Real Time, Onboard-only Landing Site Evaluation for Autonomous Drones

by

Joshua Springer

Thesis proposal submitted to the School of Computer Science
at Reykjavík University in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy

September 2021

Thesis Committee:

Marcel Kyas, Supervisor
Professor, Reykjavík University, Iceland

Gylfi Þór Guðmundsson, Supervisor
Adjunct Professor, Reykjavík University, Iceland

Joseph Foley, Advisor
Professor, Reykjavík University, Iceland

External Person, Examiner
Role, University, Country

Real Time, Onboard-only Landing Site Evaluation for Autonomous Drones

Joshua Springer

September 2021

Abstract

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

Nulla malesuada porttitor diam. Donec felis erat, congue non, volutpat at, tincidunt tristique, libero. Vivamus viverra fermentum felis. Donec nonummy pellentesque ante. Phasellus adipiscing semper elit. Proin fermentum massa ac quam. Sed diam turpis, molestie vitae, placerat a, molestie nec, leo. Maecenas lacinia. Nam ipsum ligula, eleifend at, accumsan nec, suscipit a, ipsum. Morbi blandit ligula feugiat magna. Nunc eleifend consequat lorem. Sed lacinia nulla vitae enim. Pellentesque tincidunt purus vel magna. Integer non enim. Praesent euismod nunc eu purus. Donec bibendum quam in tellus. Nullam cursus pulvinar lectus. Donec et mi. Nam vulputate metus eu enim. Vestibulum pellentesque felis eu massa.

Ég veit ekki hvernig á að tala islensku

Joshua Springer

September 2021

Útdráttur

Morbi luctus, wisi viverra faucibus pretium, nibh est placerat odio, nec commodo wisi enim eget quam. Quisque libero justo, consectetuer a, feugiat vitae, porttitor eu, libero. Suspendisse sed mauris vitae elit sollicitudin malesuada. Maecenas ultricies eros sit amet ante. Ut venenatis velit. Maecenas sed mi eget dui varius euismod. Phasellus aliquet volutpat odio. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia Curae; Pellentesque sit amet pede ac sem eleifend consectetuer. Nullam elementum, urna vel imperdiet sodales, elit ipsum pharetra ligula, ac pretium ante justo a nulla. Curabitur tristique arcu eu metus. Vestibulum lectus. Proin mauris. Proin eu nunc eu urna hendrerit faucibus. Aliquam auctor, pede consequat laoreet varius, eros tellus scelerisque quam, pellentesque hendrerit ipsum dolor sed augue. Nulla nec lacus.

Suspendisse vitae elit. Aliquam arcu neque, ornare in, ullamcorper quis, commodo eu, libero. Fusce sagittis erat at erat tristique mollis. Maecenas sapien libero, molestie et, lobortis in, sodales eget, dui. Morbi ultrices rutrum lorem. Nam elementum ullamcorper leo. Morbi dui. Aliquam sagittis. Nunc placerat. Pellentesque tristique sodales est. Maecenas imperdiet lacinia velit. Cras non urna. Morbi eros pede, suscipit ac, varius vel, egestas non, eros. Praesent malesuada, diam id pretium elementum, eros sem dictum tortor, vel consectetuer odio sem sed wisi.

Sed feugiat. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Ut pellentesque augue sed urna. Vestibulum diam eros, fringilla et, consectetuer eu, nonummy id, sapien. Nullam at lectus. In sagittis ultrices mauris. Curabitur malesuada erat sit amet massa. Fusce blandit. Aliquam erat volutpat. Aliquam euismod. Aenean vel lectus. Nunc imperdiet justo nec dolor.

Etiam euismod. Fusce facilisis lacinia dui. Suspendisse potenti. In mi erat, cursus id, nonummy sed, ullamcorper eget, sapien. Praesent pretium, magna in eleifend egestas, pede pede pretium lorem, quis consectetuer tortor sapien facilisis magna. Mauris quis magna varius nulla scelerisque imperdiet. Aliquam non quam. Aliquam porttitor quam a lacus. Praesent vel arcu ut tortor cursus volutpat. In vitae pede quis diam bibendum placerat. Fusce elementum convallis neque. Sed dolor orci, scelerisque ac, dapibus nec, ultricies ut, mi. Duis nec dui quis leo sagittis commodo.

Acknowledgements

I would like to thank the Flying Spaghetii Monster and his noodly appendage.

