Real Time, Onboard-only Landing Site Evaluation and Landing Execution 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

March 2022

Thesis Committee:

Marcel Kyas, Supervisor
Professor, Reykjavík University, Iceland

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

Joseph Foley, Advisor
Professor, Reykjavík University, Iceland

Sebastian Scherer, External Examiner
Associate Research Professor, Carnegie Mellon University, US

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

Joshua Springer

March 2022

Abstract

Landing is a remaining, unsolved problem in autonomous multi-rotor drone flight. Many other tasks such as takeoff, waypoint-to-waypoint flight, and miscellaneous mission tasks (e.g. collection of images and video) have been reliably automated. Yet, autonomous landing remains largely a manual task because of its inherently risky, sensitive nature. The critical result of this is that fully autonomous mission cycles are just out of reach with current technology, in many contexts. Prior work towards autonomous multi-rotor landing has been subject to at least one of several disadvantages — either it depends principally on GPS and is therefore subject to possibly fatal inaccuracy (especially in Iceland), it has relied on detection of special markers (known beforehand) with a downward facing camera (such that it may easily lose sight of the markers during approach and descent), it uses differences in pixel speed to deduce terrain topology (but therefore depends on motion), or it has depended on sophisticated ground stations to carry out the computationally expensive processing required for terrain analysis. The proposed research targets the problem of autonomous landing with the goal of creating an algorithm to reliably land multi-rotor drones with the following constraints: 1. having no prior knowledge of a landing site or GPS position, 2. executing in real time with a critical deadline, and 3. using only the limited computational environment onboard a drone.

Thus far, we have created and tested several drone platforms with 3 different flight control software stacks (ArduPilot, PX4, and DJI), which have revealed many challenges of operating drones in Iceland: low GPS accuracy, unpredictable winds, and the commonality of rain. While a large, weatherproof drone system can withstand wind and rain, low GPS accuracy has proven to be a real obstacle. DJI drones and flight controllers have shown to vastly out-perform others in this regard, but additional sensors, e.g. LIDAR rangefinders and optical flow sensors, may improve flight performance for ArduPilot/PX4 systems. Further, we have tested landing algorithms based on fiducial markers in simulation, but differing from previous work in that we use a gimbal-mounted camera to allow it to track the marker over time. This has involved the further development of existing fiducial systems (April Tag and WhyCode) to optimize for execution on embedded hardware, and to test the accuracy of their orientation estimation. Finally, we have created a proof of concept of such algorithms on a physical drone.

Moving forward, we plan to expand on the following research questions: **RQ1.** With what methods can a drone autonomously identify a safe, previously unknown landing site? **RQ2.** What data do such methods require? **RQ3.** How can such methods execute in real time, and in the power-limited environment of a drone? We hypothesize that **H1:** A U-net, or variants such as Residual U-net with pre-/post-processing steps for image rectification and communication with flight control software will be able to recognize landing sights from drone sensor data. **H2:** This data can be point clouds from LIDAR units or RGBD (image + depth) cameras, which are small enough to be embedded onto a drone. **H3:** An embedded TPU, GPU, or FPGA will be able to execute the method onboard a drone in real time. We plan to generate training/testing data sets of LIDAR/RGBD point clouds in AirSim, and collect further testing data sets from the real world using our existing drone platforms. This data will serve as the basis for training several neural network models based on the U-net architecture. Promising algorithms will be optimized using pruning and will be tested for execution speed and power consumption on physical hardware. Any sufficiently fast/accurate methods will be tested in real landing scenarios.

Contents

Chapter 1

Introduction

Figure 1.1: Non-ideal, human-assisted landing of Christopher Hamilton's Matrice 300 at the Fagradalsfjall volcano in the absence of a safe, autonomous landing method that considers the surrounding environment.

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 *known* landing sites, or they depend on sophisticated ground control stations for offloading of expensive computation. The 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 (and then carry out the landing).

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 a human operator, inherently limiting the applicability of autonomous drones. Some autopilot software includes an 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. A proven alternative to depending on GPS for landing is to provide additional infrastructure at the landing area to help the drone navigate reliably, e.g. IR beacons, fiducial markers, or even ground-based cameras that identify and direct the drone. However, this paradigm requires additional overhead, in that the landing site should be sophisticated and known ahead of time.

Some methods for autonomous landing employ in-depth topological analysis of the environment, which are computationally- and power-intensive. Due to the hardware onboard most drones, which is limited in terms of power and computational performance, it is often necessary to stream data from the drone to a more powerful ground station to carry out this analysis. However, this adds the requirement of a two-way data link between the drone and the ground station, which limits the range of the drone, adds potentially prohibitive latency, and increases operational overhead.

Therefore, the goal of this research is to determine which methods a drone can use for autonomous landing in unknown environments, to determine the data those methods require, and to determine how those methods can execute onboard a drone in real time. We aim to implement at least one of these methods on a drone and demonstrate it as a real world proof of concept.

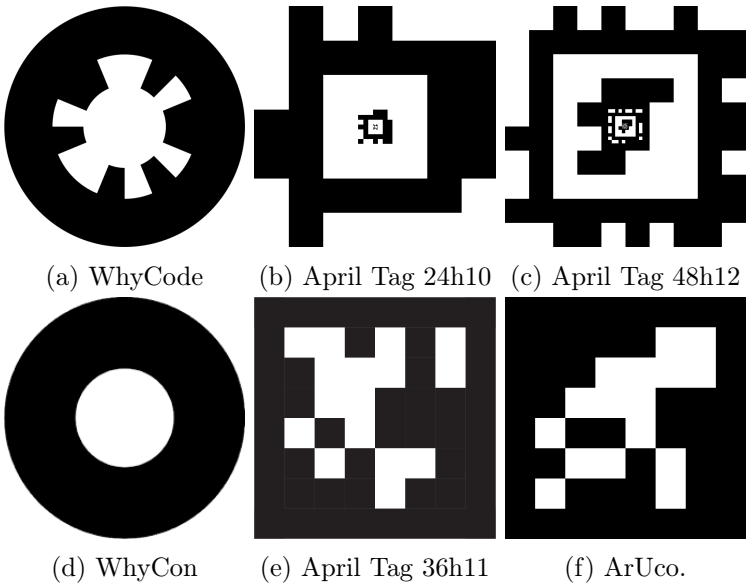
1.2 Background

1.2.1 Autopilot Software/Hardware

The most prominent, open source drone autopilot software packages are ArduPilot [1] and PX4 [2], 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 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 ROS.

1.2.2 Robotics Software

ROS is a common framework for robotics software that faciliates communication, interactivity, and compatibility between cooperating software modules. It uses a Publisher/Subscriber model of to allow concurrent programs to communicate, and even provides infrastructure for communication between threads running on different hardware platforms. Within the ROS universe are open-source, ready-built tools that are fundamental to common robotics tasks, such as interfaces to many different sensors and cameras, modules for rectification, compression, and transmission of images and point clouds, PID controllers, coordinate system transforms, and much more. ROS provides a structured means of generating modular solutions to complex robotics problems.



(a) WhyCode (b) April Tag 24h10 (c) April Tag 48h12

(d) WhyCon (e) April Tag 36h11 (f) ArUco.

Figure 1.2: Some examples of fiducial markers.

1.2.3 Simulation Software

Simulation allows robotics developers to test algorithms quickly without logistical concerns such as weather, transportation, damage to hardware, etc. Gazebo is one of the most common robotics simulators and provides a physics engine, many simulated worlds and models of items/vehicles, integration with many robotics tools, and — importantly — lots of user-generated content, software plugins, and information. It has integration with ROS, so that simulated sensors and actuators can provide data and take commands from ROS modules. This allows developers to analyze the behavior of their robots during development cheaply and quickly. Additionally, Gazebo has integration with both PX4 and ArduPilot, which, combined with existing (and free) drone models, can be used to simulate autonomous drone control algorithms in depth. AirSim is another robotics simulator, but it focuses on providing accurate physics and graphics and is therefore a heavier software package. It also has integration with ArduPilot and PX4, so that autonomous drone control algorithms that require good graphics and physics processing can be tested. Although it is more complex than in Gazebo to integrate *new* sensors and components into AirSim, there are already several configurable sensors and drone models that are available. For example, there are cameras and LIDAR sensors that are configurable to specific parameters, such as distortion parameters, resolution, field of view, etc. These make it possible not only to test navigation algorithms with realistic, dynamic data, but also make it possible to quickly generate large synthetic data sets for training deep learning networks. Several projects have succeeded in training such networks to, in the context of drones, detect and avoid crowds [4], detect animals [5], and race drones [6]. In addition to sensor data, AirSim can provide "segmentation masks" which label each pixel according to the object to which it belongs. This can help to create *semantic* labels for supervised learning without depending on expensive, error-prone human labeling, which can be used to train networks in object identification.

1.2.4 Fiducial Markers

Fiducial markers are patterns whose pose (position + orientation) can be recognized using only a monocular image and the intrinsic parameters of the camera that takes the picture. Figure 1.2 shows some common fiducial markers. They can also provide a small bit of data, such as an identifier. Fiducial markers are often attached to objects, such that they allow the pose of the object to be easily identified as well. A common application of this is to allow a robot to

locate and manipulate items. Although the position of a fiducial marker is conceptually easy to determine from a single monocular image, the orientation is subject to ambiguity because of the inherently finite pixel resolution. If the marker does not have sufficiently large pixel area, the sign of some of the components of the orientation can flip unpredictably, meaning that the orientation is not a reliable data point on which to make subsequent calculations. See Section 2.3 for an analysis of this, and for some possible solutions.

