VISION-BASED MARKER-LESS LANDING OF A UAS ON MOVING GROUND VEHICLE

A Thesis

by

BLAKE CAMERON KRPEC

Submitted to the Graduate and Professional School of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

May 2022

Major Subject: Aerospace Engineering

ABSTRACT

In recent years the use of unmanned air systems (UAS) has seen extreme growth. These small, often inexpensive platforms have been used to aid in tasks such as search and rescue, medicinal deliveries, disaster relief and more. In many use cases UAS work alongside unmanned ground vehicles (UGVs) to complete autonomous tasks. For end-to-end autonomous cooperation, the UAS needs to be able to autonomously take off and land on the UGV. Current autonomous landing solutions often use fiducial markers to aid in localizing the UGV relative to the UAS, an external ground computer to aid in computation, or gimbaled cameras on-board the UAS. This thesis seeks to demonstrate a vision-based autonomous landing system that does not rely on the use of fiducial markers, completes all computations on-board the UAS, and uses a fixed, non-gimbaled camera. Algorithms are tailored towards low size, weight, and power constraints as all compute and sensing components weigh less than 100 grams. The foundation of this thesis extends upon current efforts by localizing the UGV relative to the UAS using neural network object detection and camera intrinsic properties instead of common place fiducial markers. An object detection neural network is used to detect the UGV within an image captured by the camera on-board the UAS. Then a localization algorithm utilizes the UGV's pixel position within the image to estimate the UGV's position relative to the UAS. This estimated position of the UGV will be passed into a command generator that sends setpoints to the on-board PX4 flight control unit (FCU). This autonomous landing system was developed and validated within a high-fidelity simulation environment before conducting outdoor experiments.

DEDICATION

To my family, friends, and loved ones who supported me along the way.

ACKNOWLEDGMENTS

I would like to thank my advisor Dr. John Valasek for his invaluable guidance and support throughout the completion of this degree. The experience with autonomous systems I have gained under Dr. Valasek through both undergraduate and graduate research over these past 6 years has proven invaluable.

I would like to think Dr. Stephen Nogar for his fundamental guidance and instruction throughout the entirety of this work. His assistance was vital to my learning of many modern robotics tools. The knowledge and skills I now possess regarding these tools will be crucial for my career in autonomous systems. I would also like to thank Dr. John Hrynuk and all other Army Research Laboratory personnel who helped along the way. This work would not have been possible without everyone's support.

I would like to think my committee members for all of their input and guidance in regards to this research, as well as their input and guidance in regards to topics outside of this research.

I would like to thank all of my lab mates in the Vehicle Systems and Control Laboratory for their support throughout the duration of this research, as well as the camaraderie shared. Working with you all has truly been a great experience. Special thanks to Ritwik Bera, Conor Atkins, Chris Leshikar, Garrett Jares, and Luke Moy for their assistance during the experimental testing of this work.

CONTRIBUTORS AND FUNDING SOURCES

Contributors

This work was supported by a thesis committee consisting of Dr. John Valasek and Dr. Reza Langari of the Department of Engineering Technology and Industrial Distribution, Dr. Manoranjan Majji of the Department of Aerospace Engineering, Dr. Srinkanth Saripalli of the Department of Mechanical Engineering, and Dr. Stephen Nogar holding a special appointment from Army Research Laboratory.

This work was supported by the Army Research Laboratory, specifically by Dr. Stephen Nogar and Dr. John Hrynuk. Army Research Laboratory provided the simulation environment, the autonomy software stack used, and technical expertise in support of this work.

Funding Sources

Research was sponsored by a Journeyman Fellowship from the Army Research Laboratory and was accomplished under Cooperative Agreement Number W911NF-20-2-0176. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Laboratory or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation herein.

NOMENCLATURE

UAS	Unmanned Air Vehicle
UGV	Unmanned Ground Vehicle
DOF	Degree of Freedom
VIO	Visual Inertial Odometry
GPS	Global Positioning System
ReLU	Rectified Linear Unit
LReLU	Leaky Rectified Linear Unit
CNN	Convolutional Neural Network
RCNN	Regions with CNN Features
YOLO	You-Only-Look-Once
SSD	Single Shot Detector
RGB	Red Green and Blue
AR	Augmented Reality
FOV	Field of View
RGB-D	Red Green Blue and Depth
LiDAR	Light Detection and Ranging
RADAR	Radio Detection and Ranging
NAS	Network Architecture Searches
mAP	Mean Averaged Precision
CPU	Central Processing Unit
GPU	Graphics Processing Unit
FCU	Flight Control Unit

LPF	Low Pass Filter
PCB	Printed Circuit Board
IMU	Inertial Measurement Unit

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

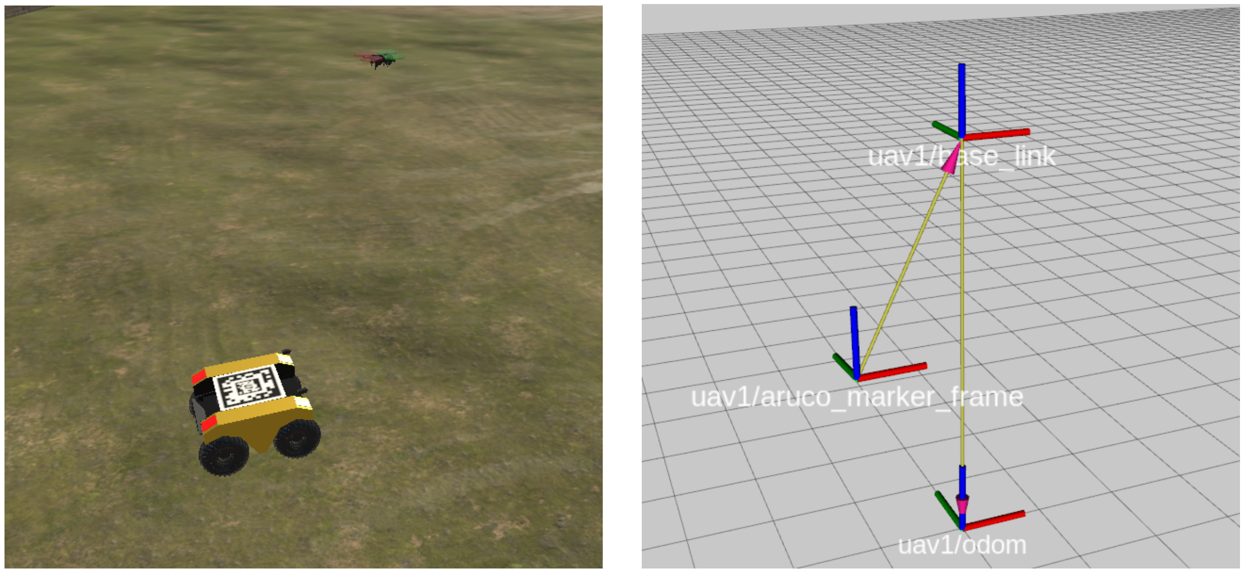
1. INTRODUCTION AND LITERATURE REVIEW

1.1 Research Problem Overview

This thesis presents an autonomous landing system that aims to increase the operational capability of cooperating autonomous air and ground vehicles. The ability for an Unmanned Air System (UAS) to deploy from and return to an Unmanned Ground Vehicle (UGV) autonomously increases the scope of possible cooperative missions as the UAS can perceive a larger portion of the operational environment from its aerial perspective. Returning to the UGV also enables the UAS to recharge it batteries, extending the duration of cooperative autonomous missions.

To enable autonomous landing this work aims to localize the UGV in a fixed inertial frame. Thus, the UAS must be able to keep track of its body frame relative to the inertial frame, as well as estimate the relation between its body frame and the frame of the landing target. Figure 1.1 is generated form the simulation environment used in this work and depicts these three coordinate frames where the UGV's frame is estimated with an ArUcO [1] fiducial marker. The use of fiducial markers is common in autonomous landing on both static [2, 3, 4, 5, 6, 7] and moving [8, 9, 10, 11] targets as they provide a full six degree-of-freedom (DOF) pose estimate of the UGV [1, 12]. The UGV may be able to transmit its location to the UAS, alleviating the need to estimate the frame of the landing target relative to the UAS [13, 14]. However, this work assumes the UGV is unable to broadcast its position to the UAS during the autonomous landing.

In tactical operational environments, the placement of a fiducial marker on the UGV may not be permissible. Thus, the presented autonomous landing system aims to enable autonomous landing of UAS on a moving UGV without the employment of fiducial markers. To accomplish this a neural network based object detector [15, 16, 17] is used alongside an algorithm to estimate a three-dimensional position of the detected object using the pixel coordinates, UAS altitude, and camera intrinsic properties [18]. Alternative methods exist that estimate a three-dimensional bounding box [19, 20, 21, 22, 23, 24]. These methodologies use a single neural network to detect the object



(a) (b)

Figure 1.1: (a): UAS hovering over UGV wih ArUcO marker. (b): Inertial Frame, UAS Body Frame, and ArUcO Marker Frame visualized.

as well as estimate its pose. However, their novelty and complexity create challenges due to the difficultly in generating training data and implementation, and thus are not considered for use in this work [25].

The computational expense of vision algorithms can limit their deployment on embedded platforms such as small UAS. Many works avoid this issue by employing a ground computer to complete all vision computations, and then broadcast only control commands to the UAS [26, 27]. This thesis aims to alleviate this requirement by completing all necessary computations on-board the UAS, akin to works such as [11, 28, 29]. While this work aims to land on a UGV, extensive efforts have also investigated autonomous vision-based landing on maritime platforms [30, 31, 32, 33]. The maritime environment introduces the unique challenge of landing on a vessel that may be perturbed by waves interacting with the vessel [32, 33]. The consideration of landing pad angular oscillations is not encompassed in the scope of this work.

Once the UAS has localized the UGV, that information is passed to a setpoint command generator. The command generator is responsible for taking the UGV position estimates as an input and outputting setpoint commands to the UAS flight controller. Other works use a custom landing controller to achieve increased performance [34], however this work will rely on the on board PX4 [35] flight controller to achieve setpoint commands that result in an autonomous landing. This approach requires the need for external positioning information for the UAS, such as Global Positioning System (GPS). Other works have demonstrated fiducial marker based landings using alternative positioning information such as Visual Inertial Odometry (VIO) [28] as well as completely removing the need for external positioning information [11]. However, this work will rely on GPS for positioning during autonomous outdoor landings. Preliminary development of this marker-less approach was completed in a simulation environment to avoid catastrophic failure before the autonomous landing was tested in an outdoor, unstructured environment.

1.2 Literature Review

This literature review will focus on three main areas: object detection, detection localization, vision-based landing of UAS.

1.2.1 Object Detection

This section addresses the use of neural networks for detecting objects within images. Specifically, the state-of-the-art in object detection, and the advancements in deploying these methods in real time on limited compute power embedded systems, such as UAS.

1.2.1.1 Neural Network Object Detectors

Neural networks, also called feedforward neural networks, or multilayer perceptrons (MLPs), are said to be universal approximators [36, 37, 38, 39]. Inspired by biological neural networks [40], neural networks utility lies in a few key attributes. The first being the ability to modify themselves to represent a provided data set without the use of any predefined explicit function or distribution in the network model [41]. Second, the ability to approximate any function given the network is of sufficient size [41, 42]. Third, neural network models are nonlinear, meaning they are able to learn, and approximate complex relationships often present in real world applications [41]. Neural networks consist of a series of functional transformations where a linear combination of weights

w is transformed for nonlinear activation functions *h(.)* along layers [43]. Layers often consist of many units which act in parallel representing a vector-to-scalar function [43]. Bishop shows that a neural network with *D* inputs, a single hidden layer consisting of *M* units, and *K* outputs can be represented as shown in Equation 1.1 [36]. Figure 1.2 is reprinted from [36] and visually depicts an arbitrary neural network.

$$y_k(\boldsymbol{x}, \boldsymbol{w}) = \sigma\left(\sum_{j=1}^{M} w_{k_j}^{(2)} h\left(\sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)}\right) + w_{k0}^{(2)}\right). \tag{1.1}$$

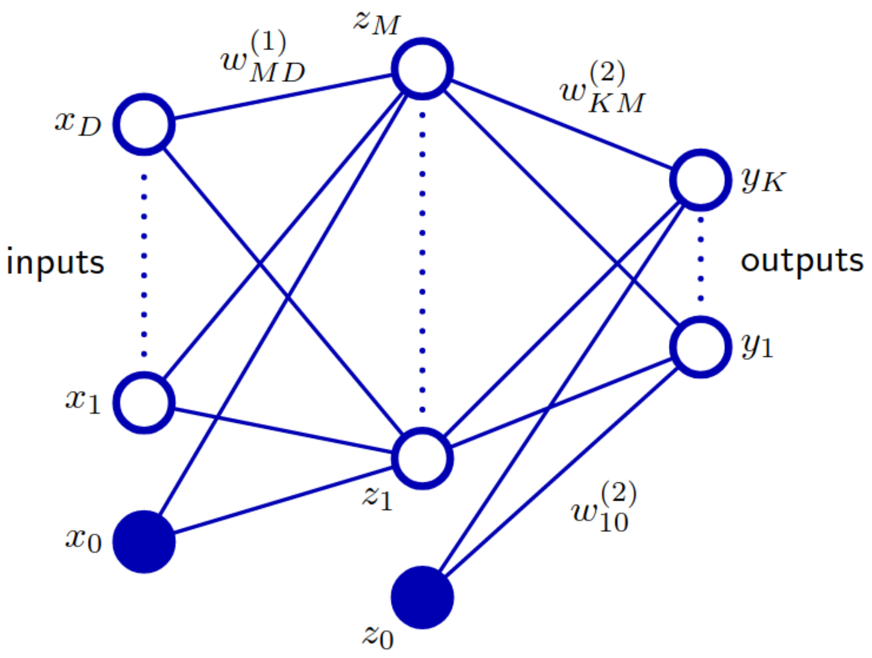


Figure 1.2: Representation of neural network with *D* inputs, one hidden layer consisting of *M* units, and *K* outputs; reprinted from [36].

The activation function *h(.)* is responsible for the transformation of an input signal into an output signal to be fed as input into the next layer of the neural network [44]. The activation function must be selected carefully as the activation function can affect prediction accuracy [45]. Some of the most common activation functions are Linear, Sigmoid, Hyperbolic Tangent (Tanh), Rectified Linear Unit (ReLU), and Leaky Rectified Linear Unit (Leaky ReLU). Table 1.1 summarizes advantages and disadvantages of these common activation functions [44, 45].

Table 1.1: Comparison of Common Neural Network Activation Functions

Name	**Advantages**	**Disadvantages**
Linear	Increased interpretability compared to more complex activation functions.	During back-propagation, gradient is indecent of input due to constant derivative. Unable to identify complex data patters.
Sigmoid	Bounded output, small changes in input result in large changes in output making it ideal for classification	Derivative converges to 0 in positive and negative directions killing gradient descent
Tanh	Bounded output, steeper derivative than Sigmoid increasing learning efficiency, and symmetric around the origin	Derivative also converges to 0 in positive and negative directions killing gradient descent
ReLU	Not all neurons in the network are activated at once, increasing efficiency	Possible for large gradient shifts to cause neurons to never activate
LReLU	Similar to ReLU but attempts to keep output from being stuck at 0 so that no neurons are permanently inactive	Still possible for large gradient shifts to cause neurons to never activate

Neural networks have been applied to forecasting, classification, clustering, approximation, and other pattern recognition tasks. However, in recent years deep Convolutional Neural Networks (CNNs) have proved to be incredibly powerful, vastly surpassing what is possible with a standard neural network [46]. CNNs use a deep architecture consisting of many layers, enabling the ability to learn more complex models. [47]. The increase in complexity that could be represented by these

CNNs eventually led to their deployment for object detection.

In 2012 a group won the ImageNet [48] object detection challenge using a deep CNN [49], sparking interest in the application of deep CNNs for object detection. Soon after in 2015 R. Girshick *et al.* progressed the field by presenting the Regions with CNN features (RCNN) for object detection [50]. The use of RCNNs increased the state-of-the-art by increasing accuracy achievable with object detectors, however these RCNNs did not run fast enough to compute detections in real time. RCNNs create proposed bounding boxes (potentially over 2000 boxers per single image) and completed feature computations on each proposed box, leading to slow detection speed. Fast RCNN increased accuracy of RCNN while achieving a detection speed over 200 times faster than RCNN [51]. Soon after, Faster RCNN would be proposed as an improvement upon Fast RCNN and become one of the first near-realtime deep learning detectors achieving detection at 17 frames per second (fps) [52, 53].

All the aforementioned object detection methods could be classified as two-stage detectors as the image is passed through the network multiple times to complete a detection. You Only Look Once (YOLO) was proposed in 2015 as the first one-stage detector [54]. YOLO applies a single neural network to the entire image and can achieve detections at 45 fps with the enhanced version and 155 fps with the fast version [53, 54]. YOLO has been improved upon in subsequent v2 and v3 versions where accuracy is increased while the fast detection times are maintained [55, 56]. W. Liu *et al.* proposed the second one-stage detector, Single Shot MultiBox Detector (SSD) [53, 57]. SSD introduced multi-reference and multi-resolution detection techniques that increased the accuracy of a one-stage detector, especially regarding the detection of small objects. SSD differs from other one-stage detectors in that it detects different sized objects on different network layers, whilst previous works completed all detections on their top layers [57].