Contents

Chapter 1

Introduction

1.1 Problem Statement and Motivation

The goal of the proposed research is to explore the topic of autonomous, unstructured drone landing. Current autonomous landing methods have at least one of the following disadvantages: they are blind to obstacles, they require previously *known* landing sites, they depend on sophisticated ground control stations for offloading of expensive computation. This proposed research targets a gap in current autonomous landing methods. Specifically, we aim to develop a method for quickly analyzing terrain and identifying safe landing sites using only embedded computational hardware and a minimal set of sensors.

Landing is a particularly difficult aspect of drone flight, owing mainly to its risky nature and required precision. As a result, most drone landings are carried out by (or under the supervision of) a human operator, inherently limiting the applicability of autonomous drones. Some autopilot software includes an Application Programming Interface (API) for *precision landing*, which allows a drone to localize and direct itself with respect to a landing pad during an autonomous landing, according to data provided by external sensors and programs. However, there is no particular method of autonomous landing in widespread use. As autonomous and semi-autonomous drones are not able to reliably handle landings on rough terrain or in non-ideal conditions, human operators often disable autonomous control during landing (opting for full manual control), or abuse/hack the landing system by descending to a low altitude, grabbing hold of the drone, and disabling the motors, as shown in Figure 1.1. Aside from potentially exposing users to dangerous rotors, this landing technique showcases the limitations induced by a lack of autonomous landing method.

In sufficiently flat, large areas, fully autonomous drone missions can end with a GPS-based autonomous landing which is blind to obstacles in the environment. However, intuitively and demonstrably, this can lead to crash-landings at landing sites that have obstacles within the error radius of the GPS, which can be anywhere from a few centimeters to a few meters. In the available open source autopilot softwares, obstacles are simply not handled, and drones will continue their landing attempts even if fatally obstructed.

1.2 Background

1.2.1 Autopilot Software/Hardware

The most prominent, open source drone autopilot software packages are ArduPilot [1] and PX4 [5], which can integrate easily with many additional/custom software packages. DJI drones, while the most commonly used consumer-grade drones, use proprietary, closed source autopilot software that has a limited API for interacting with external software. Thus, ArduPilot and PX4 and custom drones are typically used for research on drones themselves, while DJI software

Figure 1.1: Non-ideal, human-assisted landing in the absence of an autonomous, safe landing method that considers the surrounding environment.

and drones are typically used for consumer/commercial tasks. ArduPilot and PX4 communicate using the same open source, customizable protocol - MAVLink [3] - which has APIs in many different programming languages as well as with Robot Operating System (ROS).

1.2.2 Simulation Software

1.2.3 Robotics Software

1.2.4 Fiducial Markers

1.3 Related Work

Chapter 2

Current Progress

2.1 Building and Flying Two Hexacopters

After finishing a master thesis [9] wherein I developed an algorithm for autonomously landing a drone using fiducial markers in simulation, the next step was to test this method on physical platforms. The algorithm required identifying fiducial markers through image analysis, tracking the markers via a gimbal-mounted camera, calculating position targets in order to direct the drone towards the landing pad, and communicating those position targets to the flight control software. The base frame for the drones are the Tarot 680 hexacopter kit, which provides a good thrust-to-weight ratio, good flight stability, and space for mounting multiple computational components. A combination of Raspberry Pi and Navio2 [4] shield serve as a flight controller which can communicate with a companion board. The companion boards (a Google Coral Dev board and an NVIDIA Jetson Nano) communicate via a USB network to the flight controller and perform all heavy computations involving image analysis, coordinate system transforms, PID control, and position target generation.

An overview the components is as follows:

- **11.1 V LiPo Battery:** this battery provides power to a battery eliminator circuit (BEC) for isolation of the power system for the computational electronics (the flight controller and companion board).
- **BEC (Battery Eliminator Circuit):** the BEC transforms 11.1V power to 5V power for the flight controller and companion board. The flight controller and companion board each have their own 4A channel to meet their given power requirements.
- **Flight Controller:** this combination of a Raspberry Pi 3 B+ and Navio2 shield runs the ArduPilot software to control the drone, and communicates with the companion board to control the gimbal.
- **Telemetry Radio:** the telemetry radio provides two-way communication between the flight controller and a ground control station that is also fitted with its own telemetry radio. It is connected to the flight controller via USB. The software on the ground control station provides an interface for real-time status messages and sending high-level commands.
- **RC Receiver:** the RC receiver provides a one-way radio link between the pilot's transmitter and the flight controller, allowing the pilot to manually control the drone. It is connected to the flight controller via SBus which provides an 8-channel multiplexed PWM signal to reduce the needed wires and space. This provides an interface for control by a human pilot, which is often used in testing but will eventually be mostly unused.
- **22.2 V LiPo Battery:** this battery provides power to the speed controllers and gimbal.