1.3 Related Work

1.3.1 Autonomous Drone Landing

Wynn [7] has developed a method for landing on a moving platform using fiducial markers to track the landing platform, with the initial aid of GPS. A larger marker allows recognition of the landing platform from long distances. A smaller marker of the same form is embedded inside the larger marker to allow for identification at close distances. After the landing platform is localized, different control states direct the drone's approach towards the marker - first causing the drone to approach quickly in the x and y dimensions, while maintaining a sufficient altitude above the marker (in the z dimension), and then gradually lowering to a small distance above the marker. At this point, the drone commits to a landing and lowers itself until detecting a successful landing, since the proximity of the camera to the landing pad means that the marker is no longer fully contained within the field of view of the camera, and thus can no longer be tracked. Other control states include switching from *patrol mode* to *tracking mode* once the relevant marker has been detected continuously for a small amount of time, and aborting a landing if the marker has not been detected for 2 seconds continuously. This method also takes into account the swaying of the landing platform itself, which is mounted on a barge.

Borowczyk et al. [8] have implemented a system allowing a DJI Matrice to land on a golf cart using GPS with wirelessly transmitted position. The drone uses a PN system for initial approach. This initial approach is carried out at a fixed altitude and a gimbal-mounted camera is used for initial detection of a fiducial marker mounted to the landing platform. A fixed, downward-facing camera then detects the visual fiducial marker and a PID controller manages close-range approach. A constant descent velocity is set during the final phase of landing. This method allows for successful landing on a platform moving at speeds of up to 50 km/h. Recommended future work includes using multiple fiducial markers of different sizes to identify the landing platform, as well as a single gimbal-mounted camera instead of the dual camera setup.

Falanga et al. [9] outline a method for landing a quadrotor running the PX4 autopilot software on a moving platform indoors. The landing platform is fitted with a specific marker made up of a cross and a circle. The drone uses 2 cameras, the first mounted straight down from the drone, and the second mounted at a 45 degree angle down and towards the front of the drone. The images from these cameras are used to solve a PnP problem which finds the relative pose of the landing platform's marker. A distance sensor helps to scale the vision-based pose estimation. The onboard computer determines optimal approach trajectories using this information. A Kalman filter makes the process robust to missed detections and helps to determine the velocity of the landing platform. Successful landings were reported with the landing platform moving up to 1.2 m/s.

Wubben et al. [10] use the typical setup of a hexacopter drone with a single camera in a fixed, downward facing orientation to identify a landing platform via 2 ArUco markers. A Pixhawk controls the drone using ArduPilot, and a Raspberry Pi handles image processing and fiducial identification. The method reports successful and accurate landings, but also occasional failures due to visual loss of the landing platform. This visual loss was caused by sudden gusts of wind which pushed the drone away from the landing platform and out of the fixed camera's view.

Pluckter et al. [11] have developed a method for precisely landing a drone in an *unstructured*

environment, which is to say an environment that has not been significantly artificially marked. On takeoff, the drone visually captures key points of interest in its environment. It then performs its mission and returns to a location above its takeoff location using GPS. Subsequent visual analysis of the surroundings and comparison of this information with the similar information captured at takeoff allow the drone to localize itself. This process is continued throughout the entire landing. This method is interesting and significantly different from the fiducial methods, but only allows the drone to land at the takeoff location.

Polvara et al. [12] introduce a method of training and testing DQN for landing drones in a simulated environment with simple outputs (left, right, forward, back, land), feeding the networks low-resolution images as input. Multiple networks were trained for specialized tasks, such as policy control, approach, and descent. The method performed with only slightly less accuracy than the conventional vision-based methods which use fiducial markers. However, the caveat is that the conventional methods can fail when the fiducial marker cannot be detected, whereas the method presented by Polvara et al still achieved a relatively high success rate. The method is, however, very simple and requires significant modification before it is viably applicable in a physical drone setting.

Cocchioni et. al [13] present a custom landing pad and marker to allow a quadcopter to land and recharge autonomously. The marker is a black circle with a smaller black triangle embedded in its white center, which allows pose estimation in 6 degrees of freedom. A downward-facing camera detects the marker and PID systems control the position of the drone to direct it onto the landing pad. A downward-facing ultrasonic distance sensor provides the altitude of the drone above the landing pad. The landing pad has 4 conical indentations in it - one for each of the drone's legs. The idea is that the drone can slide into the ideal position passively if each of the legs can fall into its hole on the initial landing. Then, charging terminals attached to each leg make contact with terminals in the landing pad, and charging can take place. The method is successful in 95% of cases, but involves a very specialized landing pad and custom fiducial marker, making distribution difficult.

1.3.2 Landing Site Evaluation

Desaraju et al. [14] discuss a method for exploration and evaluation of rooftop landing sites using motion stereo image processing. In this method, a downward-facing monocular camera regularly captures images of the terrain below. Disparity maps and feature matches between pairs of images provide extrinsic camera parameters which allow the system to mimic stereo processing and recreate a 3D model of the terrain below. The system then assigns confidence levels to each unit of area in the 3D model which correspond to that area's viability as a landing site, based on the radius of clear terrain, absence of obstacles, and certainty of the information describing the landing site. Trajectories are generated to direct the drone to each landing site and are ranked by several metrics including required distance and angular velocity (to minimize the variation in the camera's orientation). The method is successful in distinguishing the rooftop from the ground and identifying viable rooftop landing sites but is computationally expensive and therefore requires some adjustment to run in real time.

Patterson et al. [15] introduce a method of detecting safe landing zones (SLZs) in single RGB images (not time sequences of images). The algorithm first attempts to find large areas in images taken from a downward-facing camera using Canny edge detection, and selecting large regions with no edges. The assumption is that edges correspond to physical region boundaries which represent obstacles. The edges are then "dilated" in order to create a safety buffer around the potential landing zones. These potential landing zones receive a "safety score" based on minimization of the possibility of human casualty, minimization of the possibility of property damage, and maximization of the possibility of survival for the drone and payload. Terrain is classified manually on a data set of 490 images with 9 terrain classes. Distance to man-made structures is a second component of the safety score, which is determined from analysis of

existing maps. Lastly, a measure of roughness determined by the standard deviation of pixels from the mean of a safe landing zone, is used to determine which regions are smooth enough for landing. Fuzzy logic categorizes potential landing zones into "unsuitable," "risky," and "suitable." 100 aerial images were manually labeled and SLZs were identified within them. The algorithm correctly identified the regions 94% of the time with the knowledge of the existing maps, but reported about 10% false positives without this knowledge.

Whalley et al. [16] use LADAR to model terrain below a large model helicopter (182 pounds) in order to identify possible landing sites and trajectories. The required components weighed 6 pounds and consumed 2.9 A and 12 V. The system successfully avoided obstacles and selected landing sites, but no in-depth information on the landing site evaluation algorithm were given.

Johnson et al. [17] use structure from motion (SfM) to generate terrain maps of the area below a drone. Features selected from input images allow the ground station computer to calculate relative motion between sequential images. Image segmentation enforces that the features should be dispersed relatively evenly in the dimensions of the image. The dense structure of the terrain below creates an estimate of the terrain map, from which the computer can extract steep slopes and other hazards. This terrain is then labeled by the perceived hazard rating at each pixel, and safe landing sites are determined by minimal roughness and sufficient largeness. The algorithm runs in less than 1 second on a 400 MHz processor. A total of 4 safe landings were achieved.

Garg et al. [18] develop both a monocular method and stereo method to classify landing sites based on their viability, and to land at safe landing sites. The monocular method uses planar homography to generate a dense representation of terrain below a drone. It allows the drone to distinguish between rigid surfaces like ground and non-rigid surfaces like water. The stereo approach allows depth to be extracted from the images, which are also analyzed to identify roughness and slopes. The approaches disable landing if the environment detected is deemed unsuitable. The methods were not robust to surfaces that were covered in leaves because their changing nature led the methods to determine that they were not rigid.

Matunara and Scherer [19] present a method to identify safe landing sites for autonomous helicopters using a 3D convolutional neural network. The data set is partially synthetic, and partially hand-labeled. Their method is able to identify low-height vegetation as a safe landing site, whereas conventional methods of analysis would label them as too rough. In this case they are viewed as "porous" instead of "unsafe." The method is able to analyze successfully a cubic meter of ground in less than 5ms. It is tested on real and synthetic data, but not embedded onto a physical drone for real world landing tests.