Alongside the desire to increase the accuracy of detections, researchers have also aimed to increase the speed at which detections are computed [58]. Reducing the size and complexity of object detection schemes enables their deployment on less powerful mobile compute systems such as laptops and mobile phones. MobileNet is a CNN for mobile vision applications such as object detection on mobile hardware akin to mobile phones or embedded computers [15, 16, 17]. MobileNet uses depthwise separable convolution to reduce computational expense within the first few layers of the CNN greatly reducing computational expense per detection [15]. Depthwise separable convolution uses separate layers for filtering and combining compared to a standard convolution that completes both tasks in one layer [15]. This greatly reduced the computational expense of object detection, facilitating its deployment on mobile compute platforms [15]. With similar goals of deploying object detection on mobile compute platforms, modifications of the aforementioned YOLO [54] framework were developed to reduce computational expense of detections by reducing the complexity and size of the model. Fast YOLO demonstrated detections 3.3 times faster than YOLOv2 while maintaining similar accuracy [59]. YOLO-LITE demonstrated detections 10 times faster than YOLOv2 but was less accurate [60]. These reductions in computational expense of object detection in recent years have been crucial in enabling object detection to run on mobile and embedded compute systems.

1.2.1.2 Object Detection with UAS

Enabling a UAS to detect objects within its operational environment can enable autonomous behaviors such as target tracking, and autonomous landing. As interest increases in introducing visual sensor data into the control pipeline of UAS, object detection remains a prominent interest in increasing autonomous capabilities. Prior works demonstrate the ability for vision-based target tracking by using color-based blob detection instead of object detection [61]. This approach is proven successful with lightweight color-based blob detection, but relying on color blob detection can lead to problems in environments where the target does not contain unique red blue green (RGB) values. Even before CNN based object detection, more dated object detection methods such as Haar-Cascades [62], quadrilateral transformation [63], and Fisherfaces [64] were investigated for use on UAS for target tracking applications [27]. While Boyers runs the object detection on an off-board computer [27], Haar-Cascade face detection was successfully deployed on-board a UAS [65].

In more recent years interest has increased in deploying deep CNN based object detection for use with UAS imagery, but the computational expense of large CNN object detectors can make them difficult to deploy on mobile compute platforms such as UAS [66]. The computational power limits of UAS can be avoided by completing defections off-board on a more powerful computer. By streaming images captured by a UAS to a ground computer, CNN based object detection has been performed without the need to run the detection on-board the vehicle [67, 68]. This approach introduces the infrastructure requirement of having a ground computer connected to the UAS for deployment. Cloud computing offers an alternative to avoid the use of a ground computer while removing the need to complete the computations on-board the UAS. By keeping the UAS connected to the internet in flight, images can be streamed to a remote computer that performs the object detection and returns the result [69, 70]. While alleviating the need for a ground computer, cloud computing still demands the infrastructure requirement of maintaining an internet connection during operation. Running the detection on-board the UAS is ideal to allow the UAS to operate in environments where a connection to a ground computer or the internet is not available.

Even if not running on-board, object detection still increases the capabilities of a UAS. CNN based object detectors have been used with UAS to search for targets of interest, such as landmines [71] and excavators [72] in aerial images captured by UAS. The ability to send a UAS on an autonomous mission and use the collected imagery to detect hazards such as landmines or excavators digging near a pipeline provides utility in reducing the human workload required to manually search for objects within captured images. Running object detection on UAS imagery has also been used to create an autonomous system to monitor available parking spots in a parking lot, again autonomously completing a task that normally would be completed manually by humans [73]. MobileNet [15] has been used to detect a human target, enabling a follow-me algorithm to follow the detected human target [67]. Detection of another autonomous agent can enable teaming between autonomous agents when explicit communication between them is not available. Perez *et al.* demonstrated the use of object detection on UAS imagery to detect a ground vehicle [74]. The detection of a ground vehicle can facilitate coordination between the UAS and the ground vehicle as demonstrated in [75]. Object detection has also been utilized to enable UAS to detect other

UAS [76]. To enhance performance of UAS detection, a detection method combining RGB and infrared sensor data proved to be effective at detecting UAS as well as distinguishing them from birds [77]. Similar to the detection of ground vehicles, the detection of other UAS can serve as the cornerstone for cooperation between UAS, or interception of an adversarial UAS [78].

1.2.2 Landing Target Localization

This section reviews vision techniques enabling autonomous systems to localize targets of interest. Firstly, the use of fiducial markers commonly used in robotic applications. Secondly, alternative methods for localizing targets without the use of fiducial markers.

1.2.2.1 Fiducial Markers for Pose Estimation

Fiducial markers in a general sense are objects that provide a point of reference or measurement within an image. The use of fiducial markers for estimating pose was popularized by the Augmented Reality (AR) community for tracking objects within the AR environment [79]. By fixing a fiducial marker to a location within the AR environment, any cameras with the marker in their field of view (FOV) could localize themselves relative to the fixed tag, thus localizing themselves in their environment as long as the fixed tags position is know a priori. This capability made its way into the robotics community as fiducial markers can aid an autonomous system in localizing itself within its environment, or localize itself relative to a target of interest [80]. Fiducial markers such as the commonly used QR code often contain visually encoded information that is interpreted by the detection algorithm. However, in the robotics community the amount of information that can be encoded is often traded for increased detection robustness [81].

ARToolkit is a fiducial marker framework developed for AR, developed to overlay virtual images enabling user interaction with virtual objects [82]. ARToolkit was one of the earliest fiducial marker frameworks to see use in the robotics community for pose estimation, and serves as a foundation for most square-shaped markers used for pose estimation [80]. VisualCode is a fiducial marker of a general square exterior shape with 83 internal square bits for encoding information [83]. Instead of developing the marker separately form the detection system, the VisualCode marker was developed for use specifically with mobile hardware such as cell phones. Based on ARToolkit, Rohs proposed binARyID markers using simpler patterns that were selected for robustness against marker rotation and varied lighting conditions resulting in increased accuracy and reduced detection error [84]. ARTag is another ARToolkit based package and remains one of the most widely used fiducial marker software packages [85, 86]. The interior of an ARTag consists of a six-by-six grid enabling each marker to be identified by a unique 36-bit long word [85]. ARTag also employs a gradient based line detection improving both detection reliability and detections during partial tag occlusion [85]. Another marker framework AprilTag expands and improves upon ARTag [12]. AprilTag utilizes an image segmentation algorithm that determines gradient patterns in the image for line detection while also addressing misdetections occurring from rotations and outdoor conditions [12]. This work is expanded further in AprilTag 2 which uses the same coding scheme as AprilTag but increasing detector performance resulting in higher detection rates [87]. Similar to AprilTag, ArUcO is yet another package based on ARTag and ARToolkit [1]. ArUcO allows the user to define the marker library to contain only the markers desired, resulting in a smaller library of markers and thus higher detection rates. The size of a fiducial marker can be an important consideration as smaller markers are able to be detected from a short distance but not from farther distances. Large tags enable detection from farther distances, but cannot be detected from short distances as all four corners of the square marker need to be within the image frame. Fractal fiducial markers solve this problem by embedding smaller markers within a larger marker to increase the detection range of the marker [88, 89, 90]. Fractal tags are useful in UAS applications where the marker needs to be detectable when the UAS is at significant altitude as well as lower altitude. Table 1.2 summarizes the comparison of some of the most common fiducial markers used in robotics, Figure 1.2 provides visual examples of these markers as well as a fractal marker [80].

Table 1.2: Common Fiducial Markers for Pose Estimation

Name	**Key Features**
ARToolkit	One of the first square markers, hardly used in modern efforts.
ARTag	One of the most widely used packages, has a large user base.
AprilTag	Advanced segmentation provides increased detection robustness.
ArUcO	Configurable marker library can lead to faster detection rates.

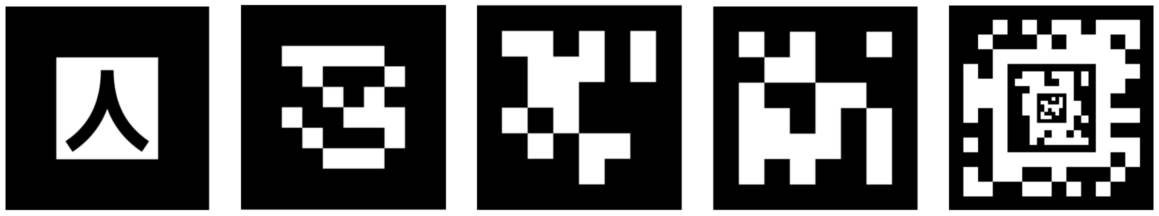


Figure 1.3: Examples of common fiducial markers for pose estimation.

1.2.2.2 Alternative Target Localization

While fiducial markers offer reliable and fast pose updates, the requirement of having a marker on the target of interest may be undesirable in certain applications. Recent efforts have aimed to utilize a two-dimensional bounding box from an object detector as the basis for extracting position or pose estimation of the detected object. The camera intrinsic properties of focal length and principal point coordinates have been used to estimate a detected objects position and velocity from a two-dimensional bounding box [91]. Other works combine the use of stereo cameras with RGB cameras to estimate the position of detected objects [92, 93]. If the stereo and RGB sensors are co-boresighted, the pixel coordinates from a bounding box generated using the RGB image can be mapped to the depth map produced from the stereo sensors, thus giving a position estimate of the detected object. The authors in [94] are able to estimate the pose of a detected object from a bounding box by using a neural network to determine the rotational component of the pose, while a set of equations are used to estimate the translational component. Cabrera *et al.* demonstrated the use of a Hough Forest Regression [95] to generate a continuous probabilistic pose estimate capable of estimating the pose of a detected object between detection frames [96]. An alternative approach is presented in [97] where the viewing angle of the image relative to the object is estimated and used to generate a pose estimate of the detected object. Deformable parts models have also been employed to generate pose estimates from two dimensional bounding boxes [98]. The aforementioned efforts use a method separate from object detection to generate pose estimates. However the computation of two-dimensional bounding boxes and pose estimation have been combined into end-to-end approaches through the use of Structure Kernel Machines [99] and deep Mask RCNNs [100].

Other efforts forego a two-dimensional bounding box and utilize models that can predict a three-dimensional bounding box, providing a pose estimate of the detected object. Figure 1.4 is reprinted from [25] to visually display both a two-dimensional and three-dimensional bounding box, and how a three dimensional bounding box is used as a estimate of the detected objects pose.

Many of these approaches achieve three-dimensional bounding boxes by using stereo cameras [19], RGB-Depth (RGB-D) cameras [20], Light Detection and Ranging (LiDAR) [21], and Radio Detection and Ranging (RADAR) [22]. Serving as a improvement on the previously mentioned methods that use additional sensors, various methods have been developed to estimate a three-dimensional bounding box using only input from an RGB camera. The authors of [23] present

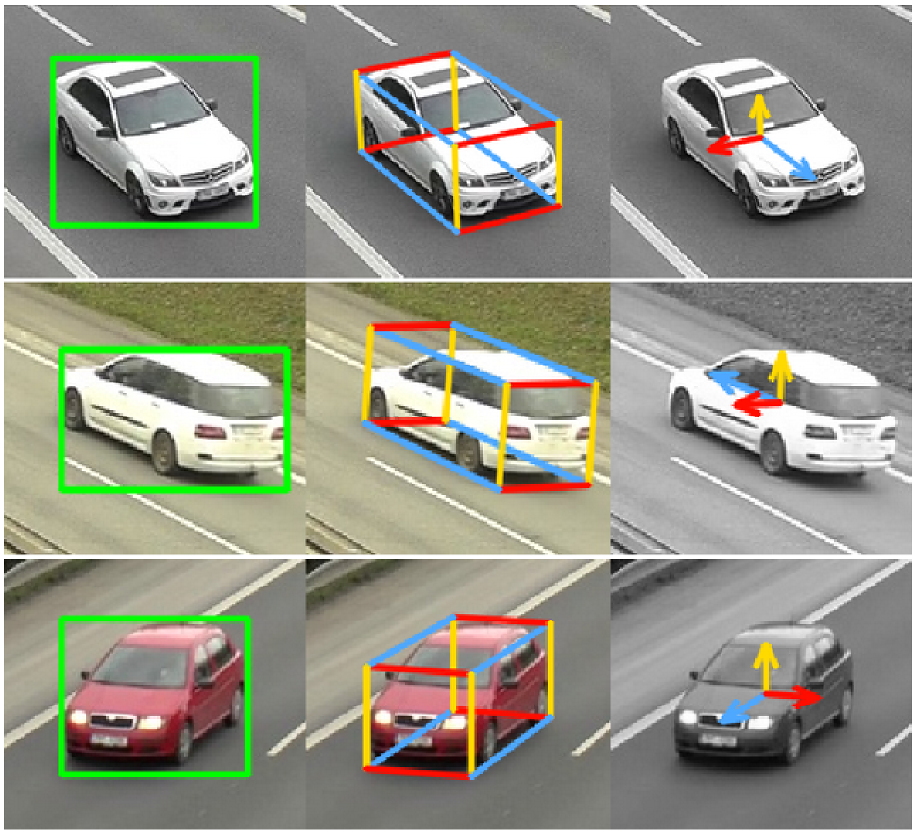


Figure 1.4: Left Column: Two-dimensional bounding box. Middle Column: Three dimensional bounding box, Right Column: Detected object pose estimate from three-dimensional bounding; reprinted from [25].

a single model where a detector shares key features of the detected object with a pose estimator, generating a three-dimensional bounding box. Similar approaches using a CNN with an energy minimization approach [24], and a CNN with structure motion techniques [101] can predict three-dimensional bounding boxes with the assumption that the detected object is on the ground. The assumption that the detected objects are on the ground stems from the development of these methods for autonomous driving. It is worth noting the aforementioned works that use addition sensors (LiDAR, stereo cameras, RGB-D cameras, and RADAR) do not rely on this assumption [19, 20, 21, 22]. While these three-dimensional object detectors have proven successful, many of the most popular methods have attributes such as the reliance on depth information, assumption that objects are on the ground, lack of training data sets, and non-trivial implementation that are currently holding them back from seeing more wide spread usage [102].

1.2.3 Vision-based Landing of Unmanned Air Systems

Visual servoing is simply the use of vision sensor data as feedback to control robotic systems [103, 104]. This sections reviews the use of visual sensor data by UAS for target tracking and autonomous landing.

1.2.3.1 Autonomous Target Tracking

While this thesis intends to demonstrate an autonomous landing, the investigation of target tracking efforts is worthwhile as many of the methods reviewed in this section are either similar to common autonomous landing methods or could be extended to an autonomous landing problem. Fiducial markers capable of providing an accurate pose estimate are common place in many target tracking works. The authors of [105] demonstrate the capability of a UAS to detect a fiducial marker and maintain a position directly above the maker, thus tracking the marker. The same capability has been demonstrated in a multiple target environment and proven to be robust to marker position disturbances [106]. While robustness to small movements is an improvement, other works demonstrate the ability to track a moving fiducial marker target. In [107] the authors demonstrate a UAS tracking a fiducial marker moving arbitrarily, however the movements are bounded as to not exceed the capability of the tracking method. The tracking of an arbitrarily moving fiducial marker target without bounded movements has also been demonstrated [108].

Many other efforts successfully demonstrate target tracking capability without the use of fiducial markers. Pixel coordinates and the geometry of object detector generated bounding boxes have been used as the input for a PID based target tracking controller capable of tracking moving targets [109]. Using pixel coordinates and the geometry of the bounding box alleviates the need to localize the target relative to UAS, potentially increasing computational efficiency of the target tracker. Computation efficiency is an important consideration when using CNN based object detectors, as their computational expense can result in the required use of a ground station computer. A CNN based object detector has been used as the basis for both a Q-Learning target tracker [110], and a

PID target tracker [68] which both utilized an off-board ground station computer. However, some works have been able to achieve the tracking of a moving target using CNN based object detectors while still completing all computations on-board the UAS [111, 112]. Methods not relying on CNNs have also been deployed to successful track moving targets. In [61] color blob detection is used as the basis for a Q-Learning [113] target tracking algorithm to track a moving red truck with a fixed wing UAS. Specific visual features of a target of interest have also been used by an optical flow algorithm to estimate the velocity of the target for tracking purposes [114].

1.2.3.2 Autonomous Landing

Early autonomous landing work demonstrated the use of Hu's moments of inertia [115] to estimate pixel coordinates of the target, and then use simple photogrammetry to estimate the position of the target relative to the UAS [116]. The introduction of fiducial markers increased the performance of autonomous landings as instead of using assumption restricted estimations of target position, researchers have access to an accurate pose of the target. This capability is used to facilitate autonomous landing on a static fiducial marker using traditional control [2, 3, 4], a neural network [5, 6], and Q-Learning [7]. Fiducial markers have also been used to enable autonomous landing on moving targets, such as ground vehicles [8]. A model predictive controller has been implemented with the use of a fiducial markers for target pose estimation, and a gimbaled camera to enable a UAS to land on a ground vehicle moving at twelve meters per second but was only demonstrated in simulation [34]. Other works have demonstrated autonomous landings on moving fiducial markers [9, 10], just not at the speed that [34] was able to achieve in simulation. While many previously mentioned works use GPS for UAS position estimation within its environment, others have implemented autonomous landings on moving fiducial markers without the use of GPS in simulation [117, 118], as well as on hardware in experiments [26, 11, 28, 29]. However, only the authors of [11, 28, 29] were able to implement their methods on-board the UAS, whereas the authors of [26] rely on a ground station computer to complete all vision computations. Some works replace the position information provided by GPS with VIO [28], and others remove the requirement of external position information [11]. Most of the aforementioned works operate independent of their landing target. However, if the landing target is a UGV capable of communicating to the UAS and vice versa, a collaborative framework between the two allows the UGV to broadcast its GPS position to the UAS for use in autonomous landing [13]. This approach does require wireless communication with the UGV, where purely visual approaches do not. The authors of [14] combine the two approaches by having the UGV broadcast its GPS position to the UAS for use when the UAS is far from the UGV, but utilize a fiducial marker on the UGV once the UAS is close enough to visually detect the tag.

1.3 Research Objectives

The objective of this research is to develop a vision-based autonomous landing system enabling a UAS to land on a moving UGV whilst satisfying four specific constraints. First, assume communication between the UAS and UGV is either compromised or nonexistent. Second, complete the autonomous vision-based landing without the use of fiducial markers for pose estimation of the UGV. Third, all vision and control computations necessary to complete the landing must be completed on-board the UAS where all sensing and compute components weigh less than 100 grams. Fourth, the camera used must be fixed to the UAS (not gimbaled).