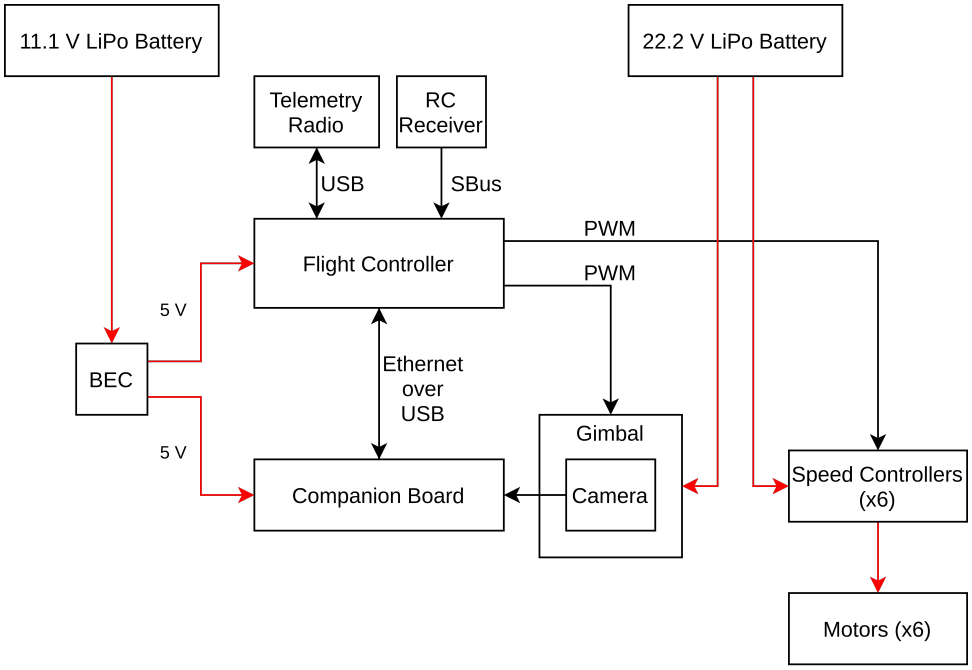


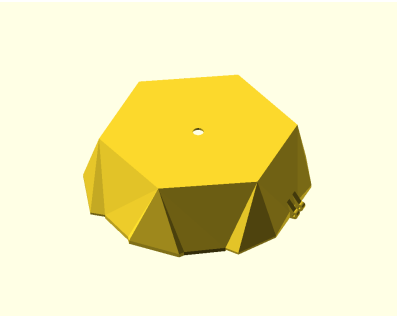
Figure 2.1: Hardware Setup

- **Speed Controllers:** the speed controllers receive a PWM signal from the flight controller which indicates a throttle value. They then provide corresponding power signals to the motors.
- **Motors:** the motors spin propellers to provide thrust in order to control the drone's position in the air.
- **Gimbal:** the gimbal controls the orientation of the camera based on PWM signals from the flight controller which indicate target angles. Its onboard IMU and driver filter the motion of the camera in order to provide a smooth camera image.
- **Companion Board:** the companion board reads an image from the camera and calculates the position of the landing pad relative to the drone. It then communicates this information to the flight controller via an Ethernet over USB connection using ROS.

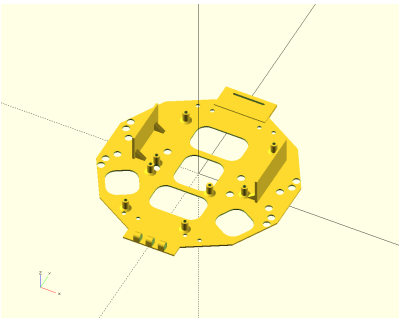
The computational components require some protection from the harsh Icelandic weather, and we therefore designed and 3D-printed a component mounting plate with a connector for a canopy. We also designed and printed cases to protect camera modules and allow them to be mounted in a gimbal with a GoPro form factor. The final versions of these components (after several iterations) can be seen in Figure 2.2. The fully assembled hexacopters are shown in Figure 2.3

The algorithm is implemented as a set of ROS modules, as explained below:

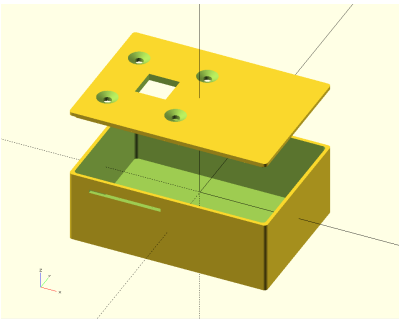
1. `gscam` retrieves camera input and makes it available as a ROS topic,
2. `whycon_ros` analyzes camera images to detect WhyCon/WhyCode markers and determine their pose,
3. `gimbal_controller` reads the poses of any detected markers, passes this information as input to two PID systems, converts the output of the PID systems to PWM outputs, and forwards the PWM outputs to the autopilot software to control the gimbal.
4. `landing_controller` reads the poses of any detected markers, performs coordinate system transforms to generate a target position, and forwards them to the autopilot software.