1.3.3 Summary

Many methods exist for autonomous drone landing, with some depending on special infrastructure to mark a known landing pad, and with others analyzing terrain to identify previously unknown landing sites. Infrastructure at known landing pads can be active (e.g. radio/IR beacons, cameras, etc.) or passive (e.g. visual fiducial markers). Methods for identifying previously unknown landing sites depend on sensor-based terrain analysis or visual matching and typically use RGB, RGBD, or LIDAR/LADAR to collect information about the ground below. Methods of analyzing the sensor data range from very finely human-tuned methods of edge detection/dilation in images, to black-box deep learning methods. Additionally, older methods tend to use only real-world data, while newer methods begin to leverage quickly-generated, cheap, synthetic data from simulators. Some methods can use single images, while others require images from multiple locations in order to deduce the underlying 3D structure of the terrain. In almost all cases, the sensors are fixed rigidly to the drone, instead of being mounted on a gimbal. It is common to approach the general location of a landing site using GPS, and then to carry out the landing using some augmented sensing. Processor-heavy methods cannot necessarily run in real time on a drone, and must either be run offline as a proof of concept, or must be offloaded to a sophisticated ground control station for real time processing on bulky hardware.

Chapter 2

Completed and Ongoing Projects

2.1 Master Thesis Proof of Concept Attempt

In my master thesis [20], I developed a method for fiducial precision landing with a gimbal-mounted camera. The algorithm involves identifying fiducial markers through image analysis, tracking the markers via a gimbal-mounted camera, calculating position targets via coordinate system transforms in order to direct the drone towards the landing pad, and communicating those position targets to the flight control software for approach. I tested this algorithm in Gazebo simulator only, and the natural next step was to test it on a physical platform. We have therefore built 2 Tarot 680 hexacopters, which provides a good thrust-to-weight ratio, and space for mounting multiple computational components. A combination of Raspberry Pi 3B+ and Navio2 [21] shield serve as a flight controller that can communicate with a companion board. The companion boards (a Google Coral Dev board and an NVIDIA Jetson Nano) communicate via Ethernet over USB to the flight controller and perform all heavy computations involving image analysis, coordinate system transforms, PID control, and position target generation. Figure 2.1 outlines the hardware setup.

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 fully assembled Jetson hexacopter are shown in Figure 2.2. (The Coral drone looks and performs similarly.) The algorithm is implemented as a set of ROS modules. A `gscam` node retrieves camera input and makes it available as a ROS topic, after which a `whycon_ros` node analyzes camera images to detect WhyCon/WhyCode markers and determine their pose. A `gimbal_controller` node reads the WhyCode detections, and passes the positions of the markers to 2 PID controllers, which aim the gimbal tilt and pan in order to center the marker in the camera frame. Finally, it passes the PID outputs as PWM signals to the gimbal. A `landing_controller` node reads the WhyCode detections, performs coordinate system transforms to generate position targets, and forwards them to the autopilot software.

2.1.1 Results

The drones fly with good stability even in high Icelandic 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 Figures 2.2c-2.2d, with the position of the marker in the camera frame in the given axis, and with

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

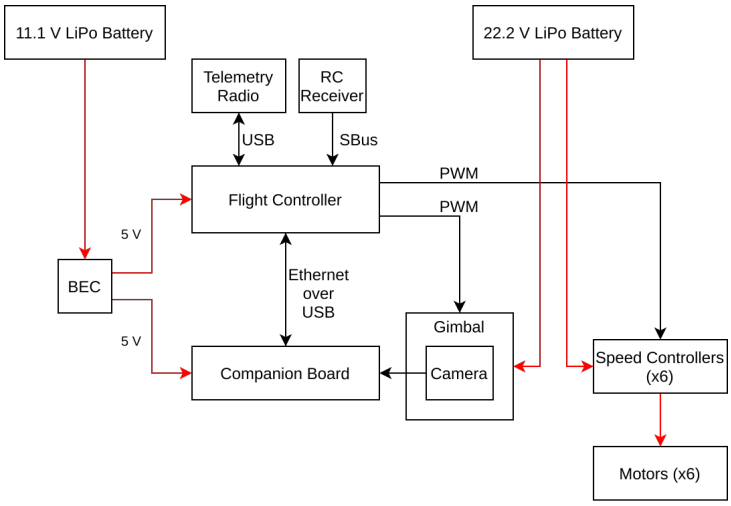
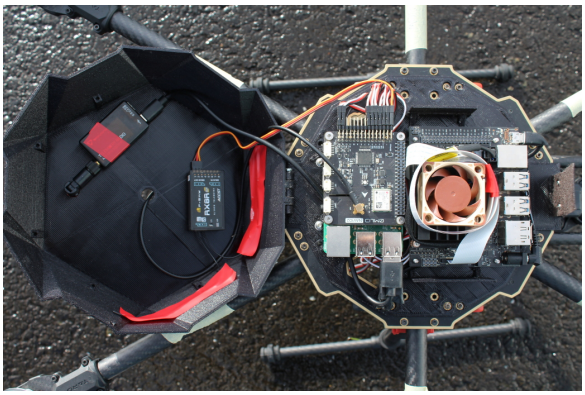
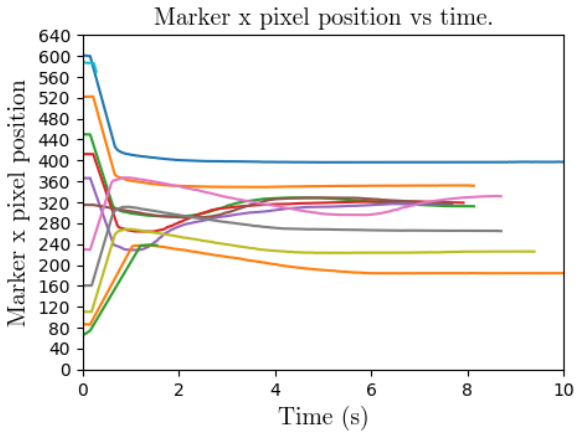


Figure 2.1: Hardware Setup

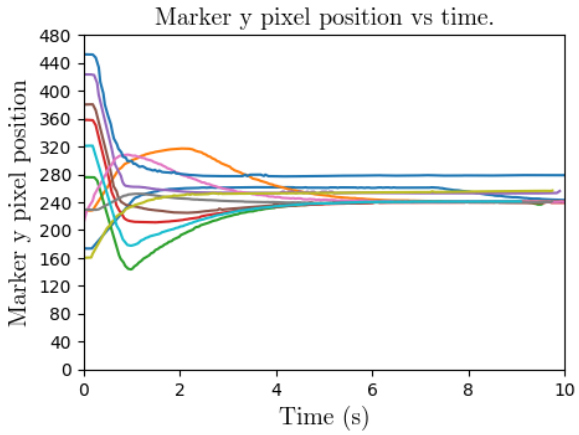
(a) The Jetson drone.



(b) The Jetson drone's electronics compartment.



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



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

Figure 2.2: Performance of aiming the gimbal.

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 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.

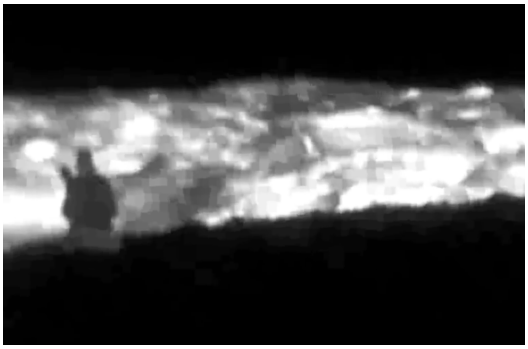
Second, there is a fundamental challenge with fiducial markers which comes as a result of the limitations of embedding 2-dimensional shapes into 3-dimensional spaces. While the position of the marker in the camera frame can be detected unambiguously, the orientation of the marker cannot. Especially when 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 (roll, pitch, yaw)$=(1°, 0, 0)$ versus $(-1°, 0, 0)$. 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 Section 2.3.

2.2 Heavy Lift Drone with Infrared Camera

Remote sensing is perhaps the most common application of drones in many fields, and to this end, geologist Christopher Hamilton brought a heavy lift drone to Iceland for collecting infrared video at the Geldingadalur lava field. Along with Baldur Björnsson, I assembled and tested the drone, making modifications to attach a large gimbal and FLIR (infrared camera). The drone uses a WuKong (DJI) industrial flight controller, six 2-horsepower motors, a 22.2 V, 22000 mAh LiPo battery, and has a diameter of 1.6 m. It sits on a gimbal with the mounted IR camera. The gimbal uses a 14.8 V battery to control the orientation of the camera, and receives RC signals from a secondary, dedicated receiver. (One operator controls the drone while the other controls the gimbal.) The camera does not have stringent power requirements, but in this setup the only option for powering it was to provide a 48 V PoE (Power over Ethernet) source which required extra batteries. A Raspberry Pi 3 connects to the camera over ethernet and streams the video into a file, using a special version of GStreamer developed for the FLIR.

Due to time constraints (about 48 hours from receipt of the gimbal to flying over the lava field), much of the drone has been quickly improvised. Many of these things must be rectified in the near future for further testing. For example, the mounting plate between the gimbal and the drone is completely inadequate (too flexible, not enough points of contact, etc.), and must be replaced with a more solid/purpose-made mounting plate. The 48 V requirement of the camera required an additional three 4S battery packs wired in series and injected into the ethernet, and this can be replaced with a transformer that can generate 48 V from the same 14 V power source that the gimbal uses. The IR video is not streamed to the operators during flight, so the operators must guess what the camera is seeing, and attempt to obtain good footage. Still, even with all these disadvantages, the drone flew successfully two times over the lava field at

(a) Heavy lift drone in flight.