1.4 Contributions

The key contribution of this work lies in the combination of vision and robotic tools to improve the vision-based landing state of the art by alleviating the requirement of fiducial markers for UGV detection and localization. While demonstrated in an vision-based landing use case in this thesis, this methodology could be applied to other applications involving cooperating or adversarial agents. The presented detection localization scheme could also facilitate robots detecting and localizing objects of interest in their environment without the use of fiducial markers.

1.5 Organization

This thesis is organized as follows. Chapter 2 details the neural network-based object detection methods used for detection of the UGV landing target within UAS captured images. Chapter 3 describes the method utilized to localize the detected UGV relative to the UAS and compares the presented method against a commonly used fiducial marker. Chapter 4 describes the command generator algorithm responsible for generating setpoints that facilitate the autonomous landing, and presents the results achieved in both simulation and experiments. Conclusions from the presented work are summarized in Chapter 5, succeeded by recommendations for future work in Chapter 6.

2. NEURAL NETWORK OBJECT DETECTOR

This chapter presents MobileNet SSD [15, 16, 17], the neural network object detector implemented in this work for detecting the UGV landing target. An overview of the network model is followed by specifics on the training of the network as well as the performance and results achieved.

2.1 Problem Definition

As this work aims to move away from the reliance on fiducial markers for autonomous vision-based UAS landings, the landing target must be detected and localized in an alternative fashion. While fiducial markers offer detection and localization in one algorithm, this work will separate detection and localization into two separate processes. MobileNet SSD object detection is employed to address the former problem by detecting the UGV landing target within images and outputting pixel coordinates of the detected UGV. This chapter aims to demonstrate MobileNet's ability to be trained on a set of UAS captured images to robustly detect the UGV landing target in near real time at a rate sufficient to support the presented autonomous vision-based landing.

2.2 MobileNet Single Shot Detector Neural Network

MobileNets are a family of efficient, small, low latency models that achieve detection rates fast enough to enable near real-time object detection on mobile and embedded compute platforms. Their performance on limited compute power platforms, such as the UAS used in this work, makes them an ideal option for the object detection portion of this autonomous landing work.

2.2.0.1 Depthwise Separable Convolution

MobileNetV1 significantly increased detection speed through the introduction of depthwise separable convolution into the inferencing process [15]. While traditional convolution filters and combines the input in a single step, depthwise separable convolution separates these two tasks to increase efficiency[15]. To show this, the number of multiplications from a tradition convolution,

$m_{convolution}$ can be compared to the number of multiplications required by a depthwise separable convolution, $m_{depthwise}$ [15]. Figure 2.1 depicts traditional convolution, where an input is shown on the left of dimensions $F * F * M$. Applying convolution of a kernel with shape $K * K * M$ results in output shape $C * C * 1$. C denotes the number of unique positions in which the kernel can be slid across the input. If N such kernels are applied to the input, an output of $C * C * N$ is generated. For one convolution operation the number of multiplications is $K^2 * M$. Since the kernel is slid across the input, C convolutions are performed along the height and width of the input, making the number of multiplications per kernel equal to $C^2 * K^2 * M$. Finally if this operation is performed with N kernels, the total number of multiplications $m_{convolution}$ for the described convolution is:

$$m_{convolution} = N * C^2 * K^2 * M. \tag{2.1}$$

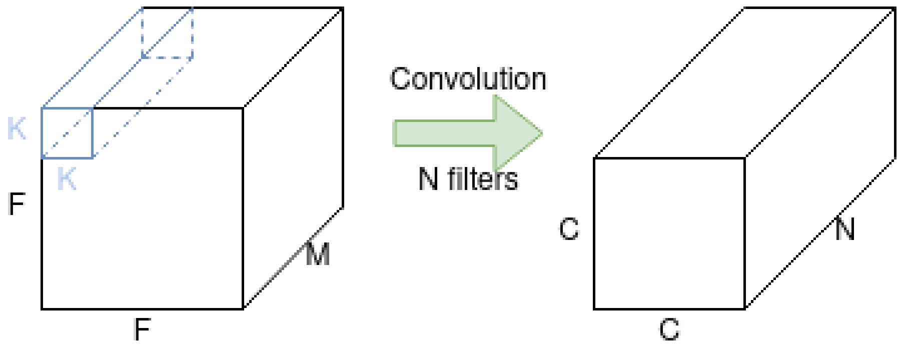


Figure 2.1: Traditional Convolution.

For depthwise separable convolution, the first filtering stage is completed on each channel, instead of all channels simultaneously. Figure 2.2 depicts this process by showing an input of $F * F * M$ where M number of $K * K * 1$ kernels are used instead of the singular $K * K * M$ kernel used in traditional convolution. The product of this step is a data volume of size $C * C * M$. Since convolutions are applied to each input channel, the number of multiplications per convolution is $K^2 * 1$, or just K^2. When applied over the entire input channel, this convolution is performed C

times in height, and width of the input. This increases the multiplications required per channel to $C^2 * K^2$, and when this is applied over all M input channels the total number of multiplications in the first stage, defined as m_1, is:

$$m_1 = M * C^2 * K^2. \tag{2.2}$$

The second step of depthwise separable convolution, the pointwise convolution, is depicted in Figure 2.3. The data volume produced in the first filtering stage of shape $C * C * M$ is convoluted with a kernel of shape $1 * 1 * M$, and if N such kernels are applied, the resulting output is a data volume of shape $C * N * N$, resulting in the same output data volume that was achieved using traditional convolution. The number of multiplications per convolution is $1 * 1 * M$, or just M. Since the convolution is performed C times in both height and width across the input, the number of multiplications per kernel is $C^2 * M$. Finally, if there are N kernels applied, then the number of multiplications required by the second stage, defined as m_2, is:

$$m_2 = N * C^2 * M. \tag{2.3}$$

Thus, the total multiplications needed for a depthwise convolution, the sum of m_1 and m_2 can be defined as:

$$m_{depthwise} = M * C^2(K^2 + N). \tag{2.4}$$

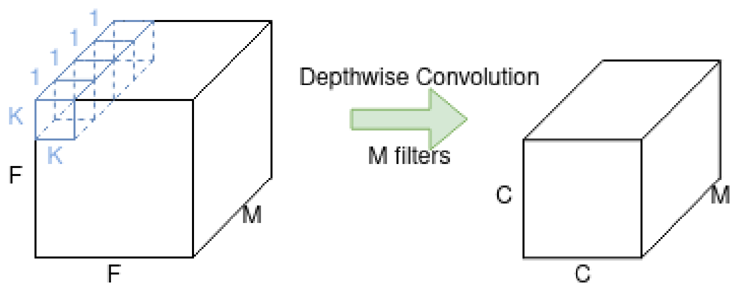


Figure 2.2: Depthwise Convolution, first stage of Depthwise Separable Convolution.

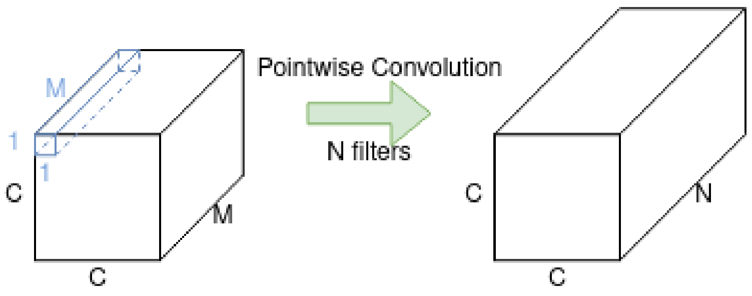


Figure 2.3: Pointwise Convolution, second stage of Depthwise Separable Convolution.

The ratio of and $m_{depthwise}$ and $m_{convolution}$,

$$\frac{m_{depthwise}}{m_{convolution}} = \frac{M * C^2(K^2 + N)}{N * C^2 * K^2 * M} \tag{2.5}$$

can be simplified to

$$\frac{m_{depthwise}}{m_{convolution}} = \frac{1}{N} + \frac{1}{K^2}. \tag{2.6}$$

Equation 2.6 above displays the ratio of multiplications required for depthwise separable convolution and traditional convolution. The ratio simplifies to the sum of reciprocal of the depth of the output volume N, and the reciprocal of the kernel dimension K squared. Showing that for any N and K greater than 1, depthwise convolution is more efficient in terms of multiplications required, reducing computational expense of the convolution.

2.2.0.2 Linear Bottlenecks and Inverted Residuals

MobileNetV1 employed the use of depthwise seperable convolutions to increase performance on mobile compute platforms, and MobileNetV2 expanded upon its predecessor by introducing the use of linear bottlenecks and inverted residuals [15, 16].

In [16], the authors describe a "manifold of interest" as a set of layer activations for any layer *L* for an input set of images. It is also stated that it can be assumed that the manifolds of interest can be embedded in low-dimensional subspaces due to two key properties [16]:

1. "If the manifold of interest remains non-zero volume after ReLU transformation, it corresponds to a linear transformation."

2. "ReLU is capable of preserving complete information about the input manifold, but only if the input manifold lies in a low-dimensional subspace of the input space".

This low-dimensionality of the "manifold of interest" can be captured via the insertion of linear bottleneck layers into the conventional blocks. Results in [16] show that these linear bottlenecks prevents non-linearity from destroying too much information, and in fact show that non-linear bottleneck layers negatively affect performance. Other works also report on findings where the removal of non-linearity from the input to a CNN block lead to improved performance [119].

Residual connections are often used in CNNs as they can improve the ability of a gradient to propagate across multiple layers. Traditionally, residual connections are made between layers with high number of channels. However, the authors in [16] show that residual connections between bottleneck layers are considerably more memory efficient leading to improved performance [16]. Figure 2.4 is reprinted form [16] to visually demonstrate the difference between a traditional residual block and an inverted residual block. Note the hatched layers represent layers not using any non-linearities [16].

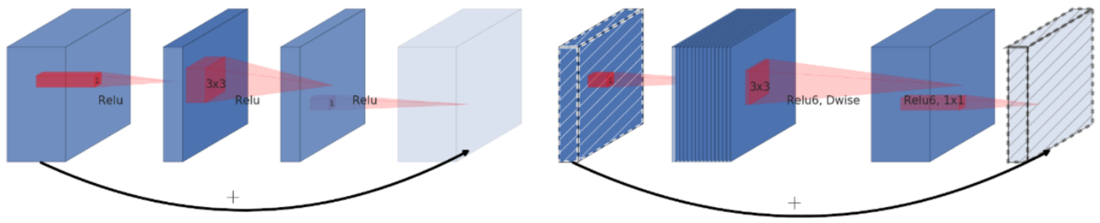


Figure 2.4: Left: Traditional Residual Block. Right: Inverted Residual Block; reprinted from [16].

2.2.0.3 Network Architecture Searches and Model Architecture Advances

MobileNetV3 expands further upon its predecessors by implementing hardware aware network architecture searches (NAS) and model architecture advances [17]. NetAdapt [120] is the NAS employed by MobileNetV3, and is capable of adapting a pre-trained deep neural architecture for mobile platforms. The algorithm decreases the resource consumption by removing filters from one layer during each iteration. It simplifies each layer individually and then selects the resulting network with the highest accuracy. This iterative simplification is repeated until a predefined computational resource budget is satisfied, then the simplified network is fine-tuned in a final step until convergence is achieved. Figure 2.5 is reprinted from [120] and visually represents the NetAdapt algorithm.

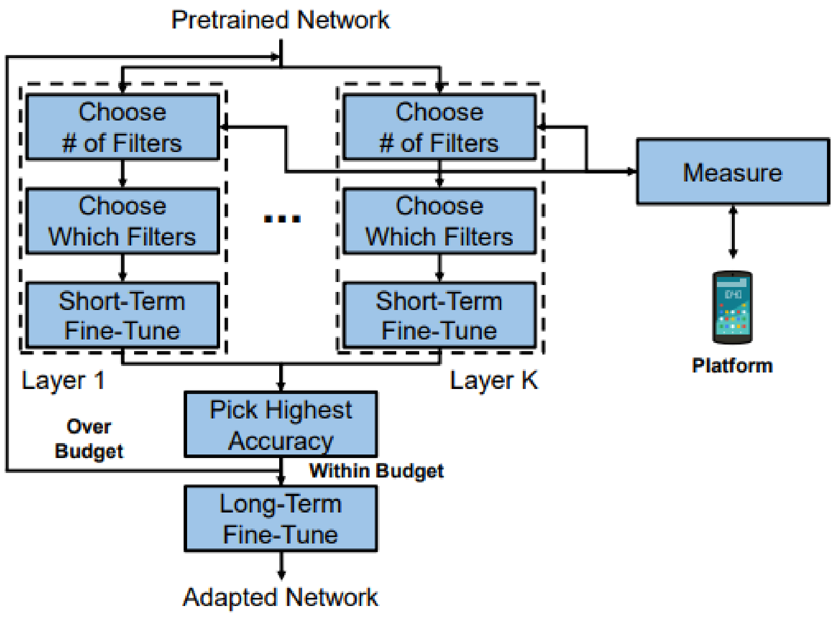


Figure 2.5: Left: Traditional Residual Block. Right: Inverted Residual Block; reprinted from [120].

In addition to the implementation of a NAS, MobileNetV3 also implements novel architecture advances. It expands upon the building block presented for MobileNetV2 (shown in Figure 2.4) by including squeeze and excitation into the linear bottleneck layers as well as a modified activation function. Squeeze and excitation blocks are novel architectural blocks that aim to increase the representational power of a network by using inter channel dependencies to recalibrate the channel-wise feature response [121]. Figure 2.6 reprinted from [17] visually displays the addition of a squeeze and excitation block, shown as the solid blue blocks, to the linear bottleneck inverted residual block presented MobileNetV2.

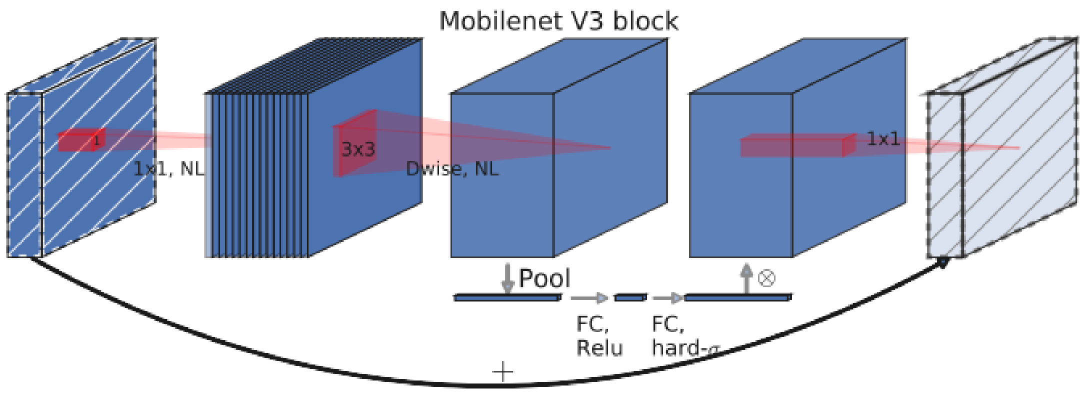


Figure 2.6: MobileNetV2 Linear Bottleneck Inverted Residual Block with Squeeze and Excitation Block; reprinted from [17].

A new activation function is also used in MobileNetV3, hard-swish or h-swish. Works such as [122, 123, 124] present a non-linearity called swish which serves as a drop-in replacement for ReLU and provides a significant increase in neural network accuracy.

$$swish(x) = x * \sigma(x),\ where\, \sigma(x) = \frac{1}{1+e^{-x}} \tag{2.7}$$

While the swish function increases accuracy, the sigmoid function it relies on is computationally expensive, and can hinder performance on mobile compute platforms. MobileNetV3 addresses this by replacing the sigmoid function with its hard alternative, resulting in h-swish defined in Equation 2.8 [17]. Figure 2.7 is reprinted form [17] and provides a comparison between hard and standard versions of sigmoid and swish activation functions.

$$h - swish(x) = x * \frac{ReLU6(x+3)}{6} \tag{2.8}$$

It is observed that the benefits of swish activation are only realized in deeper layers of the network, thus MobileNetV3 only employs h-swish in the second half of the model, ReLU is used elsewhere [17].

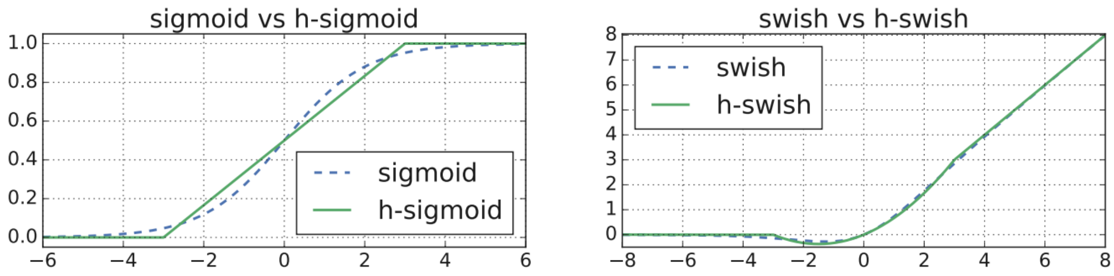


Figure 2.7: Comparison of hard and standard versions of sigmoid and swish activation functions; reprinted from [17].

2.2.1 Model Definition

MobileNetV3 SSD will be used for this work as its improvements provide improved performance over its predecessors MobileNetV1, and MobileNetV2. MobileNetV3 consists of two models MobileNetV3-Small and MobileNetV3-Large, which are optimized for low and high compute resource platforms respectively [17]. As this work aims to deploy the object detection on-board a UAS with limited compute power, MobileNetV3-Small will be used.