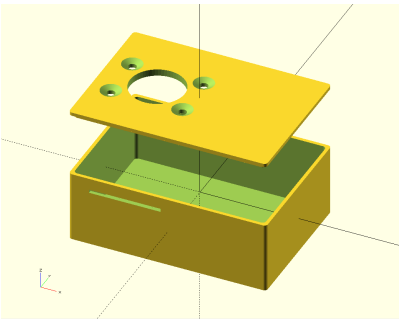
Protective Canopy.



Component mounting plate.



Google Coral camera case.

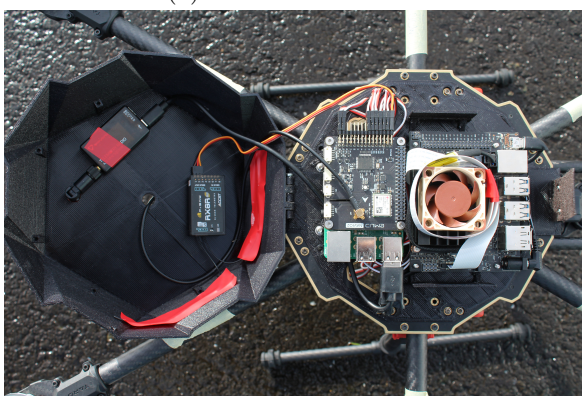


Jetson Nano camera case.

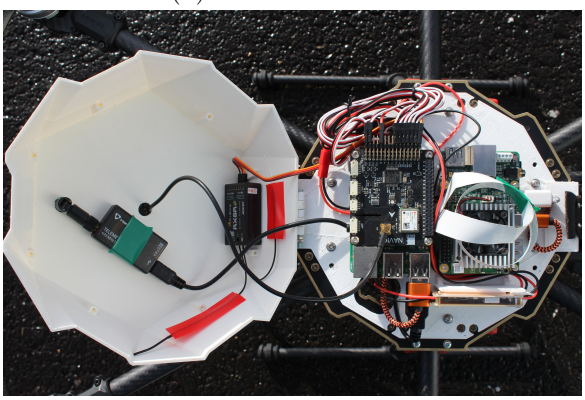
Figure 2.2: 3D printed parts for the Tarot hexacopters.

(a) The Jetson drone.

(b) The Coral drone.

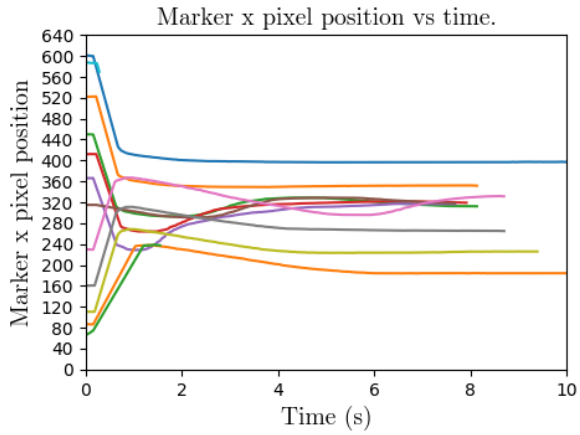


(c) The Jetson drone's electronics compartment.

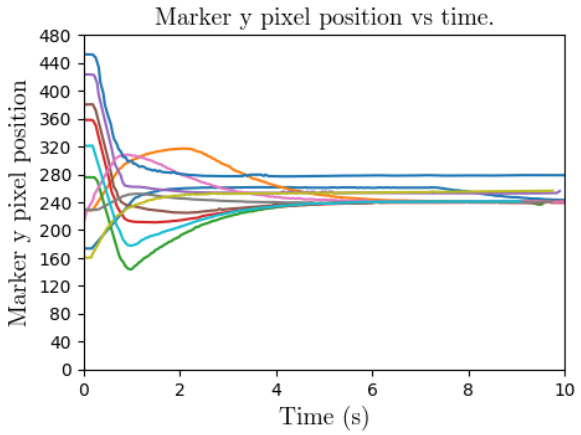


(d) The Coral drone's electronics compartment.

