pulled off the UAV were very promising and provided very promising depth information, however in orientation they were slightly deviant from one another. In an attempt to fix this issue, we encountered a catastrophic problem as one of the camera modules was rendered unusable.

Moving to QR Code Detection

At this point in our project, we made a quick pivot to select a different vision targeting system. Instead of using a stereo camera module capable of resolving depth, we opted for a monocular camera module. While this would mean we could no longer determine depth information, we could still identify landing spaces by means of marking the target with a QR code. The laser cut landing pads were etched using the laser cutter with a QR code pattern, and the new camera module purchased at Micro Center provided immediate access to a functional camera. We then formulated a new plan for QR code landing, as shown in the diagram below. First the drone would take off, or already be in flight position. Then it would constantly scan for the QR code, and if one was not found, it would move in a spiral path to search for the code. Once a QR code was found, using trigonometry and known distance from the ground, it would

determine how far it needed to reorient itself in order to be above the code. Finally, it would land on the determined QR code.

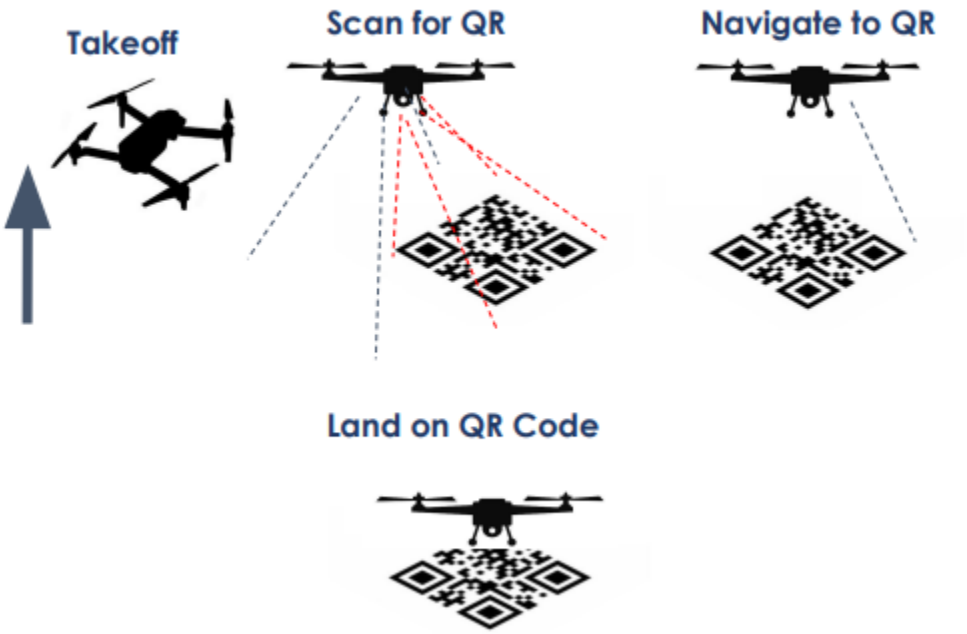


Figure 3: QR code detection landing process

After mounting the new camera to the UAV, we began coding. A significant issue occurred when attempting to utilize openCV, a popular computer vision library that has a built-in QR code detection algorithm, caused by dependency issues with the other software requiring Python 2.7. After spending 14 hours building the library from source code, we determined that it was unusable and pivoted to using pyzbar (a public python library for reading barcodes and QR codes) instead for the QR code detection. We were able to successfully obtain QR code position using this library and our own math, as shown below.

Figure 3: QR code bounding box creation and position calculation

To test our QR code algorithm, we held the drone up by hand and confirmed detection at our operating altitude of 1.5 meters. At this stage, we simply needed to fly the drone and have it navigate to the QR code on its own before landing. This was tested over the span of 3 days, and has the UAV taking off, navigating to the QR code, and landing successfully. We added on our plan of having the drone move in a spiral in case the code was undetected, and this concluded our goals for the landing algorithm.

Current State

The current state of the project has fully implemented a vision-targeted landing system from takeoff, in flight identification, correction, and landing. Utilizing a 5MP Raspberry Pi Camera Module v1.3(pictured below), we capture images at a resolution of 960 x 720 at an aspect ratio of 4:3. With a field of view at 55(H) degrees by 34.2 degrees(V) at our operating height of 1.5m, our operating field of view is 156cm by 92cm. With a QR code sized at 40cm by 40cm, we have a viable operating range of 116 by 52cm.

Figure 4: Camera Module v1.3

The flight sequence as currently implemented has the following sequences.