2.2.1.1 Training and Performance

The object to be detected in this work is a Clearpath Robotics Warthog [125] with a custom landing pad attached, as displayed in Figure 2.8. A model was also trained on just the landing pad to enable experimentation without the Warthog UGV by placing the landing pad on a garden cart, as displayed in Figure 2.9. Furthermore, a model was trained on the simulated UGV for use within the simulation environment. All models were trained according to the specifications listed in the subsequent sections.

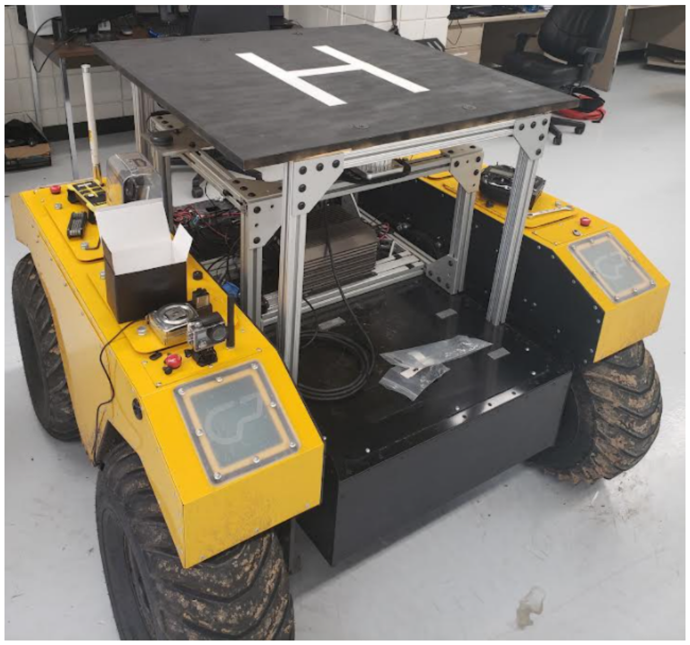


Figure 2.8: Clearpath Robotics Warthog with Landing Pad attached.

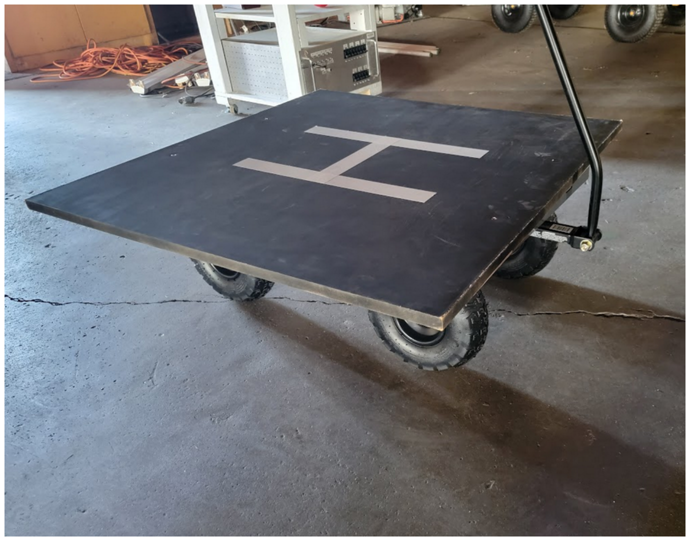


Figure 2.9: Garden cart with Landing Pad attached.

The deep-learning framework TensorFlow [126] was used in this work to aid in the training and deployment of the object detector. TensorFlow was also used to convert the trained network to a TFLite model, which enables the acceleration of inferencing on mobile and embedded compute platforms. TensorFlow was also used to quantize the trained TFLite model to further increase performance on mobile compute platforms.

The training data used is a set of 710 images captured by the VOXL m500 UAS [127] used in this work. The software tool RoboFlow [128] was utilized to augment these images. Augmenting images can aid in the training of neural network object detectors by increasing the robustness in the presence of noise and distortion in images. Table 2.1 below summarizes the augments used, and Figure 2.10 displays some examples of images before and after augmentation. The noise augmentation parameter defines how many of the pixels within the image are replaced by either white or black pixels. The location of these noise pixels is random. Rotation defines the amount by which the image is rotated in degrees, the rotation amount is chosen randomly as a value within the predefined bounds. Similarly, saturation augments the images by adjusting the saturation of the image by a random amount within the predefined range. The training data set for this work consisted of all original images without augmentation, as well as two sets of augmented images bringing the total number of images used in training up to 2130. This number of training images was sufficient to facilitate the training of the model to detect the UGV at an altitude of five meters.

Table 2.1: Augmentations Applied to Training Images

Augmentation	**Parameters**
Noise	3%
Rotation	± 15 °
Saturation	± 35%

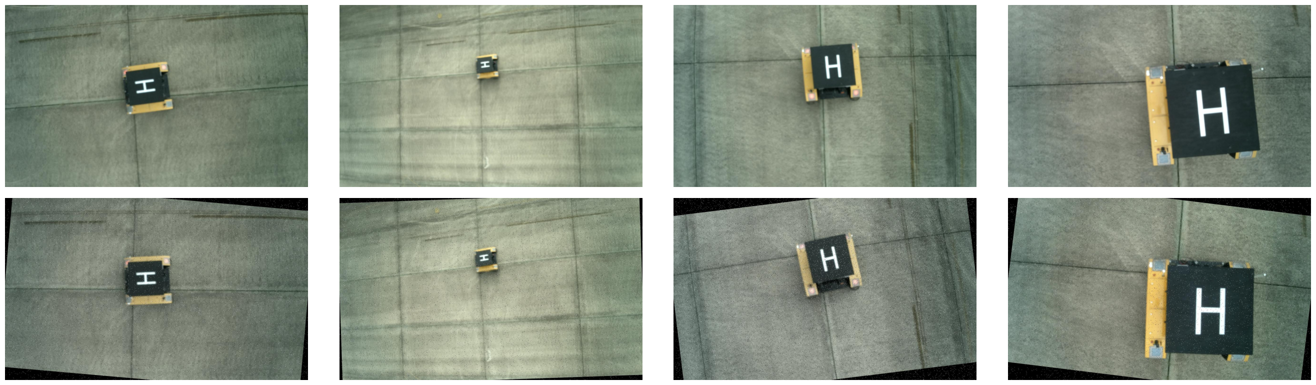


Figure 2.10: Top Row: Raw images. Bottom Row: Examples of images augmented as defined in Table 2.1.

Using this augmented set of training images, various network sizes were trained in an attempt to observe if network size significantly affected speed or accuracy of the trained detector. MobileNetV3 defaults to a model size of 300 x 300. This model size and both one larger and one smaller were trained. Large network sizes can more reliably detect smaller objects within an image at the expense of increased computational cost. Two performance metrics were observed for each model for comparison, inference speed, and mean average precision (mAP). Inference speed is used as a metric to represent computational expense, high inference times indicate an increase in computational expense. The same image was input into each trained network 30 times. Each time the inference time was measured, and all 30 inference times for each trained network were averaged to yield the values presented in Table 2.2. All inference times were measured on-board the VOXL m500 UAS whose specifications are listed in Appendix A. mAP is a metric commonly used to quantify object detection accuracy, and represents the accuracy of the trained model on the training data set. Higher mAP value indicate higher accuracy, with the maximum mAP value being 1.0. Training was run until the mAP was observed to no longer increase in a span of 1500 training steps.

Table 2.2: Network Size Comparison

Network Size	**Inference (ms)**	**mAP**
200x200	16.5	0.744
300x300	27.1	0.774
400x400	37.7	0.802

As expected, increasing the network size increases the mAP achieved at the expense of increased inference time. Examination of Table 2.2 shows that increases in network size yield incremental increases in the mAP achieved at the expense of the increased inference time. Since neither of the larger network sizes yield significant accuracy improvements, it was decided that a model size of 200 x 200 is adequate for detecting the features of interest for the UGV landing target. In summary, a model size of 200 x 200 is selected for this work as it proved successful in detecting the UGV landing target, whilst minimizing the computational power required resulting in faster inference times.

2.2.1.2 Quantization

Quantization of a neural network is the process of approximating the 32-bit floats representing fixed values such as trained weights with lower bit numbers, often 8-bit integers of 16-bit floats. This process reduces model size and improves detection speed with little degradation to model accuracy [126]. 16-bit quantization is used in this work as it is recommended by the Tensorflow framework for models deployed on Graphics Processing Units (GPUs). By quantizing the trained model to 16-bit floats from 32-bit floats, the model size and inference latency were reduced. Specifics in these reductions are specified in Table 2.3.

Table 2.3: Quantization Performance Change

	Non-Quantized Model	**Quantized Model**	**Percent Reduction**
File Size	7,178,300 bytes	3,646,188 bytes	49.2 %
Inference Speed	16.5 (ms)	13.7 (ms)	17.0 %

The ability for quantization to reduce latency may lead to inquiries about using a larger model for increased accuracy, and using quantization to reduce the incurred additional inference time. This was not considered in this thesis as the reduction in latency from quantization was not significant enough to reduce the majority of additional latency of larger networks.

2.3 Object Detection Results

The resulting quantized, 200 x 200 size MobileNetV3 SSD object detector was selected for the detection of the landing target portion of this work for the reasons summarized in subsequent sections. Figure 2.11 displays a few example UGV detections completed on-board the VOXL m500 UAS, and Figure 2.12.

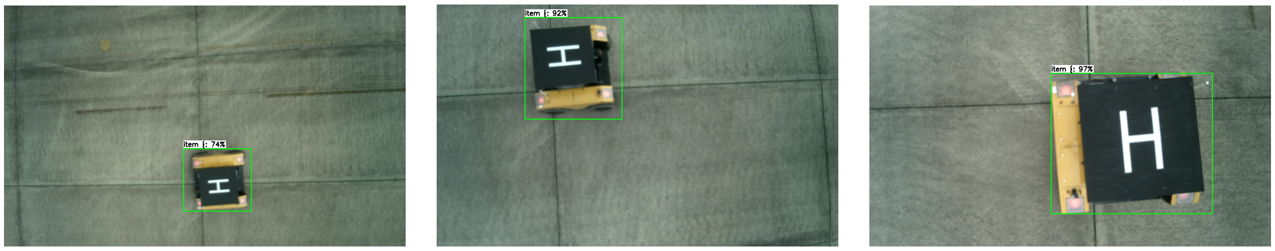


Figure 2.11: Examples of UGV landing target detection.

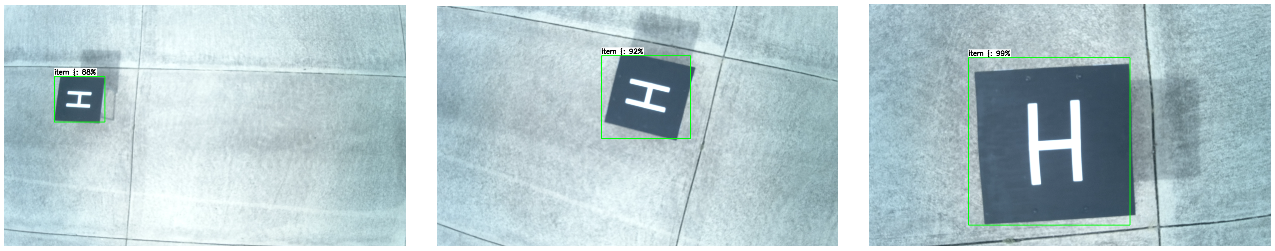


Figure 2.12: Examples of cart landing target detection.

3. LOCALIZATION OF DETECTED GROUND TARGET

The method and software described in this chapter relies upon concepts existing within the ROS package *image_undistort* [129]. The functionality provided by *image_undistort* was expanded upon by Army Research Lab to create the software capable of estimating the three-dimensional position of the UGV target, as described in this chapter [18].

3.1 Problem Definition

Object detection handles the localization of the landing target within an image; however, this work also aims to estimate the UGVs position in three-dimensional space relative to the UAS. Fiducial markers are often used in autonomy development and can a 6 DOF pose estimate of the target. This thesis aims to alleviate the requirement of fiducial markers by estimating the position of the UGV relative to the UAS by utilizing camera intrinsic properties. To accomplish this, the camera must be modeled. This model was used in conjunction with the pixel coordinates of bounding boxes generated from object detection to provide a position estimate of the UGV.

3.1.1 Camera Model

The cornerstone of this works ability to transfer pixel coordinates in the image frame into three-dimensional coordinates in the world is the use of the pinhole camera model, as described in [130]. The pinhole camera model is used to describe the mathematical relation between the three-dimensional position of a point and its projection into the image frame of a camera. This model is not without shortcomings as it does not account for geometric distortions, or any affects caused by lenses, and apertures. Attempting to correct for any distortion in images captured can negate some of the affects not captured by the pinhole camera model.

A visual depiction of the pinhole camera model is displayed in Figure 3.1. The model depicts a point in three-dimensional space with coordinates of (X, Y, Z), and its projection on to the image plane in pixel coordinates (u, v). The image plane exits parallel to the plane formed by the X and Y axis of the coordinate system, and is a fixed distance f, the focal length, from the origin of the coordinate system.

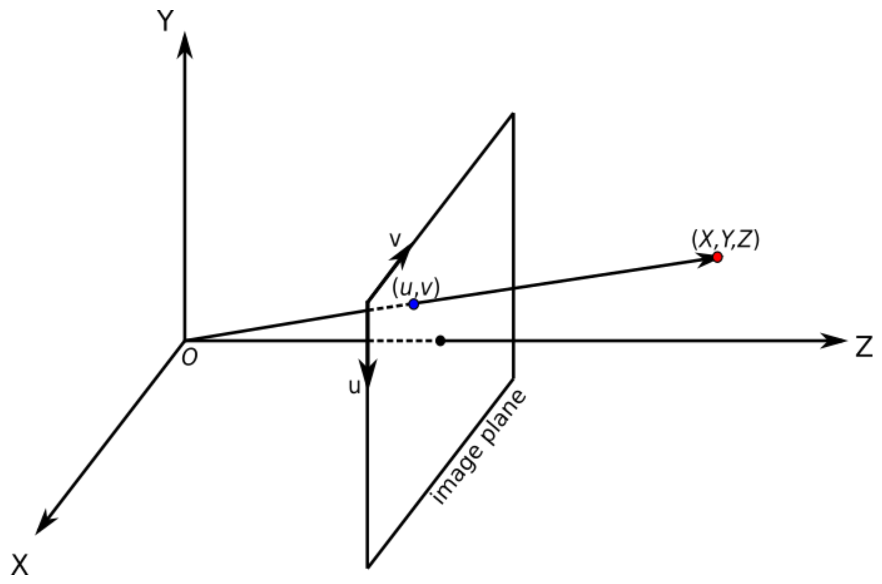


Figure 3.1: Visualization of the pinhole camera model.

By examining the Y-Z plane formed in Figure 3.1, a relation using similar triangles can be developed to generate an expression for the pinhole camera model. Figure 3.2 visually depicts how similar triangles exist within the pinhole camera model. The blue right triangle formed to the point (u, v) is similar to the red triangle formed to the point (X, Y, Z). Using the information that these triangles are similar, and that the image plane exists at f along the Z axis, their relation can be writeen as depicted in 3.1

$$\frac{u}{f} = \frac{Y}{Z}, \text{ or } u = \frac{Yf}{Z} \tag{3.1}$$

In similar fashion, Equation 3.2 can be written to describe the same relationship, but in the X-Z plane.

$$\frac{v}{f} = \frac{X}{Z}, \text{ or } v = \frac{Xf}{Z} \tag{3.2}$$

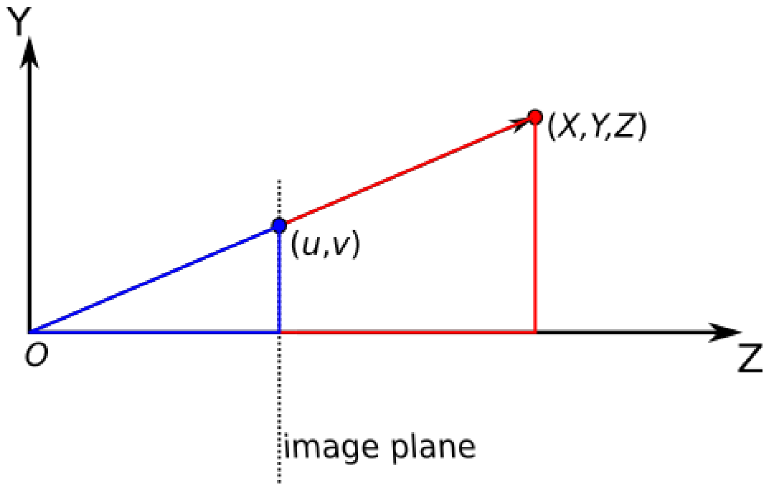


Figure 3.2: Visualization of similar triangles within the Y-Z plane of the pinhole model.

Equation 3.3 summarizes the expression for the pixel coordinates of a point in three-dimensional space projected onto an image plane with a known focal length.

$$\begin{bmatrix} u & v \end{bmatrix} = \begin{bmatrix} \frac{Yf}{Z} & \frac{Xf}{Z} \end{bmatrix} \tag{3.3}$$

To represent Equation 3.3 in a more usable form, homogeneous coordinates are introduced as w and T [131]. This enables Equation 3.4 to be written in matrix form and map the projection of a point in three-dimensional space onto an image plane. c_x and c_y account for the offset in origin between the image plane and coordinate frame, as typically an image frame coordinate system's origin exists at the top left corner of the image plane as depicted in Figure 3.1.

$$\begin{bmatrix} v \\ u \\ w \end{bmatrix} = \begin{bmatrix} f & 0 & c_x & 0 \\ 0 & f & c_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ T \end{bmatrix} \tag{3.4}$$

By examining the results of Equation 3.4 it can be seen the resulting terms are:

$$v = fX + c_x Z \text{ , } u = fY + c_y Z \text{ , and } w = Z. \tag{3.5}$$

If this result is unhomogenized, that is u and v are divided by w, the same result is captured form Equation 3.3 with the addition of the origin offset parameters c_x and c_y. It can also be observed that the additional component T that is introduced during the homogenization of the coordinates (X, Y, Z) has no effect on the result obtained, and is thus ignored to obtain the form presented in Equation 3.6. This form is also convenient in the fact that the matrix is now square and can be inverted. The resulting three by three matrix is defined as the camera intrinsic matrix, K.

$$\begin{bmatrix} v \\ u \\ w \end{bmatrix} = \begin{bmatrix} f & 0 & c_x \\ 0 & f & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \tag{3.6}$$

3.1.2 Landing Target Position Estimation

The pinhole camera model described in Equation 3.6 is often used to describe the projection of a point onto the cameras image frame. However, this work is interested in estimating (X, Y, Z) given pixel coordinates (u, v) from object detection. Since the component Z is lost in the projection, additional information will be required to recapture the full three-dimensional position of the projected point. This section describes the method in which the full three-dimensional position of the point of interest is estimated with the use of the UAS altitude.