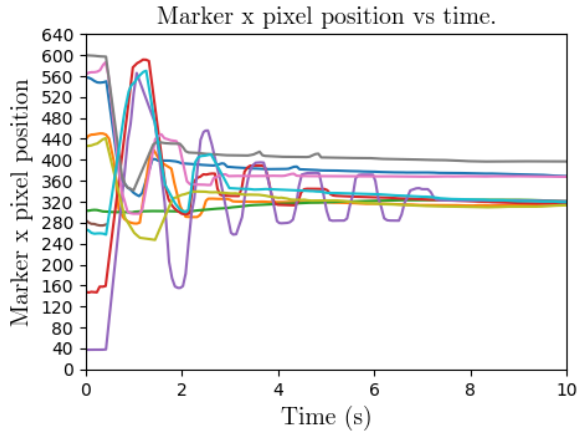
Figure 2.3: The assembled drones and their electronics compartments.



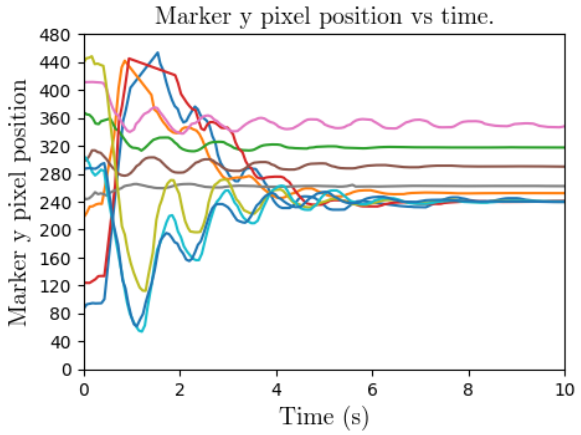
(a) Performance of the Jetson drone aiming the gimbal in the x axis.



(b) Performance of the Jetson drone aiming the gimbal in the y axis.



(c) Performance of the Coral drone aiming the gimbal in the x axis.



(d) Performance of the Coral drone aiming the gimbal in the y axis.

Figure 2.4: Performance of aiming the gimbals.

2.1.1 Results

The drones fly with good stability even in high winds, with an estimated 15-20 minute flight time. [1] In lab tests and in flight, they are able to detect and track fiducial markers using the method tested in simulation. However, only the more lightweight WhyCon/WhyCode fiducial system was used in testing, instead of the April Tag system, because of the time constraints of this summer project. The performance of the drones in tracking the markers is shown in Figure 2.4, where each subfigure shows the position of the marker in the camera frame in the given axis, with a resolution of 640x480 pixels after resizing in order to decrease the computational requirements of the image analysis. The wide angle lens of the Jetson Nano camera module results in much smoother tracking, since each pixel corresponds to a larger distance than with the Google Coral camera module.

The drones are able to approach the landing pad autonomously, but have not yet touched down autonomously. This is due to two principal factors. First, GPS precision is low in Iceland (i.e. geometric dilution of precision (GDOP) is relatively high) because of Iceland's distance from the equator. Methods of autonomous positioning within the ArduPilot framework which were not ostensibly GPS-based are eventually translated into lat/lon/alt position targets, and the drones attempted to navigate to them with GPS only (instead of other methods such as dead

[1] Video of some of the flights and landing attempts can be found at: https://vimeo.com/461576798

reckoning). Since the GDOP was prohibitively high, the drones could not accurately estimate their position. Therefore, they could only follow a coherent path for a limited amount of time when navigating autonomously, after which the trajectory was unpredictable, even if the position target commands were correct.

Second, there is a fundamental challenge with fiducial markers which comes as a result of the limitations of embedding 2-dimensional shapes into 3-dimensional spaces. While the position of the marker in the camera frame can be detected unambiguously, the orientation of the marker cannot. Especially when the marker is almost normal to the camera's view, its orientation becomes increasingly ambiguous in the roll and pitch components. (This can be intuitively visualized by imagining the difference between the appearances of a marker when it is at $(R, P, Y) = (1°, 0, 0)$ versus when it is at $(R, P, Y) = (-1°, 0, 0)$. These orientations would be difficult to distinguish even for the human eye, and much more difficult for a camera with 640x480 pixels.) Tests in simulation revealed this phenomenon, however, because of a variety of factors (a more "perfect" camera/marker/world, better GPS, lack of logistical constraints, etc.), this phenomenon was not prohibitive in simulation, and successful landings were plentiful.

These two problems set the stage for more research and modifications to the fiducial systems, outlined in Sections 2.2 and 2.3.