(b) IR still-frame from heavy lift drone.

Geldingadalur and obtained aerial IR footage of the lava, as can be seen in Figures 2.3a and 2.3b. Additionally, the BBC program documenting the event[2], and the raw IR footage[3] can be found online.

2.3 Fiducial System Modifications

After the initial proof of concept attempt at fiducial-based landing with a gimbal-mounted camera (shown in Section 2.1), we have decided to make some modifications to the tested fiducial systems to make a success more likely. To restate, the goal is to successfully execute precision landings using a fiducial marker to designate a landing site, and the key difference between this work and previous work is the gimbal-mounted camera for marker tracking. Most previous projects use a fixed, downward facing camera and have reported loss of sight of the landing pad due to wind and thrust vectoring by the drone. Marker tracking makes the detection of the landing pad more robust to the drone's movements, but also complicates the pose estimation. Since the orientation of the camera cannot be assumed to be straight down, we must transform the pose of the detected fiducial marker in order to generate a position target. However, due to the planar pose estimation problem, the detected orientation is unreliable in monocular detection of planar fiducial markers, as shown in Figure 2.4. To mitigate the issue of orientation ambiguity, we have created 3 new variants of the April Tag and WhyCode fiducial systems, and compared them to 2 default variants, as outlined below. This is the content of a paper we have submitted to IROS 2022.

2.3.1 Methods

WhyCode Orig: Tooth Arclength Sampling

We use a version of WhyCode created by Ulrich [22] as a baseline for testing. The method samples the ellipse that goes through the centers of the "teeth" forming the marker's ID (i.e. the yellow and green ellipses in Figure 2.5a), to determine the marker's orientation. Of the 2 possible candidate solutions that are implied by the detected semi-axes of the outer black ellipse, the detector chooses the one that has lower variance on the number of sample points per tooth. This works because the candidate solutions predict this ellipse to be in slightly different places, and 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, such that the sampling ellipse does not line up well with the marker, and the variance is higher. We use WhyCode markers with 8-bit IDs, so that there are several sample points and a meaningful variance.

[2]BBC program documenting the drone flight: `https://www.youtube.com/watch?v=6SIgFPhhRPE&t=1190s`
[3]Raw IR footage: `https://vimeo.com/580302507`

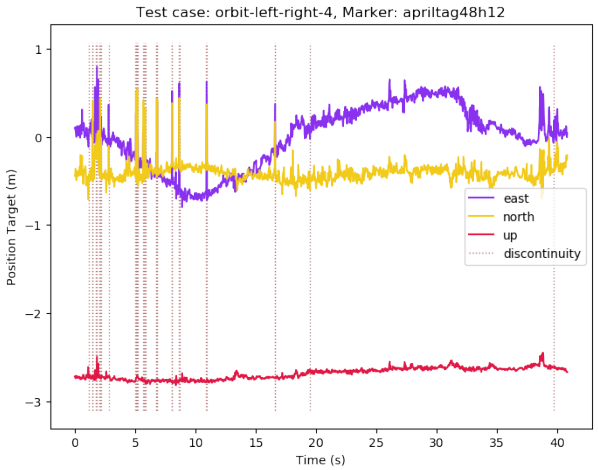
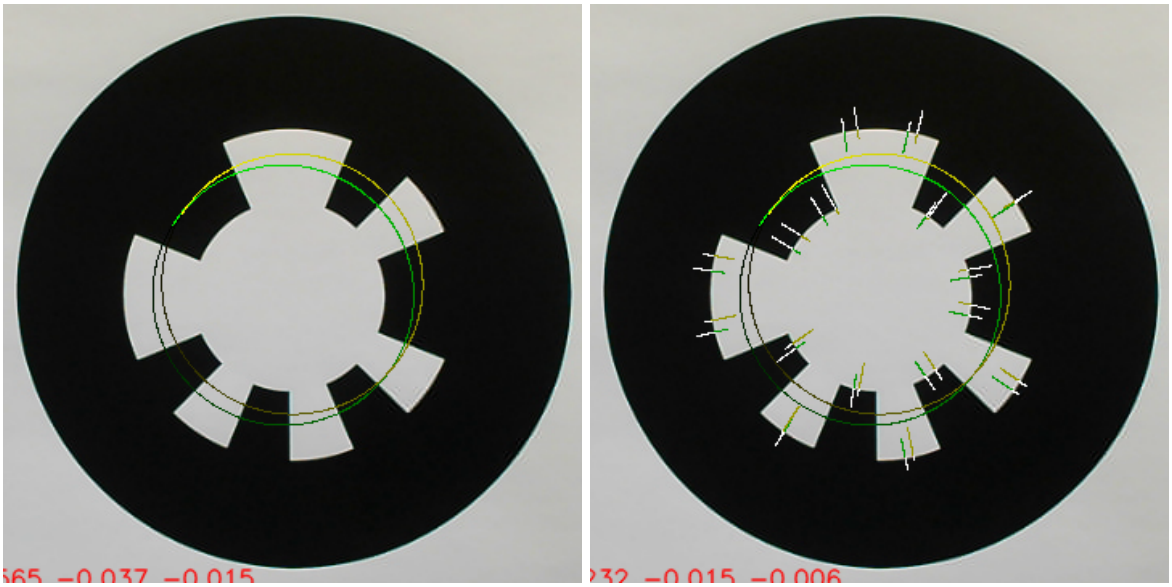


Figure 2.4: Demonstration of discontinuities in position targets as a result of orientation ambiguity.



(a) WhyCode Orig method.

(b) WhyCode Ellipse method.

WhyCode Ellipse: Radial Tooth Sampling

The first method that we have created for reducing orientation ambiguity is the `ellipse_sampling` branch of [23]. The method determines the marker ID and the two candidate solutions for the orientation as in WhyCode Orig, after which it identifies the lines that goes from the center of the white region through the center of each tooth. It then samples the input image on these lines, as illustrated by the radial sample lines in Figure 2.5b. It expects a white-to-black transition at the predicted edge of each tooth, and the sampling line is centered on this edge. The true edge is determined during sampling, and its value is recorded as a percentage of the length of the line segment, oriented such that 0 corresponds to the centermost end of the line segment, and 1 corresponds to its outermost end. The detector chooses the solution that minimizes the variance of the location of the true edge over the sample lines.

WhyCode Multi: Planar Regression from Multiple Markers

The second method that we have created for reducing orientation ambiguity (the `multi` branch of [23]) works under the assumption that all markers are co-planar. For each input image, the WhyCode algorithm identifies all markers and then finds the normal vector to the plane implied by the markers' positions, after which it can calculate the pitch and roll components of the bundle's orientation. The position of the bundle is defined to be the mean of the positions of its constituent markers, and its yaw is that of any constituent markers, with the assumption they are all the same. The detector then calculates all additional attributes for the bundle as if it were a marker. This system is tested on an arrangement of 3 co-planar, 8-bit WhyCode markers.

April Tag 48h12: Large, Embeddable April Tag Family

April Tag provides a default family 48h12, shown in Figure 1.2c, for which the 4 center squares are undefined, 20 squares provide a white border, 28 squares provide a black border, and 48 squares provide data bits, giving a total of 96 defined squares. The undefined center provides a space to embed a smaller marker, which is useful in the last stages of a drone landing scenario, where the camera is too close to the landing pad to see the larger markers. We test this family as a baseline for the performance of April Tag.

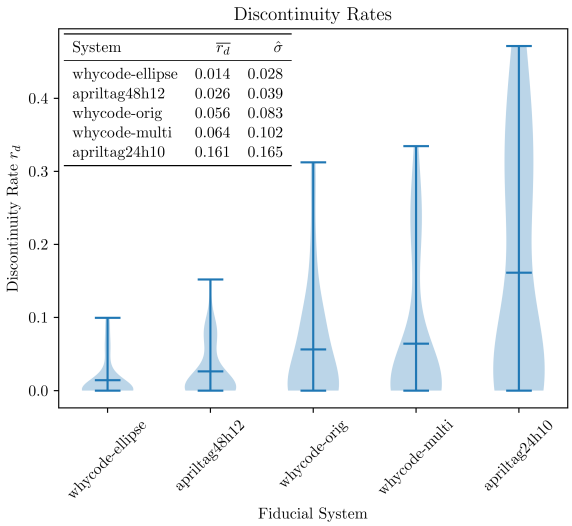
April Tag 24h10: Small, Embeddable April Tag Family

We have created an April Tag 24h10 family (shown in Figure 1.2b) in response to our We have created an April Tag 24h10 family (shown in Figure 1.2b) in response to our initial experiments, which showed that April Tag 48h12 has a slower detection rate than WhyCode on embedded hardware. We maintain marker embeddability while decreasing the size of the marker definition, by reducing the size of the undefined center to region to 1 square, and adjusting the surrounding regions accordingly: 8 squares for a white border, 16 squares for a black border, and 24 outer data squares. This gives a total of 48 defined squares(compared to the 96 squares of April Tag 48h12). We test April Tag 24h10 to see if it can offer an increase in detection rate with respect to April Tag 48h12.