First a unit vector originating from the origin O is defined to be oriented towards the point of interest. This vector $\boldsymbol{r}$ is found by inverting the camera intrinsic matrix K and multiplying by the homogenized pixel coordinates as shown in Equation 3.7.

$$\boldsymbol{r} = \begin{bmatrix} r_x \\ r_y \\ r_z \end{bmatrix} = \begin{bmatrix} f & 0 & c_x \\ 0 & f & c_y \\ 0 & 0 & 1 \end{bmatrix}^{-1} \begin{bmatrix} v \\ u \\ w \end{bmatrix} \tag{3.7}$$

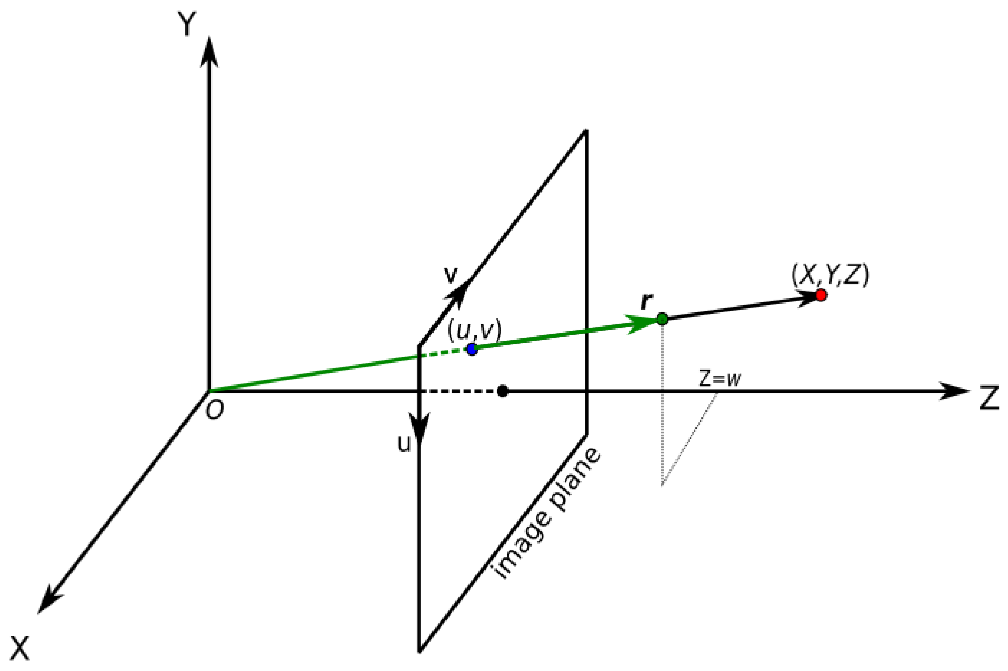


Figure 3.3: Visualization of the vector $\boldsymbol{r}$ on the pinhole camera model.

X, and Y can be directly calculated if w is equal to Z. Since Z is unknown, for now w is set to any value, in this work it was set to one. This defining of w is required to account for the information that is lost during the projection of the three-dimensional point onto the two-dimensional image plane. By observing the resulting expression for $\boldsymbol{r}$ in Equation 3.8 $\boldsymbol{r}$ can now be described as a vector oriented towards the point (X, Y, Z), but does not have the correct magnitude required to represent the point (X, Y, Z). $\boldsymbol{r}$ is normalized as the magnitude of $\boldsymbol{r}$ is not significant, only its orientation. It can be seen in Equation 3.8 that $\boldsymbol{r}$ is in essence a function of w. Figure 3.3 visually depicts $\boldsymbol{r}$ as a green vector on the pinhole camera model.

$$\boldsymbol{r} = \begin{bmatrix} r_x \\ r_y \\ r_z \end{bmatrix} = \begin{bmatrix} \frac{v}{f} - \frac{c_x w}{f} \\ \frac{u}{f} - \frac{c_y w}{f} \\ w \end{bmatrix} \tag{3.8}$$

Since $\boldsymbol{r}$ exists as a function of w, w needs to be estimated to calculate the position of the UGV relative to the camera. As the image itself lacks any depth information, the UAS altitude will be used to provide the depth information required. The altitude estimate is used to create a plane at the ground level, G. This plane G is then offset by a predetermined height h_0 to account for the height of the UGV with respect to the ground, as depicted in Equation 3.9. By using this approach some limitations are placed such as the assumption that the UGV is operating on flat ground, and the required prior knowledge of the height of the UGV. These limitations could be addressed by replacing the plane G with the use of a point cloud generated from stereo vision or LiDAR, but these approaches were not considered due to size weight, and computational power constraints of the UAS used in this work. The vector $\boldsymbol{r}$ will serve as the basis for a parameterized line whose intersection with G yields the position estimate of the UGV.

$$G = g_x(X) + g_y(Y) + g_z(Z) + h_0 \tag{3.9}$$

Next a vector $\boldsymbol{l}$ is defined as <0, 0, 100>. A quaternion $\boldsymbol{q_r}$ defined to represent the orientation of the unit vector $\boldsymbol{r}$. Two points are generated as displayed in Equation 3.10.

$$p_1 = (0,0,0) \text{ and } p_2 = \boldsymbol{q_r l}. \tag{3.10}$$

p_1 and p_2 are the two endpoints of a line whose length is 100, and orientation equivalent to that of $\boldsymbol{r}$. Next a third order parameterized line $p(t)$ is fit between p_1 and p_2, as defended in Equation 3.11.

$$p(t) = p_x(t) + p_y(t) + p_z(t) \tag{3.11}$$

The non-linearity included in the parametrization can aid in accounting for any distortion in the image not accounted for by the camera intrinsic matrix K. For a camera that is accurately calibrated, where K is populated with a high level of fidelity, a linear parameterized line p is adequate. Given the simplified version of K employed in this thesis the non-linearity is employed in the generation of $p(t)$. Now that $p(t)$ exists as a parameterized line that maintains the orientation of $\boldsymbol{r}$ in the presented coordinate frame, the intersection of $p(t)$ and G can be found. The respective components of $p(t)$ can be substituted into Equation 3.9, and t can be solved for, as shown in Equation 3.12.

$$g_x(p_x(t)) + g_y(p_y(t)) + g_z(p_z(t)) + h_0 = 0 \tag{3.12}$$

The resulting value of t is then substituted back into $p(t)$ to yield the position estimate of the UGV. The figures in this chapter have depicted the pinhole camera model with the Z axis in a horizontal orientation. However in this work the camera is downward facing on-board the UAS, meaning the Z axis of the pinhole camera model is oriented in a vertical fashion. Figure 3.4 depicts this as well as provides a visual summary of the process depicted in this chapter.

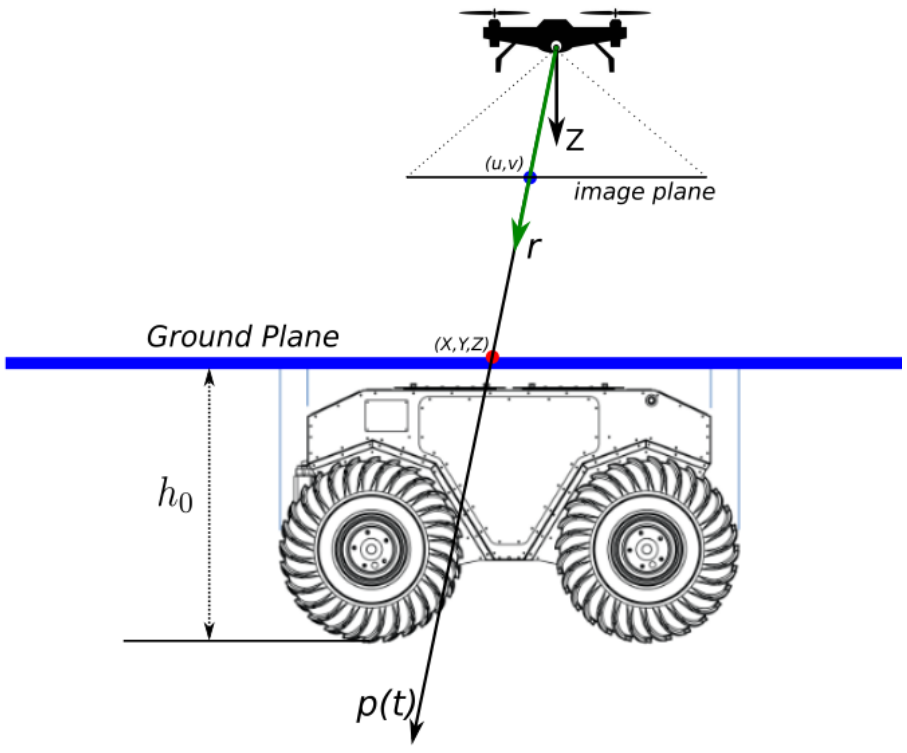


Figure 3.4: Visualization of the detection localization approach used in this work.

3.2 Localization Results

This section provides a few examples of the performance achieved by the presented method. The data presented for each example was collected in a simulation environment in an attempt to remove as many environmental variables as possible. Simulation can also provide exact position differences between the UAS and UGV to be used as a ground truth. The error between this ground truth data and the UGV position estimate from detection localization is plotted below for various scenarios.

First, the UAS was commanded to hover directly over the UGV at an altitude of five meters. The results from this demonstration are displayed in Figure 3.5, and Figure 3.6 displays a few images captured during the experiment to visually represent the UGVs location within the UAS image frame. It can be observed that while the UAS is hovering above a still UGV, the detection localization method discussed in this chapter can yeild accurate position estimates of the UGV relative to the UAS.

Second, the UAS was commanded to hover at an altitude of five meters while the UGV passes underneath such that it enters the bottom of the image and exits the top. Figure 3.7 displays the position error, and Figure 3.8 displays a few images captured during the experiment to visually represent the UGVs path of travel in the UAS image frame. This experiment is a good representation of how image distortion can affect accuracy. When the UGV is near the edge of the image the error increases as the distortion in the image is higher near the edges. Increasing the fidelity of the camera intrinsic matrix will likely reduce the errors seen at edges of images due to distortion. These errors are acceptable for this work as the UAS will eventually navigate to a position near directly above the UGV.

The final experiment conducted is similar to the second, except that the UGV will move across the image from left to right. Similar to the previous experiment, the closer the UGV is to the edge of the image, the increase in distortion causes an increase in the error. Figure 3.9 displays the error plot, and Figure 3.10 displays a few images of the UGVs path across the image. The error as the UGV is near the edges of the image is higher than the previous experiment. This is because the image width is greater than the image height. As previously mentioned, image distortion becomes worse as distance from the center of the image increases. The width of the image is greater than the height, causing this distortion effect to become more significant as the UGV can travel further from the center of the image.

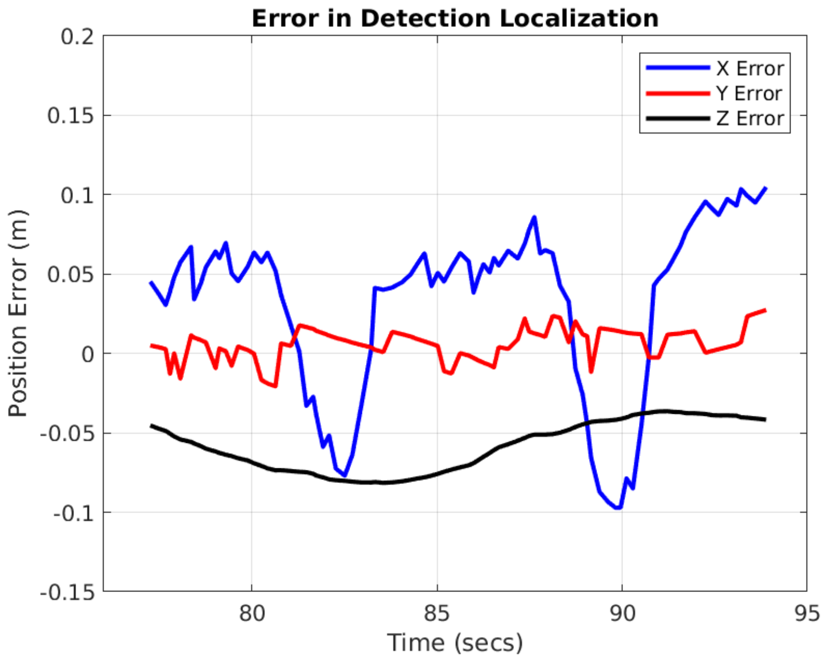
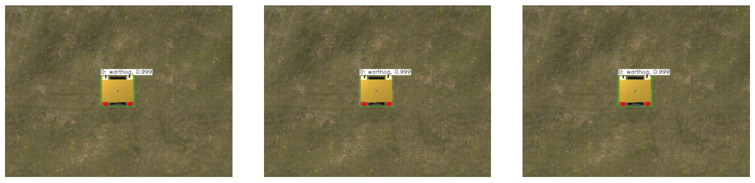


Figure 3.5: Detection localization error with UAS hovering over static UGV.



t = 80 secs t = 85 secs t = 90 secs

Figure 3.6: Time series of the static UGV's position within the image.

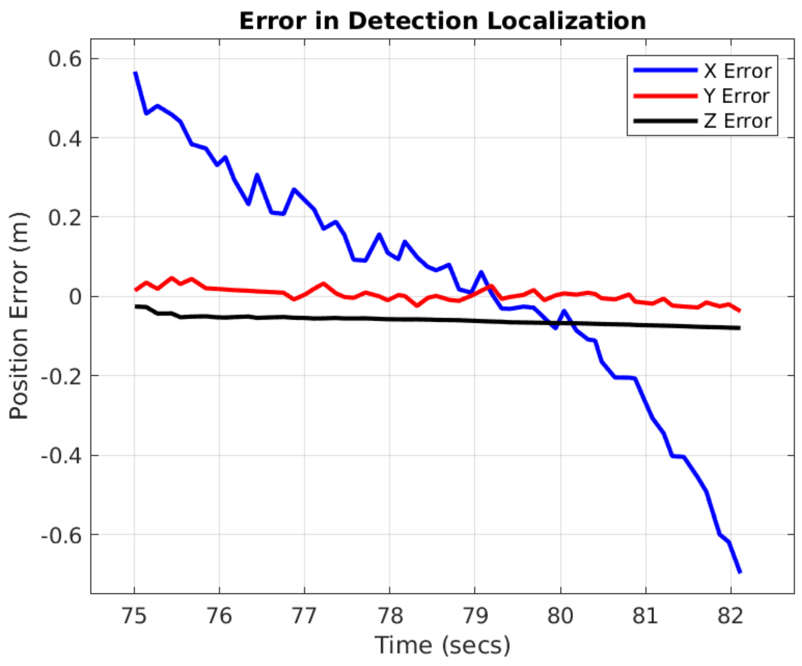
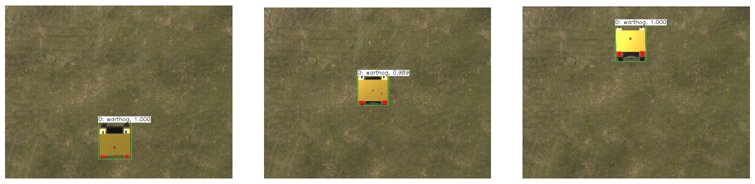


Figure 3.7: Detection localization error with UAS hovering over moving UGV.



t = 77 secs   t = 79 secs   t = 81 secs

Figure 3.8: Time series of the UGV's path across the image from bottom to top.

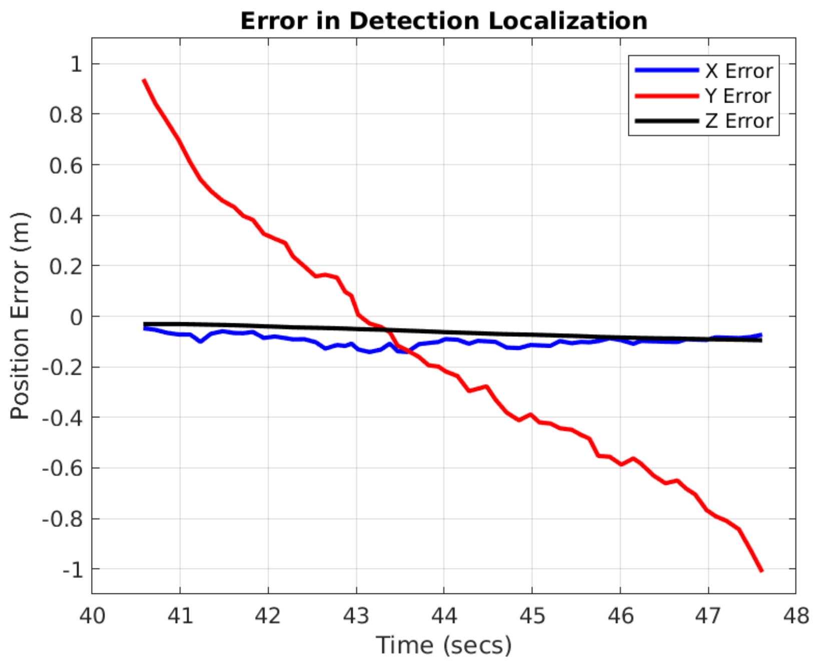


Figure 3.9: Detection localization error with UAS hovering over moving UGV.