2.2 WhyCode Modifications

2.2.1 Background

The WhyCon fiducial marker, shown in Figure 2.5 consists of a white circle inside of a larger black circle. The identification algorithm is computationally simple:

- An input image is searched for white regions.
- The detector flood-fills and determines a measure of circularity for each white region.
- The detector seeks a black region directly outside of each white region, flood fills it, and determines a measure of circularity for it.
- The circularity measures are compared, and the white/black region combination is called a WhyCon marker if it is adequately circular (under projection).
- The position of the marker is determined using the pinhole model of a monocular camera with known intrinsic parameters (pixel resolution, lens distortion, principal points, focal lengths), and the diameter of the marker.
- The major and minor axes of the elliptical regions provide a basis from which to determine the marker's orientation.

The WhyCon software's purpose is to determine the pose (position *and* orientation) of detected WhyCon markers with respect to the camera field of view. The position of the markers can be determined with centimeter accuracy in most use cases. However, the full radial symmetry of the WhyCon marker inherently prevents the assignment or recognition of a "yaw" orientation - that is, only two components of the marker's rotation can be determined.

WhyCode expands on WhyCon to add an ID and yaw orientation to the plain WhyCon marker. While the majority of the detection algorithm is the same, an ID is encoded onto the marker in the form of a Manchester encoding that is wrapped around the inner, white circle, as shown in Figure 2.6. By sampling along the circle halfway between the white circle and black circle, the detector can determine the ID encoding. Figure 2.7 shows potential collisions between WhyCode markers that are rotationally symmetric but have different IDs. While it is impossible to distinguish between rotationally symmetric markers, this problem can be overcome

Figure 2.5: WhyCon marker.

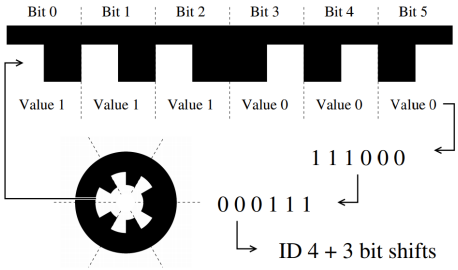


Figure 2.6: WhyCode ID Manchester "Necklace" Encoding.

by bit-shifting the ID's binary string to its lowest value in all cases before determining the ID. This also gives a means of determining a "yaw origin" so that the orientation of the marker can be determined in 3 dimensions.

2.2.2 Orientation Ambiguity

In contrast to the accurately/unambiguously recognizeable position of WhyCode markers, the orientation is ambiguous in many circumstances, as can be seen in Figure 2.8, particularly during time $t \in [4, 8]$. This ambiguity comes from the fact that the orientation is principally calculated from the semi-axes of the marker's ellipses under the assumption that the marker is truly a circle. Calculating the orientation based on these semi-axes gives *two* candidate solutions that satisfy the assumption that the marker is facing the camera, and it is difficult to determine which of these candidates represents reality, especially when the marker is normal or near-normal to the camera's field of view. The original WhyCode software [6] arbitrarily chooses the first solution as correct, and as such it does not need much analysis except to say that it has a high likelihood of choosing the wrong solution.

In the context of fiducial-based drone landing, the issue of orientation ambiguity

Refinements of the code, such as that proposed by Jiri Ulrich [10] attempt to disambiguate the orientation, with partial success. This method assumes that all "teeth" that form the marker's ID should have the same number of sample points, and therefore chooses the candidate solution that has lower variance of the number of sample points per tooth. The samples are collected along the ellipse that is halfway between the white and black ellipses of the marker. The candidate solutions predict this ellipse to be in slightly different places, owing to the distortion caused

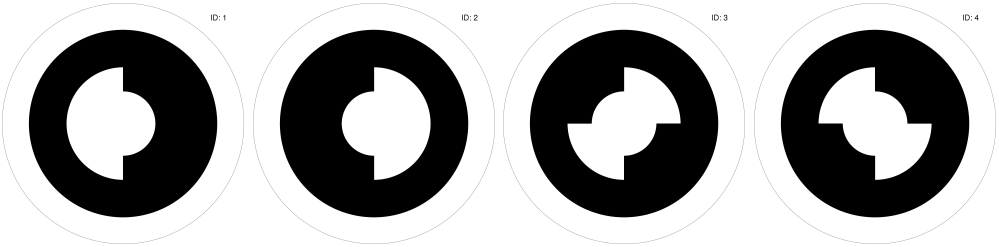


Figure 2.7: Rotational symmetry in 2-bit WhyCode markers.