1. Wait for autopilot offboard control

2. Wait for autopilot arming
3. Initiate takeoff and navigate to (0, 0), altitude of 1.5m
4. Begin capturing images at a rate of 15 frames per second.
5. Scan images for visible QR codes.
6. If no image is found within 5 seconds, move to next position in an inverse spiral fashion(see Fig 5.). Each position is a 20cm movement.
7. Loop steps 5-6 until end of spiral or image found.
8. Calculate real-world position of the center of the QR code.
9. Publish real-world position offset of the QR code from the drone to PixHawk autopilot
10. Initiate landing when real-world position offset is <5cm in Euclidean distance.

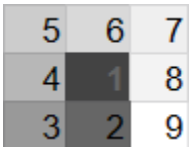


Figure 5: Inverse spiral with positions assigned as(1-9)

The algorithm and current implementation have been demonstrated to Travis Henderson and received his approval at the Senior Design Show. A hand-off folder has been created with all of our groups code and relevant design files created throughout the semester for future groups to begin work with.

Design Evaluation

Although our drone successfully executed its flight and landing algorithms, they were not without problems. The flight itself is fairly straightforward with minor errors due to Pixhawk autopilot failsafes. Once these failsafes are disengaged, the UAV flies as expected. This is only expected to be a temporary stopgap however and would need to be addressed in future iterations. Processor performance is a significant issue, as we are only able to resolve and analyze approximately 10% of our captured frames. The vast majority of our frames are dropped in that sense. Given the large number of dropped frames, we are also forced to stop for significant periods of time in flight in order to ensure we have fully scanned the visible area. Furthermore, the visible operating area of our camera is relatively low, with a vertical range of ± 25cm, it is not very flexible in the vertical directions. These are hardware problems that can be resolved

with a hardware solution. Our code is also unable to resolve a QR code that is too significantly rotated, although this is likely due to pyzbar's detection algorithm being rotation dependent. An upgrade to the detection method would likely resolve this issue. Our group has proven a barebones vision-targeted system to work on a barebones-UAV. Future work lies in modernizing the hardware and software used to ensure a respectable level of performance.

Next Steps

While meeting the design requirements of the customer, we have prioritized cost at the sake of performance. Moving forward, endeavors to increase performance while maintaining a similar cost level have several areas to investigate further. Throughout this project, we heavily relied on outdated or deprecated software out of necessity. Whenever we did utilize up-to-date software, it performed on the scale of magnitudes faster than the legacy systems. Thus, we believe that a successful upgrade of the baseline software such as operating system and ROS will provide a much appreciated bump in performance. It is also to be noted that while a Raspberry Pi provides enough room for development on the laser-cut UAV, a production variant of the Solar UAV would likely appreciate the power-cost-performance of an Nvidia Jetson [Orin] Nano processor. Other areas of hardware to be upgraded include the camera module itself. For the sake of simplicity and the pursuit of minimal cost, we utilized the Raspberry Pi v1.3 Camera Module. Referring to Table 1, this can likely be upgraded to the IMX477 camera, which would provide significantly better resolution and field of view. Performance upgrades are just one area to seek improvement in. We also suggest future groups to take advantage of our modularized code to adapt the purposes of our algorithm to target other visual targets to carry out more complex missions.

Appendix

Hardware Specification

	Camera Module v1	Camera Module v2	HQ Camera
Net price	$25	$25	$50
Size	Around 25 × 24 × 9 mm		38 x 38 x 18.4mm (excluding lens)
Weight	3g	3g	
Still resolution	5 Megapixels	8 Megapixels	12.3 Megapixels
Video modes	1080p30, 720p60 and 640 × 480p60/90	1080p47, 1640 × 1232p41 and 640 × 480p206	2028 × 1080p50, 2028 × 1520p40 and 1332 × 990p120
Sensor	OmniVision OV5647	Sony IMX219	Sony IMX477
Sensor resolution	2592 × 1944 pixels	3280 × 2464 pixels	4056 x 3040 pixels
Sensor image area	3.76 × 2.74 mm	3.68 x 2.76 mm (4.6 mm diagonal)	6.287mm x 4.712 mm (7.9mm diagonal)
Pixel size	1.4 µm × 1.4 µm	1.12 µm x 1.12 µm	1.55 µm x 1.55 µm
Optical size	1/4"	1/4"	1/2.3"
Full-frame SLR lens equivalent	35 mm		
S/N ratio	36 dB		
Dynamic range	67 dB @ 8x gain		
Sensitivity	680 mV/lux-sec		
Dark current	16 mV/sec @ 60 C		
Well capacity	4.3 Ke-		
Depth of field	approx. 1 m to infinity	adjustable with supplied tool	N/A
Focal length	3.60 mm +/- 0.01	3.04 mm	Depends on lens
Horizontal field of view	53.50 +/- 0.13 degrees	62.2 degrees	Depends on lens
Vertical field of view	41.41 +/- 0.11 degrees	48.8 degrees	Depends on lens
Focal ratio (F-Stop)	2.9	2.0	Depends on lens
Maximum exposure times (seconds)	6	11.76	670.74

Table 1: Raspberry Pi Camera specifications for officially supported modules