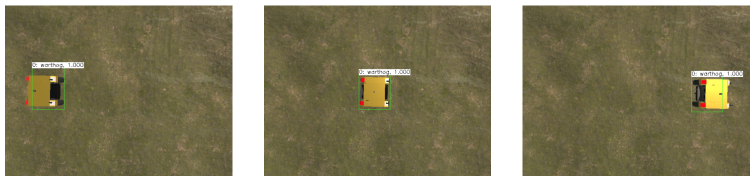


Figure 3.10: Time series of the UGV's path across the image from left to right.

3.3 Comparison to Fiducial Markers

This work demonstrates autonomous landings without the use of fiducial markers, and thus a brief comparison between the presented localization approach and the use of fiducial markers is presented in the subsequent sections. Two points of comparison are investigated, the computational expense of each method as well as the accuracy achieved.

3.3.1 Computational Expense

Computational expense of each method can be difficult to compare as the performance achieved is heavily dependent on the hardware used. Difficulty aside, this section aims to provide a brief comparison on the performance achieved with both an ArUcO fiducial marker and detection localization on two different hardware setups. The first hardware platform used for comparison is a Dell® XPS Laptop with an Intel® i7-8750H Central Processing Unit (CPU), a Nvidia® GeForce™ GTX 1050Ti Graphics Processing Unit (GPU), and 32 gigabytes of Random Access Memory (RAM). The second hardware platform is the ModalAI VOXL Flight computer. The VOXL Flight is built around the Qualcomm® Snapdragon™ 821 chip-set [132] which contains Qualcomm® Kyro™ CPU, a Qualcomm® Adreno™ 530 GPU, and 4 gigabytes of RAM.

The time it takes each method to complete a single iteration on the two computers was recorded and is displayed in Table 3.1. In table 3.1 DL is sued to designate the presented detection localization method. The time column indicates the time required for each method to complete a single iteration of the respective algorithm. The use of ArUcO fractal fiducial markers was run on both the workstation laptop, and the VOXL CPU. Detection Localization includes the same hardware tests with the addition of GPU acceleration for the TensorFlow Lite object detection. GPU was not utilized for any other tests as TensorFlow Lite models are not supported on desktop GPUs, and GPU acceleration is not supported for the ArUcO ROS package used for fiducial marker detection.

Table 3.1: Computational Expense of Detection Localization vs. ArUcO Fractal Marker

Hardware	**Aruco-CPU (ms)**	**DL-CPU (ms)**	**DL-GPU (ms)**
Dell Laptop	17	156	N/A
VOXL	44	1984	49

It can be observed in Table 3.1 that the performance of detection localization on the VOXL's mobile grade CPU is much slower than the Fractal ArUcO marker. However, if GPU acceleration is available, then the compute time between the two is similar with the ArUcO tag only being a few milliseconds faster.

3.3.2 Accuracy

Experiments were run within the simulation environment to compare the accuracy achieved with the ArUcO fiducial marker, and the presented detection localization method. The attitude estimate of the target was not considered in this comparison as detection localization does not produce attitude estimates of the target. Simulation was ideal for these experiments as ground truth data describing the transformation between the UGV and the UAS was easily available. Simulation also offers controlled and repeatable visuals, ensuring that changes in lighting or other external factors between experiments does not affect the data collected. Three experiments with the same format as described in Figures 3.6, 3.8, and 3.10 were performed with both ArUcO and detection localization running simultaneously. This was possible as the object detection method was able to detect the UGV even with an ArUcO marker on top. The collected results from each method were then plotted against the ground truth data from the simulation environment. The altitude of the UAS during the previous experiments was five meters. However, in these experiments the altitude had to be lowered to three meters. This was caused by the low resolution of the simulated camera struggling to detect and process the ArUcO marker from higher altitudes. The simulated camera's low resolution was a constraint caused by the compute power available on the workstation computer used for this work. Figure 3.11 presents data collected as the UAS hovered over the UGV at an altitude three meters. Figures 3.12, and 3.13 present the data collected in simulations where the UAS hovered above the UGV at an altitude of three meters while the UGV moved through the image frame from bottom to top, and left to right respectively. In some cases it may appear that the data generated with the ArUcO marker is noisier than the detection localization data. This can be attributed to detection localization's use of the filtered altitude estimate from the PX4 flight controller. ArUcO does not rely on any information from the flight controller, and thus its produced data is unfiltered.

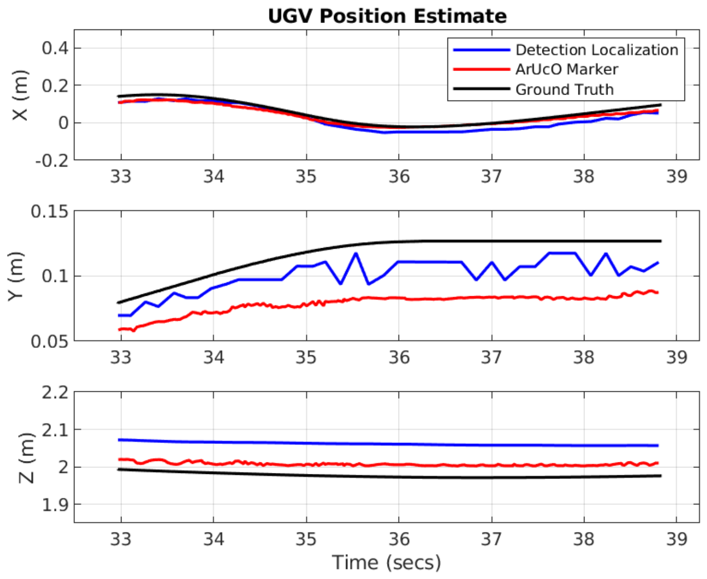


Figure 3.11: Comparison of detection localization and ArUcO fiducial marker while UAS hovers over static UGV.

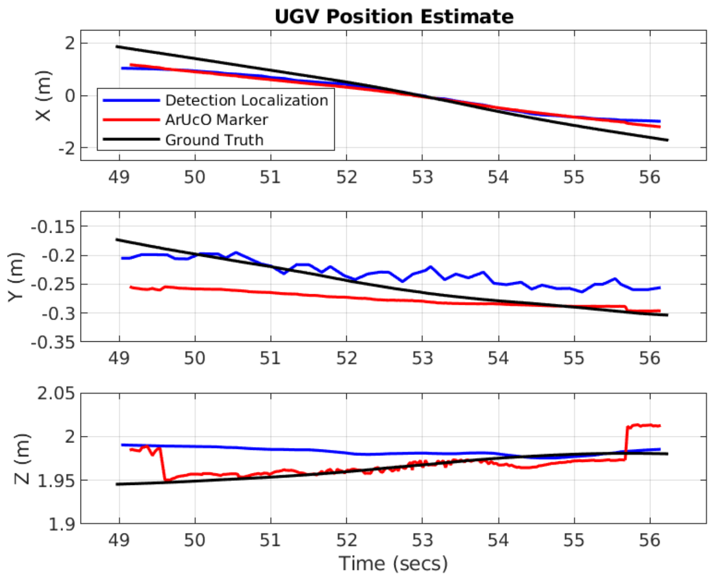


Figure 3.12: Comparison of detection localization and ArUcO fiducial marker while UAS hovers over a UGV moving from bottom to top in the image.

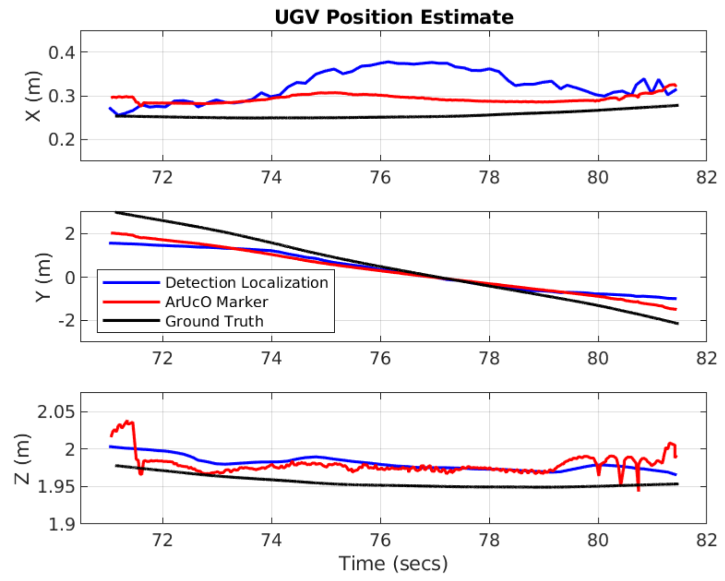


Figure 3.13: Comparison of detection localization and ArUcO fiducial marker while UAS hovers over a UGV moving from left to right in the image.

3.3.3 Comparison Summary

The main difference between fiducial markers such as an ArUcO tag and the presented detection localization method is the lack of attitude information provided by detection localization. Fiducial markers can provide a full six-degree-of freedom pose estimate of the target, while detection localization only provides a three-dimensional position estimate. Fiducial markers are also more apt for deployment on a wider range of embedded systems as they do not require the GPU acceleration that detection localization relies on for object detection. Deploying detection localization without GPU acceleration resulted in a rather slow update rate of approximately 0.5 Hz on the ModalAI VOXL Flight computer. However, with GPU acceleration available detection localization nears the update rate achieved by the ArUcO fiducial marker running on the CPU. Both methods achieve high accuracy when the target is near the center of the image and experience a decrease in accuracy as the target nears the edge of the image. In summary, the detection localization method can serve as an alternative to fiducial markers for small, low power embedded compute systems if attitude information of the target is not needed, and GPU acceleration is available on-board the embedded computer running detection localization. If deployed on a more capable embedded compute platform the GPU acceleration of object detection may not be required to run the presented detection localization method at an adequate rate to support autonomous landings.

4. COMMAND GENERATOR

4.1 Problem Definition

Detection localization yields a three-dimensional position estimate of the UGV relative to the UAS. This position estimate is to be used as input for a command generator, which generates setpoint commands for the UAS to achieve using the on-board flight controller. The command generator will need to consider the velocity of the UGV landing target as well as its position relative to the UAS to produce setpoints that result in the UAS moving closer to the target as the landing progresses. High level behavioral control of the UAS before the landing begins, during the landing, and after the landing is managed by a state machine. This chapter describes a command generator, as well as the high-level behavioral control used to facilitate the autonomous landing.

4.2 Autonomous Landing Framework

This work leverages the open source PX4 flight control software [35] running on the ModalAI® Flight Core® Flight Control Unit (FCU). More details on the hardware used can be found in Appendix A. The subsequent sections describe the PX4 flight control software, the command generator, and the state machine utilized.

4.2.1 PX4 Controller

The PX4 open-source flight control software enables multirotor UAS to fly in a variety of ways such as simple stabilized manual flight, manual flight whilst maintaining position, and end to end autonomous missions. This thesis utilizes the setpoint controller PX4 provides alongside the offboard mode functionality. Offboard mode allows a companion computer to send commands to the FCU. This functionality enables the implementation of more computationally expensive methods such as visual servoing. PX4 is responsible for all control of the UAS before, and during the autonomous landing. A brief overview of the PX4 controller used is provided in the following subsection.

4.2.1.1 PX4 Setpoint Controller

The PX4 setpoint controller utilizes a cascaded control architecture, where the controllers used are a mix of Proportional (P) and Proportional Integral Derivative (PID) controllers. The first set of controllers evaluates position errors in the inertial frame and iterates at a rate of 50 Hz. A vector of cartesian points, a setpoint $\mathbf{X_{sp}}$, is passed into a P position controller that outputs a velocity vector $\mathbf{V_{sp}}$. Figure 4.1 visually depicts this position controller. PX4 saturates the commanded velocity to ensure the commanded velocity lies within reasonable limits. The command velocity vector is passed into a PID velocity controller which outputs an acceleration vector $\mathbf{A_{sp}}$, as shown in Figure 4.2. This PID loop utilizes a low pass filter (LPF) to reduce noise on the derivative path. The integrator path implements an anti-reset windup to avoid the term from accumulating a significant error if a large change occurs in the velocity setpoint. Similar to the velocity command, these acceleration values are saturated to ensure they lie within reasonable limits. The inputted yaw setpoint ψ_{sp} and $\mathbf{A_{sp}}$ are then translated by PX4 from acceleration and yaw, into attitude, which outputs a quaternion $\mathbf{\tilde{q}_{sp}}$ and a thrust command δT_{sp}. The vector component of $\mathbf{\tilde{q}_{sp}}$ is passed into a 250Hz P control loop based on the work presented in [133]. The attitude control loop is responsible for translating the inputted vector portion of the quaternion into angular rate commands $\mathbf{\Omega_{sp}}$, as shown in figure 4.3. In similar fashion to prior loops, these angular rate commands are saturated to ensure they lie within a physically reasonable bound. The angular rate commands are passed into the final control block, a 1 kHz K-PID angular rate controller depicted in Figure 4.4. K-PID controllers utilize a form that is mathematically equivalent to standard PID but provide an advantage by decoupling the proportional gain tuning from the integral and derivative gains [35]. This allows the gains tuned for one UAS to be transferred to a new UAS of similar physical dimensions by simply adjusting the K gain instead of re-tuning each of the proportional, integral, and derivative gains. The angular rate controller also limits integral term authority to prevent wind up and uses a LPF on the derivative path to reduce noise. Unlike the other blocks, this block does not saturate the commands, as this is handled in a subsequent block. This final block outputs three commands δA_{sp}, δE_{sp}, and δR_{sp}. These three commands along with the aforementioned δT_{sp} correspond to the main controls that would be used on a fixed wing aircraft: ailerons, elevators, rudder, and thrust. PX4 uses a mixer to translate these commands into a vector of thrust commands for each of the rotors on the vehicle, $\mathbf{T_{sp}}$. The mixer is also responsible for bounding these commands and ensuring they do not exceed the capabilities of the UAS. Figure 4.5 visually represents how all the aforementioned controllers are cascaded for use in PX4 to achieve position setpoint control. Figures 4.1 - 4.5 are reprinted from PX4's documentation [35].

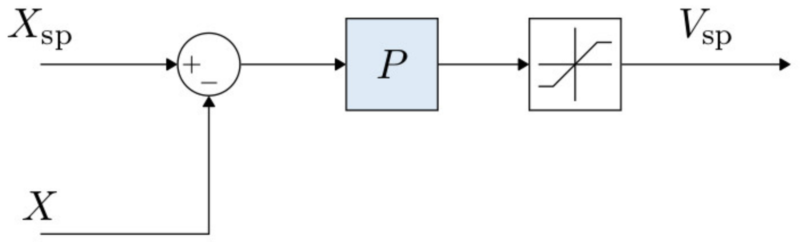


Figure 4.1: PX4 Multicopter Position Control Diagram; reprinted from [35].

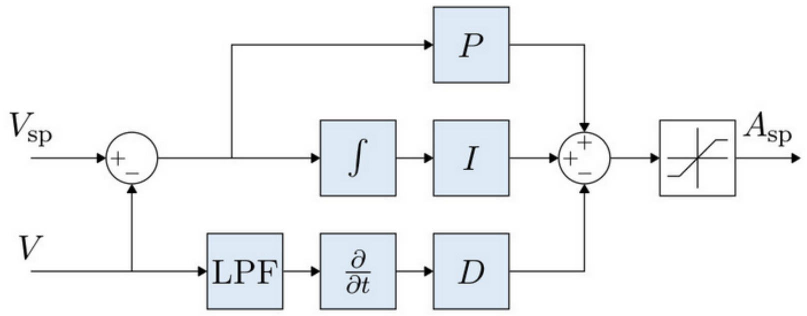


Figure 4.2: PX4 Multicopter Velocity Control Diagram; reprinted from [35].

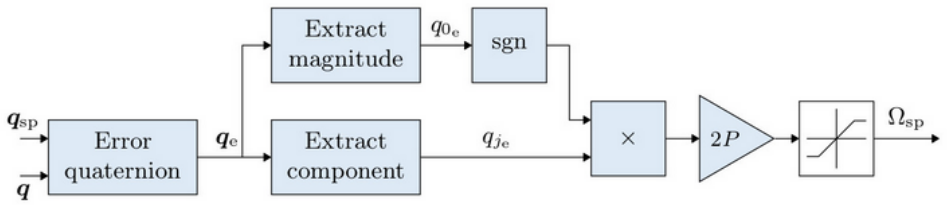


Figure 4.3: PX4 Multicopter Attitude Control Diagram; reprinted from [35].

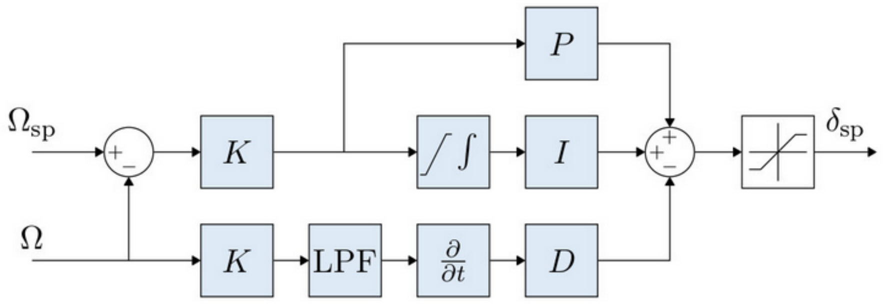


Figure 4.4: PX4 Multicopter Angular Rate Control Diagram; reprinted from [35].