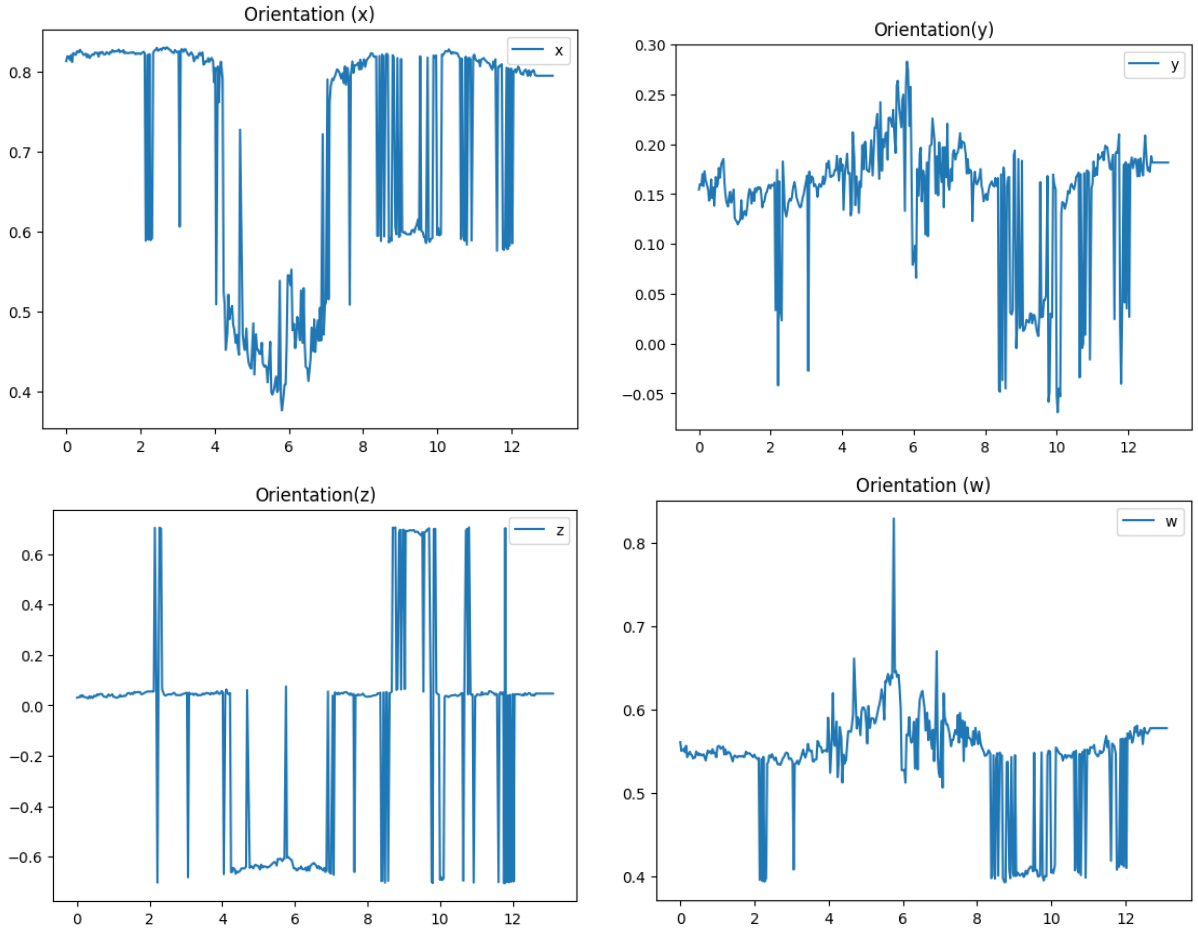


Figure 2.8: Components of the orientation quaternion for a WhyCode marker. Emphasis is placed on the discontinuties in all components, and sign flipping in the Z component.

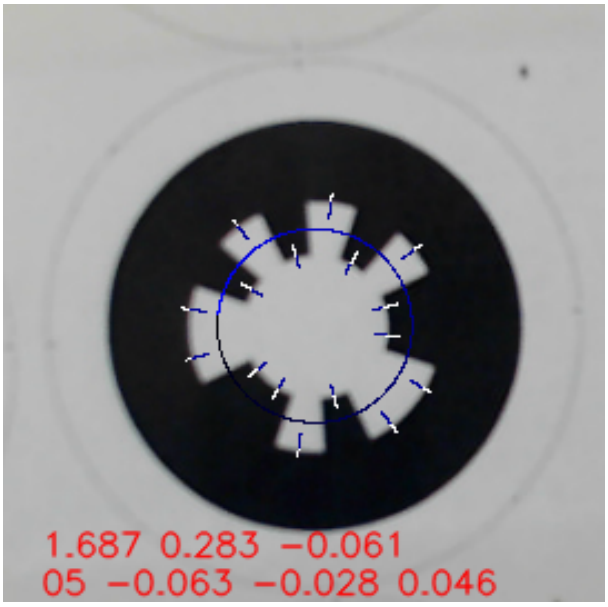


Figure 2.9: Example of sampling WhyCode markers on the radial edges of the teeth. The black/blue circle illustrates ID sampling locations, with the bluest samples occurring first, and blackest samples occuring last. Each of the blue/white radial line segments illustrates the sampling of the predicted edges of the teeth. The method chooses that solution that minimizes the variances of the ratio of blue to white on these line segments.