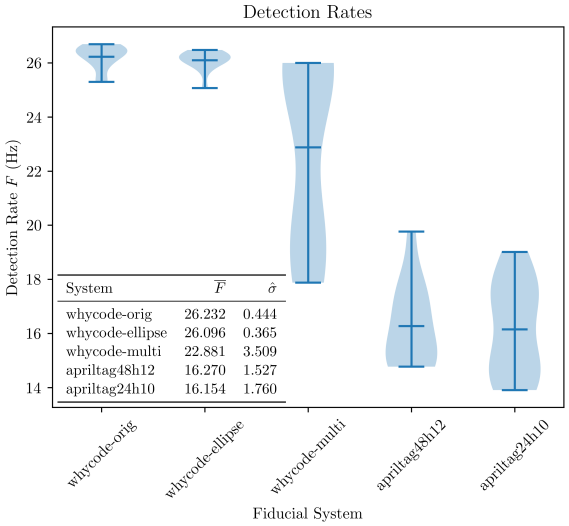
2.3.2 Performance

Discontinuity Rates:

We have achieved a lower rate of discontinuities in our WhyCode Ellipse system than the WhyCode Orig system. Figure 2.6a shows the discontinuity rates of each system, and Figure 2.6b shows the detection rates. Formal statistical tests (included in the paper) rank the systems against one another.



(a) Discontinuity rates of the detected systems.



(b) Detection rates of the detected systems.

2.4 Autonomous Landing Proof of Concept

The last part of the initial phase of this research has been to successfully develop a proof of concept of the initial autonomous landing algorithm on a physical drone (developed in the master thesis [20]), which was also the goal of the first part (in Section 2.1). The crucial difference is that we have used a DJI Spark as the drone platform, which is stable enough to remain nearly still without human intervention, even in a GPS-denied environment. However, even though the proof of concept is successful, its implementation is significantly constrained because of its basis on the DJI Mobile SDK, as explained below. The DJI Mobile SDK is necessary because the DJI Spark has no companion board, but anticipated future drone systems will have a more ideal setup using the DJI Onboard SDK. The following content will be some of the content of our next paper.

2.4.1 System Architecture

The system is outlined in Figure 2.7. All data is transmitted from the Spark to its controller via a wireless connection. The controller interfaces with an Android tablet to expose the drone's data and API. The app must authenticate with DJI using a user-specific API key before it can communicate with the drone, which drives home the proprietary, block-box nature of the system. Importantly, after the app has successfully authenticated once, it can re-authenticate without an internet connection for some period of time, meaning that an internet connection is not required in the field. The Android app then receives data about the condition of the drone, including all flight data (e.g. attitude, velocities, flight modes, etc.), and the video stream from the camera. The app periodically decodes bitmap frames from the video stream, compresses them into black-and-white .webp images, and transmits them over a WiFi connection to a companion board. The companion board then carries detects April Tags in the video frames, determines what action to take according to the control policy, instantiates a command message, and sends that command back to the app. The app then reads the command, creates a virtual stick input via the DJI SDK, updates the GUI to visually show the command, and sends the command to the drone. The drone then carries out the command.

The virtual stick inputs simulate input as if it were coming from the controller. Each stick command can be in the interval $[-1, 1]$. The drone can interpret those stick commands as setpoints either for position or velocity. Since we are abstracting from position, we only use the velocity mode. Therefore, a command of 1 on the pitch stick means that the drone should move

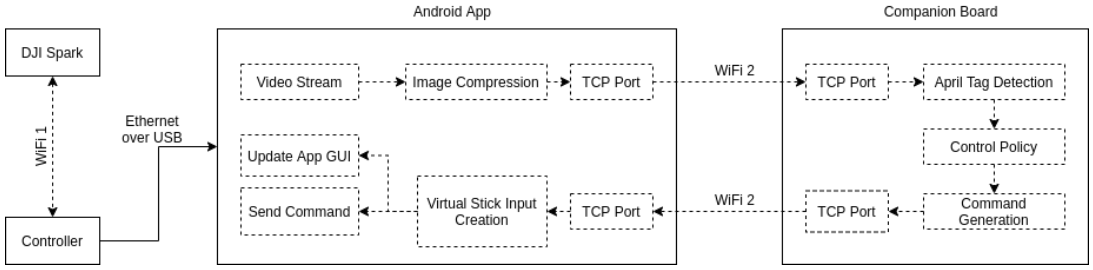


Figure 2.7: Data flow for the Spark autonomous landing system.

forward at its maximum velocity.

2.4.2 Results

The system is able to reliably land the drone without human intervention.[4] So far, we have tested it with April Tag 48h12 and April Tag 24h12 landing pads, and we will continue testing the modified WhyCode systems outlined in Section 2.3. As anticipated, the April Tag 24h12 tags are subject to common instances of orientation ambiguity, whereas the April Tag 48h12 tags are mostly free of orientation ambiguity. In the demonstration video, this orientation ambiguity appears as jittery control inputs from the companion board, as well as (sometimes) jittery behavior from the drone. Fortunately — and surprisingly — this behavior appears to be non-destructive in this case.

The Android app has 5 sliders to control the drone: pitch (translated to forward/backward speed), roll (translated to left/right speed), yaw (translated to yaw speed, i.e. rotational speed in the Z axis), throttle (translated to vertical speed), gimbal tilt (translated to gimbal tilt speed). It also has buttons to enable or disable virtual control in the flight controller, to command the drone to takeoff, and to command the drone to land (blindly). The proposed research will avoid the Mobile SDK and will therefore avoid the need for a custom app. However, subsequent app development can be the subject of a future project if necessary.

2.4.3 Limitations and Implementation Quirks

The Mobile SDK architecture is not well-suited to this application for a number of reasons. First, there is only one wired connection on the tablet, meaning that at least one connection must be wireless - either the connection to the controller, or the connection to the companion board. The tablet is not fast enough to do April Tag image analysis on its own, and there would be significantly increased development efforts required to install April Tag on the tablet alone (so the companion board is required). Moreover, connecting the tablet to the controller via its WiFi interface incurs prohibitive latency as a result of the large amount of processing required. It is also prohibitively slow to connect two clients over the controller's WiFi simultaneously. Second, the frames of the video stream are available in bitmap format, and each frame is prohibitively large/slow to transmit over a wireless connection to the companion board. In order to transmit "video" in "real time," the app converts bitmap frames to black and white, and then compresses them into .webp format with 10% quality. This allows a framerate of about 6 Hz, with a delay in each frame of about 0.25 seconds, to be delivered to the companion board. Amazingly, this is enough to land the drone with reasonable accuracy. Third, the Spark requires a blind commitment in the last stage of landing - where the flight controller forces the gimbal to point directly forward, apparently to protect it from the ground since it has very close clearance. So, although the drone responds to commands during this phase of landing, the companion board cannot see the landing pad and therefore cannot generate commands for last minute correction.

[4] Autonomous landing demonstration video: `https://vimeo.com/664863992`

2.4.4 Conclusion

This experiment shows that the algorithm developed in the master thesis [20] works in the real world, and not just in simulation, even with low-quality image streams and high network latency. The main problem in our initial experiments (see Section 2.1) seems to have been the drone platforms themselves. The several sensors (apparently undisclosed by DJI) on the Spark which provide accurate velocity/position data based on the appearance of the external environment allow the drone to accurately control its position in GPS-denied environments. This makes the Spark much more stable as a platform than the hexacopters that we have made. However, we will not continue development with the Spark, since it is essentially a closed system and provides no room for adding additional components. Instead, we will add additional sensors to our drone platforms to make them less reliant on GPS.

Chapter 3

Research Plan

3.1 Overview

Here we outline the plan for using the knowledge and physical drone platforms created until now to create a method for landing in unstructured environments. The critical difference is that we will avoid making major assumptions about the landing area, e.g. that it uses a specific type of fiducial marker. The goal is to create a lightweight method for analyzing terrain in real time, identifying safe landing sites, and navigating to those landing sites without using some exterior information such as GPS. We keep in mind not only the state of the art in drone sensing and automation, but also the sensors, drone platforms, flight controllers, and computational hardware that we have immediately available. The research plan consists of 5 general steps, shown below, with milestones listed in Table 3.1:

1. Generation of a data set of realistic drone sensor data for testing terrain analysis methods. We will programmatically create a large, synthetic data set using AirSim for offline testing of terrain analysis methods. The data will be RGBD images and LIDAR point clouds (tagged with IMU data describing the orientation of the sensor, and distortion parameters as applicable) that are labeled according to which regions are considered safe landing sites. We will also collect real world sensor data which will serve as a partial validation test set.

2. Creation of multiple terrain analysis methods, and performance comparison on the data set. We will create several methods for analyzing terrain in the format of the data set, and we will compare their performance in offline evaluations. Generally, the methods will preprocess the data (rectification, resizing, and transformation of the data according to the IMU tags), then analyze the data for safety of landing locations (according to flatness, smoothness, largeness of contiguous safe regions, and predicted material type), and finally output a mask describing the predicted "safety metrics" of the input data.

3. Performance analysis of any viable methods in simulated autonomous landing scenarios. We will create a post-processing wrapper to detect and track safe landing regions from the safety masks, and then translate that information into MAVLink commands for the flight controller. We will test this in AirSim with ArduPilot and/or PX4 plugins according to ease of implementation.