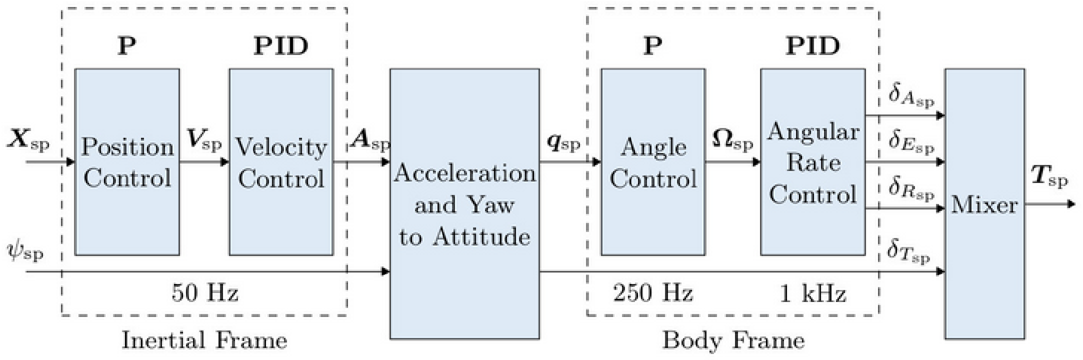


Figure 4.5: PX4 Multicopter Architecture Control Diagram; reprinted from [35].

4.2.2 Command Generator

The command generator employed is responsible for taking in position estimates of the UGV relative to the UAS, and outputting setpoints for the PX4 flight controller in a fixed inertial frame. While alternative methods such as fiducial markers can yield a full six DOF pose of the marker relative to the UAS, the detection localization scheme discussed only provides position estimates of the UGV.

The command generator used in this work is a modified version of the landing algorithm presented in [11] which was developed by Army Research Laboratory. In [11] fiducial markers are used for localization of the UGV, and the inertial position of the UAS is unknown. Modifications were made as necessary to account for the known UAS inertial position provided via GPS, and the switch from fiducial markers to the detection localization scheme used in this thesis.

4.2.2.1 Frames and Transformations

The four frames considered within this work are b the UAS body frame, n an inertial frame, c the camera frame, and ugv the estimated frame of the UGV landing target. The frame n exists in the east-north-up (ENU) coordinate frame, b and ugv exist in the forward-left-up (FLU) frame, and c is rotated from b such that the z direction of c is aligned with the cameras lens. n is fixed at the location where the UAS was powered on, and b is centered on the UAS center of gravity and rotates about all axis. ugv is centered on the landing pad and is rotated about its z axis to align with the heading of the UGV. ugv is fixed in rotation about its x and y axis. The resulting transformation from ugv to n is defined in Equation 4.1. Figure 4.6 visually depicts the frames utilized in this work.

$$\boldsymbol{T}^{n}_{ugv} = \boldsymbol{T}^{n}_{b}\boldsymbol{T}^{b}_{c}\boldsymbol{T}^{c}_{ugv} \tag{4.1}$$

Throughout the entire landing mission the transformation between n and b, $\boldsymbol{T}^{n}_{b}$, is maintained by PX4 and is always known. The transformation between b and c, $\boldsymbol{T}^{b}_{c}$, is static as the camera is assumed to be rigidly fixed to the UAS, not gimballed. Upon detection of the ground target, the

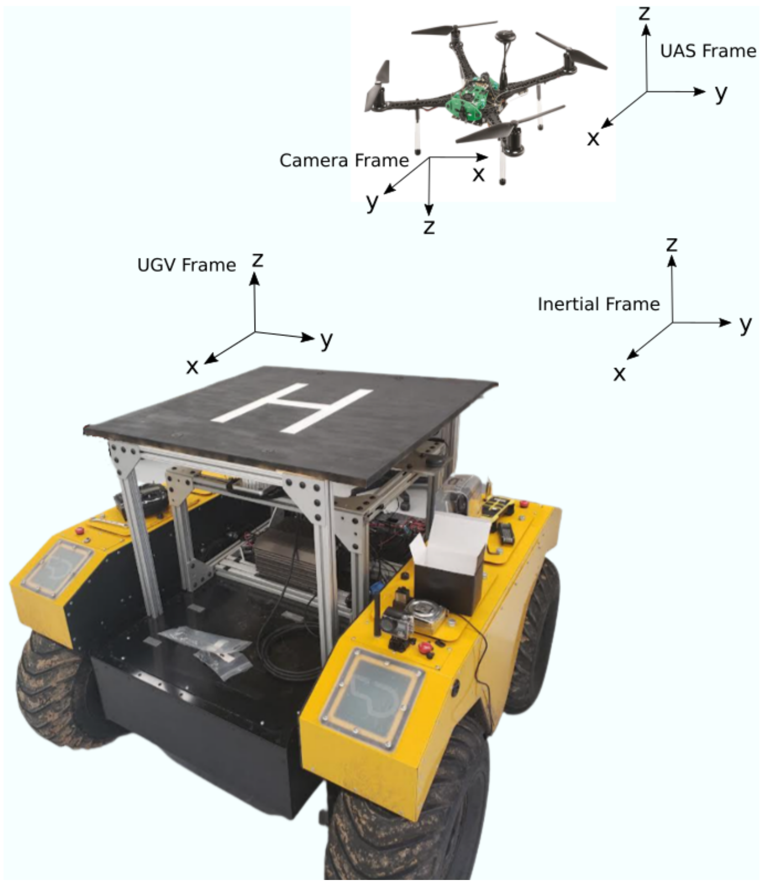


Figure 4.6: Visual depiction of the four frames defined in this thesis.

transformation from n to ugv is defined in Equation 4.1 where $\boldsymbol{T^c_{ugv}}$ is provided from the detection localization. A Kalman Filter is utilized in the maintenance of $\boldsymbol{T^n_{ugv}}$ in an attmept to filter any noise present from the detection localization or object detection methods implemented. Details of this Kalman Filter are discussed in the subsequent subsection 4.2.2.3

4.2.2.2 Setpoint Definition

Setpoints for the UAS need be sent in a fixed frame, thus this work publishes all setpoints in the inertial frame n. As mentioned above, this work utilizes the PX4 control stack. Specifically, by passing setpoints to PX4 that when achieved result in a successful landing of the UAS on the moving UGV landing target. These setpoints sent to the PX4 flight controller contain an inertial position setpoint vector $\boldsymbol{P_{sp}}$, an inertial velocity setpoint vector $\boldsymbol{V_{sp}}$, and an inertial yaw setpoint

ψ_{sp}. This setpoint to be commanded, $\boldsymbol{S}$, is defined in Equation 4.2.

$$\boldsymbol{S} = [\boldsymbol{P_{sp}}\ \boldsymbol{V_{sp}}\ \psi_{sp}],\ where\ \boldsymbol{P_{sp}} = [p_x\ p_y\ p_z]\ and\ \boldsymbol{V_{sp}} = [v_x\ v_y\ v_z] \tag{4.2}$$

4.2.2.3 Command Generator Algorithm

This algorithm begins running upon five successive UGV detections to ensure a single false detection does not start the landing process before the UGV is in the image frame. Detection localization provides estimates of the UGVs position relative to the UAS camera, and this position is transferred into the inertial frame via the transformation described in Equation 4.1. The inertial UGV position estimate, $\boldsymbol{P^{n}_{ugv}}$, is utilized by a Kalman Filter [134] to estimate the UGVs position and velocity in the inertial frame. Specifically, a kinematic, three-dimensional, first order (i.e. constant velocity kinematics) Kalman Filter was used from the package FilterPy [135]. Higher order kinematic filters such as constant acceleration and constant jerk were also evaluated, but performed worse even when the UGV was moving with a non-constant velocity. This is likely due to the UGVs velocity changing on a much slower time scale compared to the Kalman Filter's iteration rate, resulting in the velocity being near constant between Kalman Filter updates. Equation 4.3 describes the states of the filter, Equation 4.4 displays the state transition matrix of this filter F, and the process noise covariance matrix Q. Derivations of F and Q can be found in Appendix B. Simulation and experimental results determined the filter performs well when the ratio of σ over Δt is approximately 15.

$$\boldsymbol{x} = \begin{bmatrix} x & \dot{x} & y & \dot{y} & z & \dot{z} \end{bmatrix} \tag{4.3}$$

$$F = \begin{bmatrix} 1 & \Delta t & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & \Delta t & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & \Delta t \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}, Q = \begin{bmatrix} \frac{\Delta t^4}{4} & \frac{\Delta t^3}{2} & 0 & 0 & 0 & 0 \\ \frac{\Delta t^3}{2} & \Delta t^2 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{\Delta t^4}{4} & \frac{\Delta t^3}{2} & 0 & 0 \\ 0 & 0 & \frac{\Delta t^3}{2} & \Delta t^2 & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{\Delta t^4}{4} & \frac{\Delta t^3}{2} \\ 0 & 0 & 0 & 0 & \frac{\Delta t^3}{2} & \Delta t^2 \end{bmatrix} \sigma^2 \tag{4.4}$$

Upon startup of the autonomous landing no prior information about the UGVs velocity is available. Thus, the Kalman Filter velocity states are initialized to zero, and the position states are initialized to the first position estimate from detection localization: $\begin{bmatrix} x_{ugv0} & y_{ugv0} & z_{ugv0} \end{bmatrix}$. The state error covariance and measurement noise covariance matrices are initialized to identity. The initialization details are summarized in Equation 4.5.

$$\boldsymbol{\hat{x}_{0,0}} = \begin{bmatrix} x_{ugv0} & 0 & y_{ugv0} & 0 & z_{ugv0} & 0 \end{bmatrix}, \quad P_{0,0} = I_{6x6}, \quad and \quad R = I_{3x3} \tag{4.5}$$

After initialization, the states and state error covriance matrix are predicted as shown in Equation 4.6 and 4.7 respectively.

$$\boldsymbol{\hat{x}_{1,0}} = F\boldsymbol{\hat{x}_{0,0}} \tag{4.6}$$

$$P_{1,0} = FP_{0,0}F^T + Q \tag{4.7}$$

The filters measurements $\boldsymbol{z}$ and the measurement matrix H are defined in Equation 4.8. Upon receipt of a measurement, the Kalman gain K is computed (Equation 4.9) and used to update the states (Equation 4.10) and the state error covariance P (Equation 4.11).

$$\boldsymbol{z} = \begin{bmatrix} x_{ugv} & y_{ugv} & z_{ugv} \end{bmatrix}, \quad and \quad H = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix} \tag{4.8}$$

$$K_1 = P_{1,0} H^T (H P_{1,0} H^T + R)^{-1} \tag{4.9}$$

$$\boldsymbol{\hat{x}_{1,1}} = \boldsymbol{\hat{x}_{1,0}} + K_1(\boldsymbol{z_1} - H\boldsymbol{x_{1,0}}) \tag{4.10}$$

$$P_{1,1} = (I - K_1 H) P_{1,0} (I - K_1 H)^T + K_1 R K_1^T \tag{4.11}$$

After updating, the states and covariances are used to complete prediction again (Equation 4.12 and Equation 4.13). This Kalman Filter provides estimates of the UGV's position and velocity while mitigating effects of any noisy data output by detection localization.

$$\boldsymbol{\hat{x}_{2,1}} = F\boldsymbol{\hat{x}_{1,1}} \tag{4.12}$$

$$P_{2,1} = F P_{1,1} F^T + Q \tag{4.13}$$

The inertial UGV position and velocity estimates output by the Kalman Filter are denoted $\boldsymbol{V^n_{ugv}}$ and $\boldsymbol{P^n_{ugv}}$ respectively. $\boldsymbol{V^n_{ugv}}$ is used to estimate the inertial heading angle of the UGV, ψ. The estimate of ψ also serves as the as the yaw setopint sent to the flight controller, ψ_{sp}. The UGV in this work is a wheeled vehicle, thus it is assumed that the the UGV's velocity vector's orientation is equivalent to the heading angle of the UGV, ψ. This assumption proved reliable if the vehicle is only given a forward or reverse throttle command. If the UGV changes from forward to reverse quickly the resulting large oscillations in ψ hinder the landing performance. ψ is measured from the x-axis of n such that positive ψ represents a counter-clockwise rotation about the z-axis of

n. To estimate the heading of the UGV, the sign of the x and y components of $\boldsymbol{V_{ugv}^n}$ were used to feed logic filters that determined which quadrant of the XY-plane the velocity vector exists in. Then the appropriate equation was selected for estimating heading angle in the ENU frame n. Experimental and simulation results showed that $\boldsymbol{V_{ugv}^n}$ often oscillated within the first three to five seconds of the Kalman Filter running. A four second delay is enforced on the publication of ψ_{sp} to the flight controller to ensure that no early oscillatory values are used. This delay can be seen in Figures 4.15, 4.19, and 4.23. For hardware experiments, logic was implemented that ignores the commanded yaw value unless the magnitude of the x and y components of $\boldsymbol{V_{ugv}^n}$ exceed a threshold of 0.5 meters per second. This logic is intended to keep any noise in $\boldsymbol{V_{ugv}^n}$ from causing oscillations in ψ_{sp}. Figure 4.7 visually depicts the four different possible quadrants in which the x and y components of $\boldsymbol{V_{ugv}^n}$ can exist. Table 4.1 summarizes the equations used to estimate the heading angle where v_x and v_y denote the x and y components of $\boldsymbol{V_{ugv}^n}$ respectively.

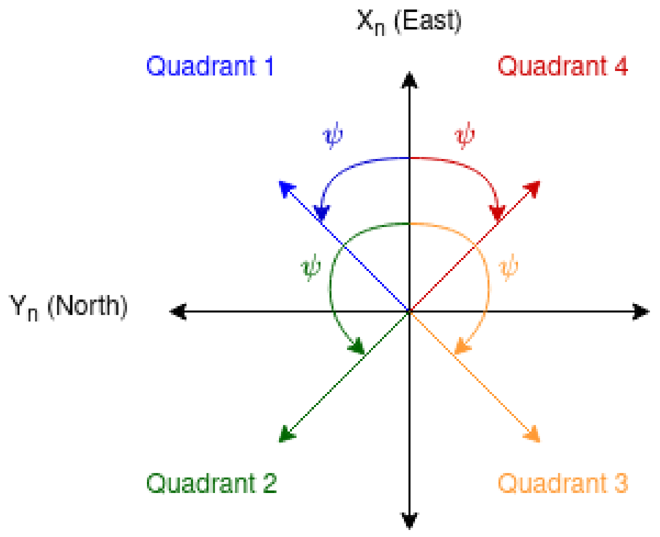


Figure 4.7: Visualization of Heading Angle Estimation

Table 4.1: Summary of Heading Angle Estimation

	Sign of v_x	Sign of v_y	Equation
Quandrant 1	$+$	$+$	$\psi = \arctan(\frac{V_y}{V_x})$
Quandrant 2	$-$	$+$	$\psi = -\arctan(\frac{V_x}{V_y}) + \frac{\pi}{4}$
Quandrant 3	$-$	$-$	$\psi = -\arctan(\frac{V_x}{V_y}) - \frac{\pi}{4}$
Quandrant 4	$+$	$-$	$\psi = \arctan(\frac{V_y}{V_x})$

$\boldsymbol{P^n_{ugv}}$ serves as the origin for the frame ugv, and ugv is rotated around it's z-axis by ψ such that the x-axis on ugv aligns with the heading of the UGV. The maintenance of the frame ugv allows for a landing trajectory to be computed relative to the landing pad, and transferred into the inertial frame for use by the flight controller. This landing trajectory is scaled in the UGVs direction of travel to help ensure the UGV remains in the UAS camera frame as the UAS pitches forward to match the UGVs velocity. Four parameters are used to define the start and endpoints of the trajectory in the XZ-plane of ugv. The first three z_0, z_f, and x_f are defined in meters and the final paramerter $x_{0offset}$ is defined in meters per meters per second such that the resulting calculated x_0 has units of meters. $x_{0offset}$ is used in conjunction with a time history average of the magnitude of $\boldsymbol{V^n_{ugv}}$, v_{avg}, to scale the trajectory in the direction of travel. To define v_{avg} a time history of the 15 most recent magnitudes of $\boldsymbol{V^n_{ugv}}$ is stored. Then an average of this vector of prior magnitudes is defined as v_{avg}. The resulting scaled x_0 is defined in Equation 4.14.

$$x_0 = x_{0offset} * v_{avg} \tag{4.14}$$

The resulting trajectory exists as a line from the coordinates (x_0,z_0) to (x_f,z_f) in the frame ugv, as depicted in Figure 4.8. The value selected for z_f coincides with the UAS altitude at which the UAS is too close to the UGV to reliably detect it using object detection. A position target,

$\boldsymbol{P_{targ}} = \begin{bmatrix} x_{targ} & 0 & z_{targ} \end{bmatrix}$, is defined as a point that translates down the trajectory over time resulting in the UAS landing on the the UGV. During the first iteration z_{targ} is defined as z_0 and during later iterations z_{targ} is updated according to Equation 4.15, where dt is the time between iterations. x_{targ}, is found by interpolation using z_{targ} and the defined trajectory line. Depending whether or not the UAS is within an acceptance sphere around $\boldsymbol{P_{targ}}$, $v_{descent}$ is defined . The acceptance sphere's radius is scaled linearly to decrease as the UAS progresses along the trajectory. If z_{targ} is greater than z_0, $v_{descent}$ is set to 2.0 meters per second. Otherwise, if z_{targ} is less than z_0 but greater than z_f, $v_{descent}$ is set to 0.5 meters per second. If the UAS leaves the acceptance sphere, $v_{descent}$ is set to 0.0 meters per second until the UAS renters the acceptance sphere around $\boldsymbol{P_{sp}}$. If the UAS is unable to detect the UGV for more than 0.5 seconds, z_{targ} is set back to z_0 and the z component of $\boldsymbol{V_{sp}}$ is set to -1.0 m/s to increase altitude to search for the UGV. Given the formulation of Equation 4.17, the velocity value of -1.0 will result in the vehicle increasing in altitude as this negative value will be subtracted. The radii values for the acceptance sphere, and the value for $v_{descent}$ were determined through trial and error in both simulation and outdoor experimentation. After $\boldsymbol{P_{targ}}$ is updated, it is transferred from the frame ugv into n. This inertial position setpoint is sent to the flight controller as $\boldsymbol{P_{sp}}$. Once the end point $\boldsymbol{P_{targ}} = \begin{bmatrix} x_f & 0 & z_f \end{bmatrix}$ is reached, the UAS enters a final descent phase as shown in Figure 4.8. During this final descent the UAS is commanded to maintain current attitude and a thrust command less than the thrust required for hover is commanded until the landing is complete.

$$z_{targ} = z_{targ} - dt * v_{descent} \tag{4.15}$$

In the event that the UGV leaves the image frame for more than 0.5 seconds, the last known position and velocity are used to extrapolated $\boldsymbol{P_{sp}}$ in an attempt to recapture the UGV in the image frame. Equation 4.16 defines this extrapolation where v_{lastx} and v_{lasty} represents the x and y components of the most recent UGV velocity estimate, $\boldsymbol{P_{last}}$ defines the last UGV position estimate, and c is predefined scalar.

$$\boldsymbol{P_{sp}} = \boldsymbol{P_{last}} + c * [v_{lastx}\ v_{lasty}\ 0.0] * dt \tag{4.16}$$

The inertial velocity estimate $\boldsymbol{V^n_{ugv}}$ is used in conjunction with $v_{descent}$ to define the inertial velocity setopint $\boldsymbol{V^n_{ugv}}$ as defined in Equation 4.17. This formulation aims to have the UAS match the UGVs velocity, and descend when appropriate.

$$\boldsymbol{V_{sp}} = \boldsymbol{V^n_{ugv}} - [0\ 0\ v_{descent}] \tag{4.17}$$

The computation of $\boldsymbol{V_{sp}}$ completes the population of $\boldsymbol{S}$. The process described in this section is iterated until landing on the UGV is achieved. Figure 4.9 visually summarizes the landing algorithm in a more technical sense, and the "Land on UGV" state in Figure 4.10 depicts the lading algorithm in a more logic based perspective.