by the camera lens. The rationale for the method is that the correct solution should predict the ellipse to be in its correct place, minimizing the variance in the number of sample points that coincide with each tooth. Conversely, the incorrect solution should predict the ellipse to be in the incorrect place, and therefore fewer sample points should coincide with teeth one side, increasing the variance. This provides a baseline on which to disambiguate the solutions, but does not reliably choose the correct one, especially as the candidate solutions get closer and closer and the marker becomes more and more normal to the camera.

2.2.3 Proposed Solutions

As part of this project, I explore two different ways of solving the orientation ambiguity.

Radial Tooth Edge Sampling

The first method (the `ellipse_sampling` branch of [8]) is similar to that proposed by Jiri Ulrich, in that it is based on decreasing variance in sample points along the teeth. After the two candidate solutions for the orientation and the marker ID have been determined, a second phase of sampling begins. Each candidate solution predicts the center of the inner, white circle in pixel coordinates. (Since this system is designed to be lightweight, the center is not actually calculated, nor needed, to determine the position of the marker, and explicit calculation of this center is neglected by default.) Given the structure of WhyCode markers, we can assume that the circle along which the ID is sampled has a radius of $1.5r$, where r is the radius of the inner circle, and the radius of the circle bounding the teeth is $2r$. Further, from the ID sampling, we can identify the angle that goes through the center of each tooth. This makes it possible to test how well the candidate solution is centered on the marker at many points, and leverages the fact that camera distortion increases (and therefore prediction errors increase in magnitude) as one moves away from the center of the image.

Co-planar Markers

The second method works under the assumption that all markers are co-planar. Since the WhyCode system can accurately and quickly determine the positions of the markers, this assumption allows the orientation of co-planar markers to be determined unambiguously, *when 3 or more markers are detected simultaneously.*

2.3 April Tag Modifications

2.4 Experiments with AirSim

2.5 Small Drone

In an effort to avoid some of the logistical limitations induced by the larger drones, e.g. transportation, material costs in the case of crashes, maintenance, overhead from the creation of component covers and mounts, and the use of large batteries, I have adapted a drone design [2] from Thingiverse.com for the purposes of autonomous landing. The base design is a very simple quadcopter platform with a large area for mounting electronics, a flexible motor mount, a symmetric arm design which allows all arms to be interchangeable with one another, and a simple press-fit system for attaching the arms to the body. After modifications, it now includes a pitch-gimbal mount for a Raspberry Pi camera module that is controlled by a micro servo, mounting holes for various components (including the Raspberry Pi, power distribution board, metal legs, and standoffs for an "upper deck"). All modifications were made with OpenSCAD and are available in the drone's github repository [7] The drone runs on a 3S (11.1V) lithium polymer power system, with a DJI Snail propulsion system.

2.6 Infrared Camera, Heavy-Lift Drone

Chapter 3

Research Plan

3.1 Data Set Generation

3.2 Terrain Classifier Creation

3.3 Testing in Simulation

3.4 Testing on Physical Hardware

3.5 Risk Analysis

Bibliography

[1] Ardupilot.org.

[2] Thingiverse (arks007). Raspberry pi drone (niacam: Txsef 2016-2017). (accessed: 2021.9.24).

[3] Dronecode. MAVLink Micro Air Vehicle Communication Protocol. (accessed: 2021.5.19).

[4] Emlid. Navio2. (accessed: 2020.6.5).

[5] Lorenz Meier. Pixhawk. (accessed: 2020.6.5).

[6] Matias Nitsche and Tomáš Krajník. Original whycon ros github repository. (accessed: 2021.9.30).

[7] Joshua Springer. Raspberry pi drone. (accessed: 2021.9.27).

[8] Joshua Springer. Whycon ros github repository. (accessed: 2021.9.30).

[9] Joshua Springer. Autonomous Landing of a Multicopter Using Computer Vision. Master's thesis, Reykjavík University, 2020.

[10] Jiri Ulrich. Whycon ros github repository. (accessed: 2021.9.30).