4. Enhancement/adaptation of current drone platforms for real world testing. We will add more sensors to our Tarot 680 platforms to try to make them stable in autonomous flight without GPS. We will also package our sensors into custom waterproof, protective cases and mount them onto a stabilized gimbal.

5. Performance analysis of any viable methods on physical drone platforms. We will test the methods offline again, to evaluate their potential runtime framerate and energy consumption on our Google Coral and Jetson Nano hardware platforms. We will test any sufficiently power-efficient and fast methods in real autonomous landing scenarios.

Task	Milestone	Work Week(s)
Data Set Generation	Creation of a script for synthetic data set generation from AirSim. The script should quickly generate RGBD images, LIDAR point clouds, and segmentation masks, then tag them with IMU data describing the orientation of the sensor, and save them to files. This task is nearly complete.	1
Data Set Generation	Creation of a script to label the synthetic data according to top-down methods of signal processing (e.g. low gradient, edge detection/dilation), and leveraging the segmentation masks.	2-5
Terrain Classifier Creation	Creation of 5-10 methods (human-specified, deep learning, and combination of both) for predicting landing site safety according to the data set.	4-24
Data Set Generation	Collection of real-world IMU-tagged RGBD and LIDAR data for validation data set. Can be done in Iceland in the summer months. Requires mounting the RealSense sensors to one of the existing drones with a companion board and flying several missions.	8-14
Terrain Classifier Creation	Performance comparison in terms of safety prediction accuracy. Test on synthetic data, validate on real world data.	20-28
Testing in Simulation	Integrate pre-/post-processing wrappers, safe region tracking, and autopilot plugins in AirSim.	20-28
Testing in Simulation	Test a series of simulated landing scenarios in AirSim. Performance comparison based on landing success/failure.	28-32
Publication	Write a paper detailing the findings from the previous steps, from data set generation to simulated landings.	33-39
Real World Tests	Power, runtime rate comparisons among viable methods when executing on real world data on Jetson Nano and Google Coral.	39-43
Real World Tests	Upgrading drones with optical flow sensors, laser rangefinders, autonomous flight tests with ArduPilot/PX4 in Iceland.	43-49
Real World Tests	Real world landing tests with a physical drone.	50-58
Publication	Write a paper detailing the findings from the real world tests.	54-61
Thesis	Summarize findings from start to finish.	62-85

Table 3.1: Milestones for each task. Time starts from thesis proposal defense, and overlapping tasks are done in parallel. Work times are estimates only.

3.2 Data Set Generation

We will create a synthetic data set of LIDAR point clouds and RGBD images, and corresponding segmentation masks in AirSim, and we will collect similar data from the real world on a smaller scale (without segmentation masks). The labels for the data set will be "safety masks" which label each pixel The synthetic data will be labeled automatically using slow, detailed terrain analysis methods, and leveraging the segmentation masks in order to have a notion of *material*. This is to say that, even though calm water may be flat (and therefore safe, according to its topology), it is obviously not a safe landing site and can be ruled out just by being water. Conversely, an area of grass that may appear rough is not necessarily a dangerous landing site. By using the segmentation masks in creating the labels, we hope to discriminate between potential landing sites on another level than simply topology. However, topography will be the principal way of determining the safety of a landing site, and the most important aspects will be to minimal roughness, minimal slant, and maximal size. So the ideal landing site will be a large, smooth, level area having the appearance of a "safe" material.

We will use a consistent pipeline for processing and labeling the images, which will be similar to traditional image analysis. First, we will rectify the depth images before processing them, to remove inevitable sensor distortion. In reality, the distortion will not be entirely eliminated, but will be mitigated. We will record the orientation of the sensor when the image is collected, and then transform the depth image so that it is oriented vertically down. We will process the rectified image to generate at least two safety masks - one for topological safety and the other for material safety. Then we will combine these intermediate safety masks to create an overall mask, which will serve as the label. The exact sizes of the training and testing data sets have yet to be determined, and as such we will rather focus on creating a script to automatically generate data sets of specified sizes. The data sets will be taken from multiple simulated environments to try to give an unbiased sense of what possible good landing sites look like.

The real world data will serve as a validation set, which will give a notion of how well the synthetic data correspond to the real world. We will the label the real world data in a similar way to the synthetic data, with the exception that we will not have corresponding segmentation masks. Additionally, it is too time-consuming to create both the labels and segmentation masks by hand. As a result, we will have to take special care in evaluating the performance of the terrain classifiers on the real world data set. It is likely that we will not be able to simply consider how well the predicted mask matches the label. We anticipate that the terrain classifiers may have to be evaluated on real world data purely by the landing locations they select, in order to be feasible in terms of time and logistics. In this case, a human who is familiar with the surveyed area can mark landing locations as feasible or infeasible by hand.

3.3 Terrain Classifier Creation

We will create multiple methods and evaluate them on their ability to predict the safety mask of the landing sites in the synthetic data sets. We will focus first on using U-nets to map from the input LIDAR or RGBD image to the output mask, as they have been proven to be good classifiers in many different scenarios, from terrain analysis, to tissue analysis, to road identification, etc. We anticipate that the main challenge in this domain will not be to train the networks, but rather to keep them small enough so that they can execute quickly on embedded hardware. Nevertheless, we will start by focusing strictly on maximizing the ability of the networks to accurately predict the safety masks.

3.3.1 Pre-Processing

An unavoidable step in this process is preprocessing the input data. In the case of RGBD images, we should rectify input to remove sensor distortion. This is helpful for understanding what the

data truly represents, and also removes sensor biases, so that the terrain analysis methods do not overfit to data from a particular sensor. We will also transform input according to IMU tags in order to properly identify slants. One option for pre-processing in the case of LIDAR point clouds is to convert them to depth images, and we anticipate that it will be generally less computationally expensive to process depth images instead of point clouds. In either case, the input must be normalized with respect to sensor calibration data - sensors tend to auto-adjust their scales in order to clarify differences in values at each point. This occurs when RGB cameras auto-adjust brightness in order to make a clearer picture, and also when an RGBD camera detects the average range of detected depths, and adjusts its format accordingly. Finally, downsizing input images and point clouds is a common method of computational speedup that will most likely be necessary, given that the target hardware platforms are limited, embedded boards.

3.3.2 Topographical Analysis

The most important aspect of a safe landing site is its topography. We want to identify regions of the pre-processed input images that are sufficiently flat and large. Signal processing methods, such as analysis of the image's gradient, and even edge detection algorithms (e.g. Canny, etc.) can be used to detect contiguous flat regions. Deep learning methods such as pooling and 2D convolutions can accomplish similar tasks. The most relevant data channel in an RGBD image for topographical analysis is the depth channel, but other channels could still offer useful information. While it is possible to "hand-make" a solution for finding a landing region that is topographical viable, it may also be useful to train deep learning methods to do this task. They will likely have inherent biases that are different from the hand-made solutions, which will give a wider view of the solution space in the worst case, and which will yield better results in the best case. Moreover, a deep learning method will be more suited to use the embedded TPU and GPU on the target companion boards, and therefore may run faster.

3.3.3 Semantic Segmentation

One key application of image processing with deep learning is to segment images into the things they represent. This would offer a huge benefit in terms of autonomous landing, as the system could recognize e.g. water as unsafe, and leaves as safe, regardless of their topography. In general, it is expensive to label real world data sets for semantic segmentation because it must be done by hand. However, the segmentation masks in AirSim allow quick generation of large, semantically labeled data sets. Therefore, we will supplement the topographical analysis with semantic segmentation in the hopes of getting better results.

We will use a conventional CNN, U-net, residual U-net, for basic semantic segmentation of the pre-processed input images. CNNs can semantically segment scenes based on kernels, U-nets use both convolutions and an encoder-decoder architecture, and residual U-nets All of these architectures are effective for semantic image segmentation, but in the context of embedded drone landing, we have to strike a balance between the more complex networks that offer more intricate segmentation, and the less complex networks that are likely to run faster. The viability of each of the approaches must be empirically tested on the target platforms.

As an example of how this will work, we take an example from my project in RU's Deep Learning course, wherein my group member and I used a Residual U-net to classify defects in manufactured steel. The data is a set of 17,000 images that are labeled by hand, according to which of 4 defects they contained. Figure 3.1 shows an example of the input data from this project, where the yellow and blue outlines show the locations of different types of defect areas. An example output mask can be seen in Figure 3.1. This project also gave some insight into accuracy metrics that can help bias the output for/against false positives/negatives. For example, in the case of the steel defect recognition, we used a Tversky metric to reduce the amount of false negative defect classifications because a second step of the process is for a human

to manually examine the defect-classified steel before discarding it. In this context it is better to make the network more "cautious," in the hopes that alerting the human to some false defects will mean that the network never incorrectly approves defected steel for sale. Conversely, in the case of autonomous drones, we would like the networks to be conservative in their identification of safe landing sites, so that they never direct the drone to land in unsafe landing sites, even though they may reject some landing sites that are actually safe. This is purely to give an intuition of the form of the problem we will use the classifiers to solve.

3.3.4 Target Output

The results of the topographical analysis and semantic segmentation will pass to a hand-made wrapper or subsequent network layer that can combine them in order to create a mask of the predicted safety metrics of the terrain below the drone. This safety mask will likely depend mostly on the topographical analysis, with the semantic segmentation giving the ability to "veto" potentially smooth but dangerous regions like water, and classify as safe somewhat rough but potentially safe regions like leaves.

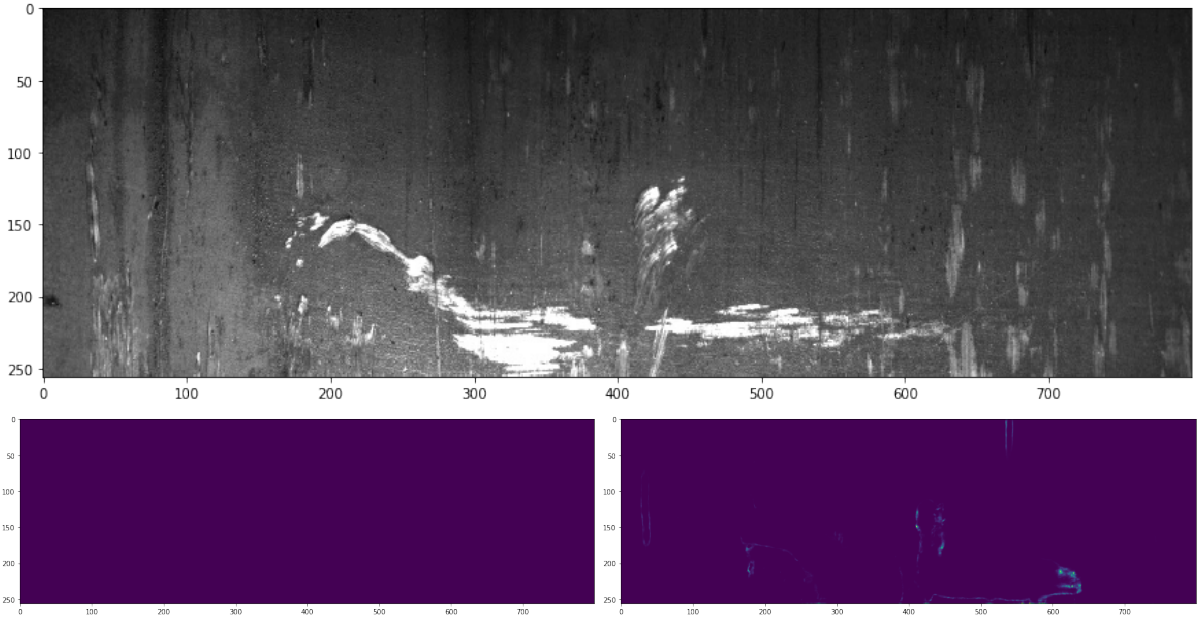
3.4 Testing in Simulation

As the terrain classifiers will output a mask, we will have to post-process the data to make it compatible with an autonomous drone system. We will therefore create an output wrapper to find contiguous regions of high predicted safety (i.e. estimated safety above some experimentally-determined threshold), and to track these regions over time. We will then select a region for landing, and create commands to direct the drone towards that region in a similar manner to the method outlined in Section 2.4. We will make no explicit estimate of the region's position relative to the drone in terms of real world distance, as this is unnecessary and requires more sophisticated wrappers which take into account the distortion parameters of the camera in order to associate pixels with physical points (as in the case of fiducial markers). It is simpler and more generalizable to different platforms to choose a safe landing region, calculate its centroid, normalize its coordinates to the interval $[-1, 1]$ based on the pixel dimensions of the camera/mask, and then convert these values to Virtual Stick inputs (in the case of a DJI flight controller), or to velocity setpoints (in the case of ArduPilot or PX4 in simulation).

After creating the output wrapper, we will move to testing the networks in simulated environments in AirSim. We will use multiple environments - both those from which the training data originates, and also unfamiliar environments - to get an idea of the generalizability of the networks. These networks will be evaluated on the number of successful landings they execute in a number of different scenarios.

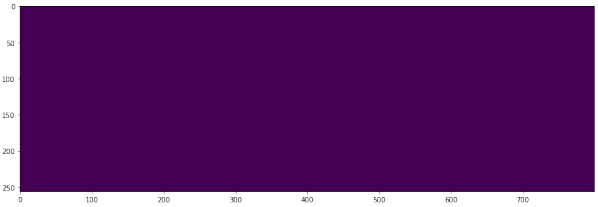
3.5 Testing on Physical Hardware

The first phase of this testing is to port the high-performing networks from testing in simulation to the real hardware. So far, we are targeting our existing Google Coral Dev Board, which has an embedded TPU, and an NVIDIA Jetson Nano, which has an embedded GPU. We will implement the pre-/post-processing wrappers for rectification and command generation in the same way as in the testing in simulation, and instantiate the trained networks on the embedded hardware. We can then pipe data from the synthetic or real-world data sets into the physical system to collect empirical results for the framerate achieved and power/computational requirements in making predictions using the networks. Piping data to the system will allow us to test the hardware/software in isolation, such that the effect of installing it onto our drone systems will be predictable. This is an area where we may have to carefully re-consider purchasing new embedded computational hardware, if none of the networks run fast enough, or if they all take

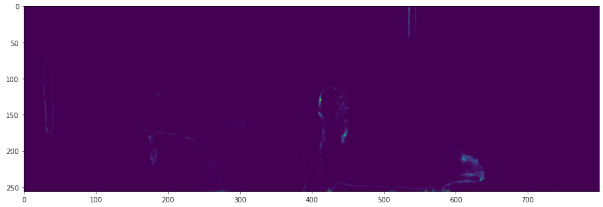


(a) Defect 1 Mask

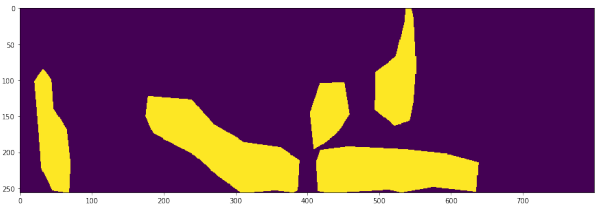
(b) Defect 1 Prediction



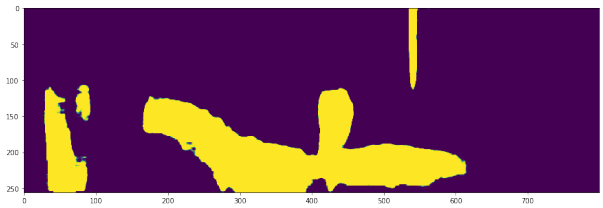
(c) Defect 2 Mask



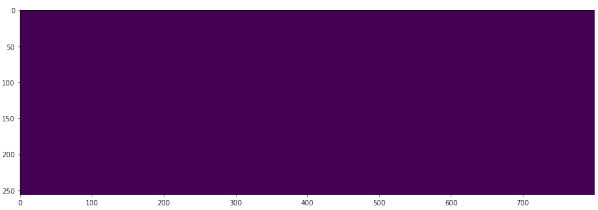
(d) Defect 2 Prediction



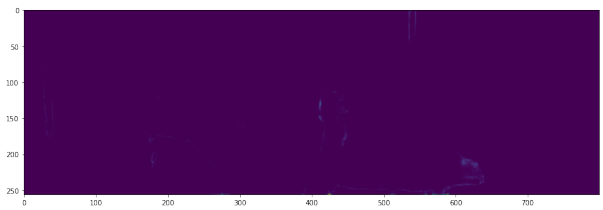
(e) Defect 3 Mask



(f) Defect 3 Prediction



(g) Defect 4 Mask



(h) Defect 4 Prediction

Figure 3.1: A visualization of the network's performance in predicting defects.

too much power. Aside from the Google Coral and Jetson Nano, we will also consider some FPGA in the future.

If any of the combinations of networks and hardware work with enough accuracy, at a high enough framerate, and by consuming little enough power, we will install this hardware onto a physical drone and test the system in a real autonomous landing scenario.

3.6 Drone Upgrades

The drone systems will need some modifications in order to accommodate the proposed unstructured landing systems in Iceland, with the principal modification being the addition of secondary positional and velocity sensors. This is motivated by the performance of DJI drones in GPS-denied environments and in environments with low GPS accuracy, i.e. most of Iceland. We will install and test optical flow sensors and LIDAR sensors, which give the drones the ability to accurately estimate their velocity and altitude, respectively, without GPS. This should give the drones a better ability to maintain their position without a human operator, and therefore should allow them to operate in *autonomous* mode. Additionally, we will test a Pixhawk Cube Orange flight controller and Here 3 GPS (RTK compatible) on one of the drone systems. This will allow a more minimal setup inside the drone, since the flight controller will be self-contained, and not a combination of Raspberry Pi and Navio2. The GPS itself should be more accurate, and can be supplemented with an RTK base station if necessary.

We have several sensors for which we will design and build waterproof, protective cases, and adapters such that they can be mounted in a gimbal:

- an Intel RealSense D435 RGBD camera
- an Intel RealSense D455 RGBD camera with integrated IMU
- an Intel RealSense L515 LIDAR module with integrated IMU
- a Texas Instruments IWR6843 60 GHz RADAR module

These sensors can provide real world analogues to the RGBD images and point clouds in the synthetic data set. There will be minor additional overhead in interfacing these sensors with the companion boards, e.g. compiling their libraries and figuring out their dependencies with likely limited documentation.

3.7 Risk Assessment

There is little risk in the synthetic data set generation, as we have already generated similar (albeit more basic) data sets in the past, and running AirSim does not pose any logistical risk. There is slight risk in collecting the real world data sets, from the ever-present risk of damage to equipment from crashes, to the difficulty in collecting data with high enough resolution and operating within the motion/distance constraints of the sensors. To mitigate this, we will run several missions to collect real world data over a long period of time, so that our method of collecting and tagging the data can be refined.

We anticipate that the terrain classifier creation should pose little risk, in that it should be feasible to create classifiers that give the desired results on the training data. This is justified by the fact that the types of networks we will be training have been shown to perform well in a variety of similar classification tasks. However, there is the risk that the bulk of the training data, being synthetic, will not adequately represent the real world. This would mean that the terrain classifiers can perform well in simulation but not in the real world. We cannot mitigate this risk by only using real world data because of the time and effort required to label the data. We can attempt to mitigate it by generating synthetic data from simulated environments that are similar to the environment where we will be testing - i.e. by limiting the scope of the data

to environments similar to Iceland, instead of using e.g. tropical environments, or contrived environments that do not represent the real world at all. Additionally, we can use human-made post-processing wrappers to handle undesired behavior - e.g. selecting conservatively from the safe landing regions that the network identifies.

Testing in simulation will have low risk, as AirSim already has plugins for both ArduPilot and PX4, as well as drone models that can fly autonomously in the simulation. This step will mostly consist of integrating existing components with the terrain classifiers and pre-/post-processing wrappers.

The first part of testing on physical platforms (in a lab setting), poses low risk. Creating the infrastructure to transfer data to an external board is not difficult, and measuring the framerate and power consumption is also not difficult. There is a risk in the case that none of the terrain classifiers can run fast enough, or with sufficiently little power consumption, on our particular hardware platforms. In this case we will need to consider getting different platforms, but at the very least we will have collected relevant and helpful data on the computational requirements of the terrain classifiers. In the case that we both cannot generate networks that run fast enough on our hardware, and we simultaneously cannot obtain sufficiently fast platforms, we can still use sophisticated ground-based hardware for real time performance (similar to the system architecture in the proof of concept), and make incremental improvements in efficiency. However, in this case we will have to give up the goal of running our solutions purely onboard the drone.

Testing the system for real world landings is the most risky part of the research, not just in terms of logistical considerations and potential of crashes, but also in that it depends on the high performance of the systems developed in the previous steps, and also on their integration together in a single system. The biggest risk that we anticipate is that the training data does not correspond well enough to the real world for the system to properly identify safe landing sites. Although we will try several ways to mitigate this risk, it cannot be entirely eliminated. However, in the case that the synthetic data do not transfer to the real world, we have a two-pronged basis on which to move forward. First, we will have already conducted research on the types of network architectures, sensor data, and pre-/post-processing wrappers that perform well in simulation, and we will still be able to use these. Second, we will have physical drone platforms that we can use to collect more real-world data, and we can label this data according to simple, slow, human-made methods, without considering segmentation masks. Then, we can re-train the networks that we have on purely real-world data.

Bibliography

[1] "Ardupilot.org." [Online]. Available: https://ardupilot.org/

[2] L. Meier. Pixhawk. (accessed: 2020.6.5). [Online]. Available: https://pixhawk.org/

[3] Dronecode, "MAVLink Micro Air Vehicle Communication Protocol," (accessed: 2021.5.19). [Online]. Available: https://mavlink.io/en/

[4] E. Kakaletsis, I. Mademlis, N. Nikolaidis, and I. Pitas, "Bayesian Fusion of Multiview Human Crowd Detections for Autonomous UAV Fleet Safety," 09 2020. [Online]. Available: https://doi.org/10.23919/Eusipco47968.2020.9287442

[5] Derek van de Ven, "Applying rendered UAV images to improve convolutional deep learning object detection performance for animal conservation," Master's thesis, Wageningen University and Research Centre, 05 2020. [Online]. Available: https://edepot.wur.nl/532441

[6] R. Madaan, N. Gyde, S. Vemprala, M. Brown, K. Nagami, T. Taubner, E. Cristofalo, D. Scaramuzza, M. Schwager, and A. Kapoor, "AirSim Drone Racing Lab," 03 2020. [Online]. Available: https://arxiv.org/abs/2003.05654

[7] J. S. Wynn, "Visual Servoing for Precision Shipboard Landing of an Autonomous Multirotor Aircraft System," September 2018. [Online]. Available: http://hdl.lib.byu.edu/1877/etd10385

[8] A. Borowczyk, D.-T. Nguyen, A. Phu-Van Nguyen, D. Q. Nguyen, D. Saussié, and J. L. Ny, "Autonomous landing of a multirotor micro air vehicle on a high velocity ground vehicle," *IFAC-PapersOnLine*, vol. 50, no. 1, pp. 10 488–10 494, 2017, 20th IFAC World Congress. [Online]. Available: https://doi.org/10.1016/j.ifacol.2017.08.1980

[9] D. Falanga, A. Zanchettin, A. Simovic, J. Delmerico, and D. Scaramuzza, "Vision-based autonomous quadrotor landing on a moving platform," 10 2017.

[10] J. Wubben, F. Fabra, C. Calafate, T. Krzeszowski, J. Marquez-Barja, J.-C. Cano, and P. Manzoni, "Accurate Landing of Unmanned Aerial Vehicles Using Ground Pattern Recognition," *Electronics*, vol. 8, p. 1532, 12 2019. [Online]. Available: https://doi.org/10.3390/electronics8121532

[11] K. Pluckter and S. Scherer, *Precision UAV Landing in Unstructured Environments*, 01 2020, pp. 177–187. [Online]. Available: https://doi.org/10.1007/978-3-030-33950-0_16

[12] R. Polvara, M. Patacchiola, S. Sharma, J. Wan, A. Manning, R. Sutton, and A. Cangelosi, "Toward end-to-end control for UAV autonomous landing via deep reinforcement learning," in *2018 International Conference on Unmanned Aircraft Systems (ICUAS)*, June 2018, pp. 115–123. [Online]. Available: http://dx.doi.org/10.1109/ICUAS.2018.8453449

[13] F. Cocchioni, A. Mancini, and S. Longhi, "Autonomous navigation, landing and recharge of a quadrotor using artificial vision," 05 2014, pp. 418–429. [Online]. Available: https://doi.org/10.1109/ICUAS.2014.6842282

[14] V. Desaraju, N. Michael, M. Humenberger, R. Brockers, S. Weiss, J. Nash, and L. Matthies, "Vision-based landing site evaluation and informed optimal trajectory generation toward autonomous rooftop landing," *Autonomous Robots*, vol. 39, 07 2015. [Online]. Available: http://dx.doi.org/10.1007/s10514-015-9456-x

[15] T. Patterson, S. McClean, P. Morrow, G. Parr, and C. Luo, "Timely Autonomous Identification of UAV Safe Landing Zones," *Image and Vision Computing*, vol. 32, 09 2014. [Online]. Available: https://doi.org/10.1016/j.imavis.2014.06.006

[16] M. S. Whalley, M. D. Takahashi, J. W. Fletcher, E. Moralez, L. C. R. Ott, L. M. G. Olmstead, J. C. Savage, C. L. Goerzen, G. J. Schulein, H. N. Burns, and B. Conrad, "Autonomous Black Hawk in Flight: Obstacle Field Navigation and Landing-Site Selection on the RASCAL JUH-60A," *J. Field Robot.*, vol. 31, no. 4, p. 591–616, Jul. 2014. [Online]. Available: https://doi.org/10.1002/rob.21511

[17] A. Johnson, J. Montgomery, and L. Matthies, "Vision guided landing of an autonomous helicopter in hazardous terrain," in *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, 2005, pp. 3966–3971.

[18] R. Garg, S. Yang, and S. Scherer, "Monocular and Stereo Cues for Landing Zone Evaluation for Micro UAVs," 12 2018. [Online]. Available: https://doi.org/10.48550/arXiv.1812.03539

[19] D. Maturana and S. Scherer, "3D Convolutional Neural Networks for landing zone detection from LiDAR," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, 2015, pp. 3471–3478. [Online]. Available: https://ieeexplore.ieee.org/document/7139679

[20] J. Springer, "Autonomous Landing of a Multicopter Using Computer Vision," Master's thesis, Reykjavík University, 2020. [Online]. Available: http://hdl.handle.net/1946/36422

[21] Emlid. Navio2. (accessed: 2020.6.5). [Online]. Available: https://emlid.com/navio/

[22] J. Ulrich, "Fiducial Marker Detection for Vision-based Mobile Robot Localisation," pp. 18–21, 2020, bachelor Thesis. [Online]. Available: https://doi.org/10.1109/SSRR.2017.8088164

[23] J. Springer, "WhyCon ROS Github Repository," (accessed: 2022.3.3). [Online]. Available: https://github.com/uzgit/whycon-ros