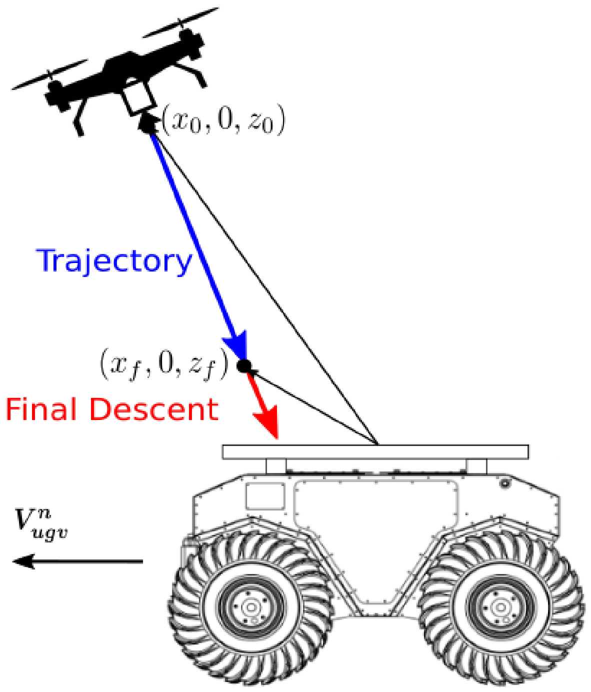


Figure 4.8: Generated landing trajectory in frame ugv.

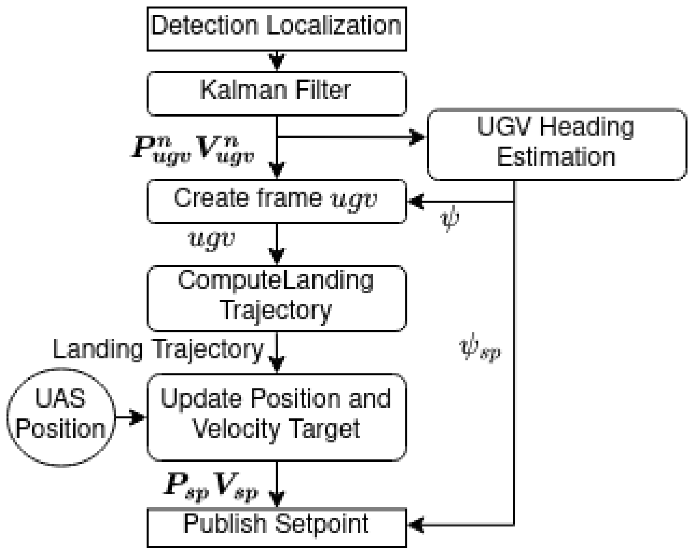


Figure 4.9: Visual summarization of the landing algorithm.

4.2.3 State Machine Description

A state machine is employed to manage high level behavioral control of the UAS. This state machine is based on the ROS package smach [136]. Smach is a ROS independent Python library that enables the hierarchical construction of state machines. In smach, the term "state" defines a local state of execution, or equivalently a "state" corresponds to the system performing a task. These smach states are constructed to provide one of a few predefined outcomes the state can achieve. The outcome of the current state determines what state the system enters next. If all states are completed, a final output of success is given when the UAS has landed on the UGV. The majority of the states exist in such a way that once completed, the state machine transitions to the next state (such as "Search for UGV" and "Land on UGV"). However, certain states such as the "Safety Checks", and "Find Pad" states are concurrent states that are continuously checking some condition. When they are successful, that is their condition is met, then the states within them are enabled to be run. For example, in Figure 4.10 the state "Find Pad" enables the state of "Land on UGV" to be executed. That is to say, "Land on UGV" succeeds "Search for UGV" is the mission hierarchy, but "Land on UGV" can only run if the condition for "Find Pad" is met. This is different than other states as the condition checked by "Find Pad" continues running even after "Land on UGV" has commenced. Meaning that even after the "Land on UGV" state has begun, if "Find Pad" fails the state will be changed to "Search for UGV" as "Land on UGV" can only be executed while "Find Tag" is successful. The same process concept is used for the passive state "Safety Checks". Table 4.2 describes the states used, their function, and their possible outputs. Figure 4.10 visually depicts the construction of the state machine where the "Land on UGV" state is expanded to show the logic used to facilitate autonomous landings.

Table 4.2: Description of State Machine States.

State	**Function**	**Possible Outcomes**
Safety Checks	Passive state constantly checking the UAS is safe to continue autonomous mission.	"fail", "success"
Check Flying	Determines if the UAS is already in flight.	"fail", "success"
Get In Air	UAS performs a takeoff.	"fail", "success"
Follow Path	UAS navigates to a defined setpoint.	"fail", "success"
Search For UGV	UAS hovers and waits for the UGV to be detected.	"found ugv", "fail"
Find Pad	Passive state constantly monitoring the transformation between UGV and UAS frames.	"fail", "success"
Land on UGV	UAS begins the autonomous landing.	"fail", "success"

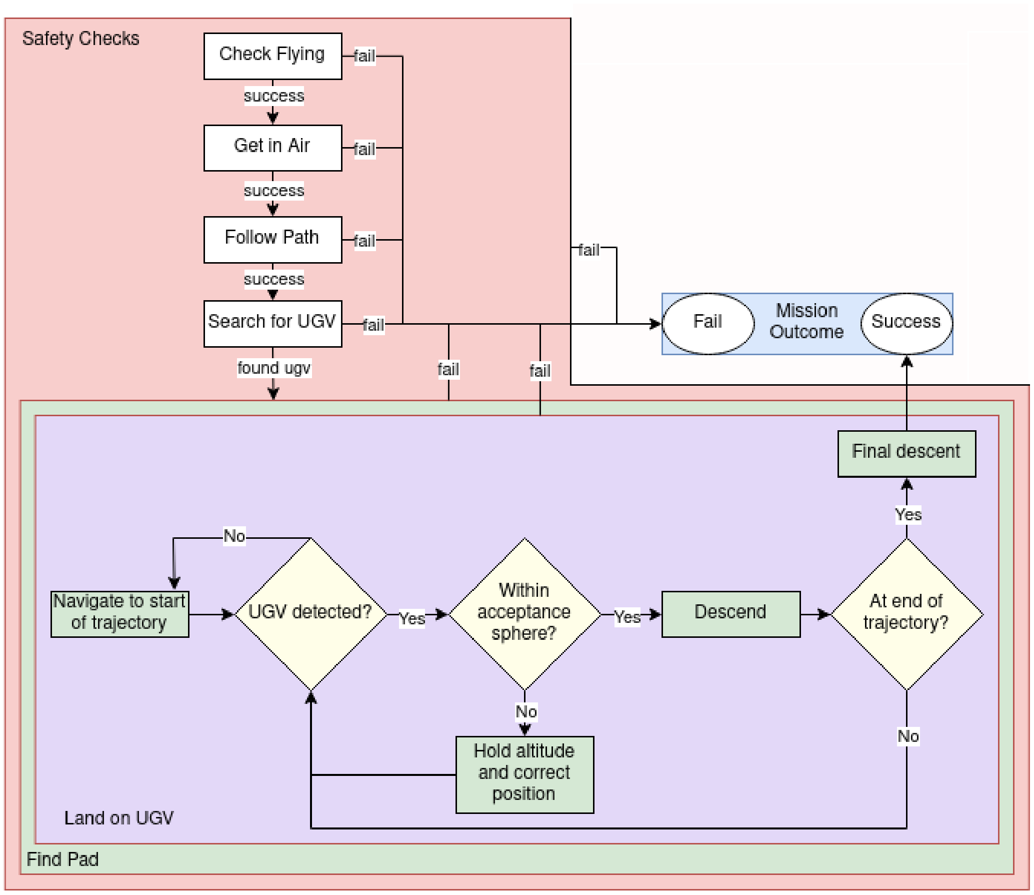


Figure 4.10: Flowchart depicting the state machine used in this work.

4.3 Landing Performance

This section presents the results achieved with the presented landing command generator. Results achieved with various UGV velocities in both simulation and during hardware experiments are presented.

4.3.1 Simulation

All simulation experiments were completed within a Unity® simulation environment [137].Within the simulation environment various UGV velocity cases were considered in the evaluation of the presented autonomous landing approach. Simulation results are shown for two cases where the UGV was traveling in a straight line at 1.0 and 3.0 meters per second. The final case shown presents results where the UGV was traveling at 1.0 m/s while turning at approximately 0.1 radians per second. Plots in the subsequent section reference UAS Position, UAS Velocity, and UAS Yaw as well as the setpoint $\boldsymbol{S}$. The values used for the UAS Position, UAS Velocity, and UAS Yaw data plots were generated from the data produced by the PX4 flight controllers Extended Kalman Filter (EKF). The setpoint data plots were generated by the command generator algorithm described in this chapter. UGV position denotes the UGVs position as reported by the simulation environment.

Table 4.4 summarizes the landing trajectory parameters used for these simulation experiments. These values were experimentally determined through trial and error. Figure 4.11 displays a time history of screenshots from the simulation environment during the first case, the UGV moving in a straight line at 1.0 meter per second. Each image contains a picture in picture view where the downward facing camera image from the UAS is overlaid on the view of both the UGV and the UAS.

Table 4.3: Command Generator Parameters for Landings in Simulation

Parameter	**Value**
$x_{0offset}$	0.25 (m/m/s)
x_f	0.10 (m)
z_0	6.0 (m)
z_f	1.25 (m)

4.3.1.1 UGV straight line at 1.0 meters per second

In this experiment the UGV was commanded to drive forward at a constant velocity of 1.0 meter per second. The UAS was commanded to takeoff to an altitude of six meters at a position

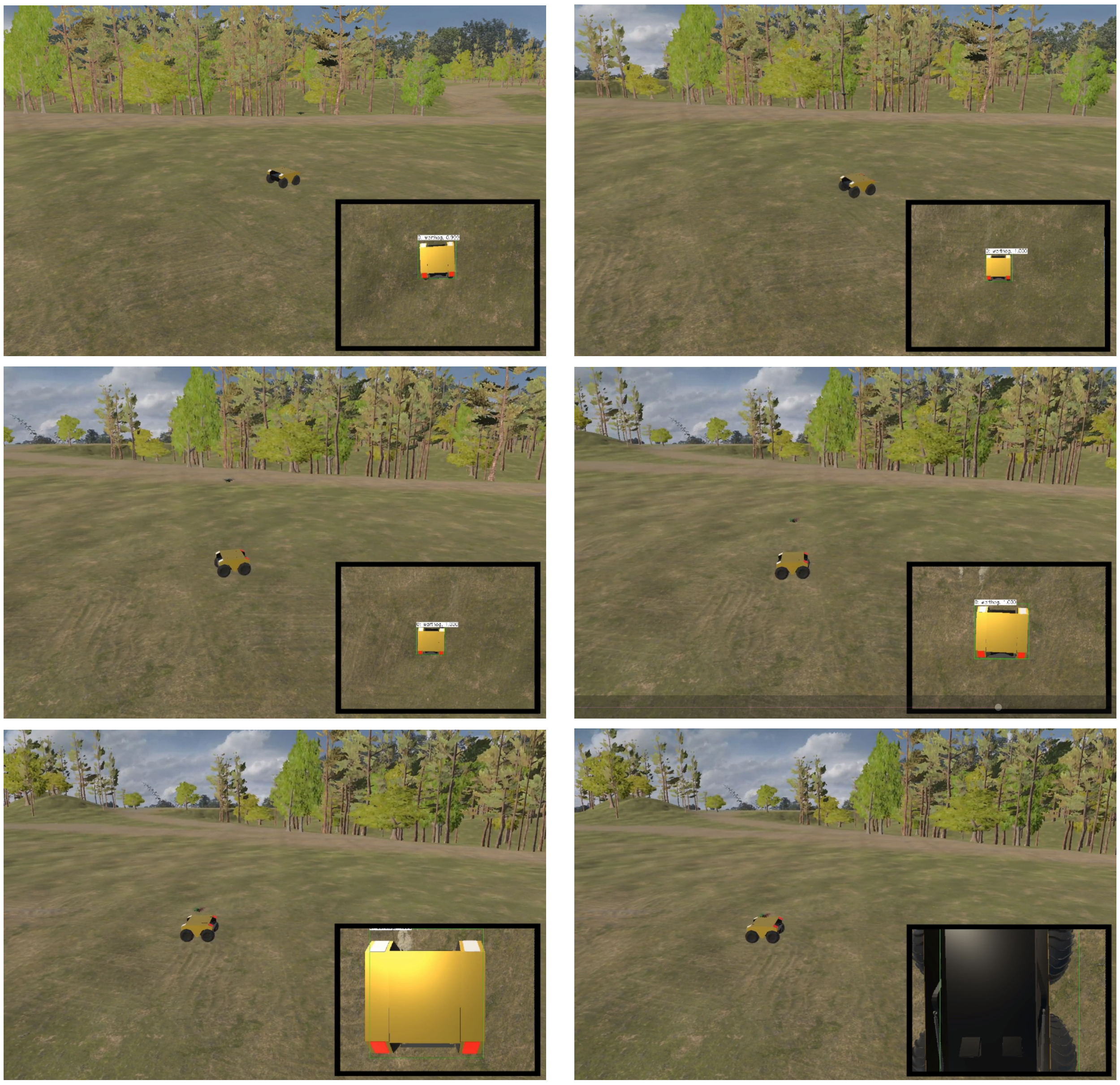


Figure 4.11: Time series of screenshots from simulation with images from UAS downward facing camera overlaid.

such that the UGV was out of the UAS image frame. The UGV was then commanded to start moving forward. Upon the UGV entering the bottom of the UAS image frame, the autonomous landing began. Figure 4.12 displays a 3D plot overlaying the UAS position, $\boldsymbol{P_{sp}}$, and the UGVs path. Figure 4.13 presents the x, y, and z components of both the UAS position, as well as the x, y, and z components of $\boldsymbol{P_{sp}}$. In similar fashion figure 4.14 presents the components of UAS velocity as well as the components of $\boldsymbol{V_{sp}}$, and finally Figure 4.15 presents the UAS yaw angle and ψ_{sp}.

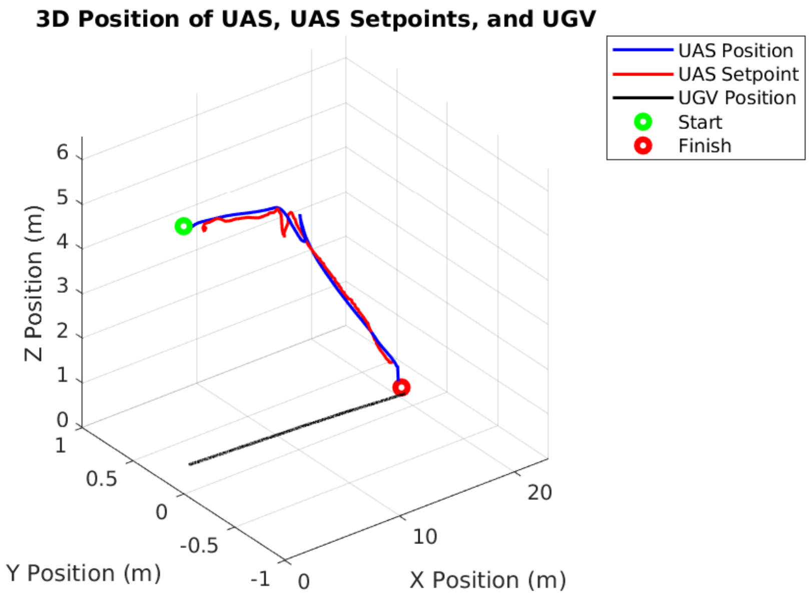


Figure 4.12: 3D Plot of the UAS position, commanded setpoint, and UGV path during moving landing with UGV speed of 1.0 m/s.

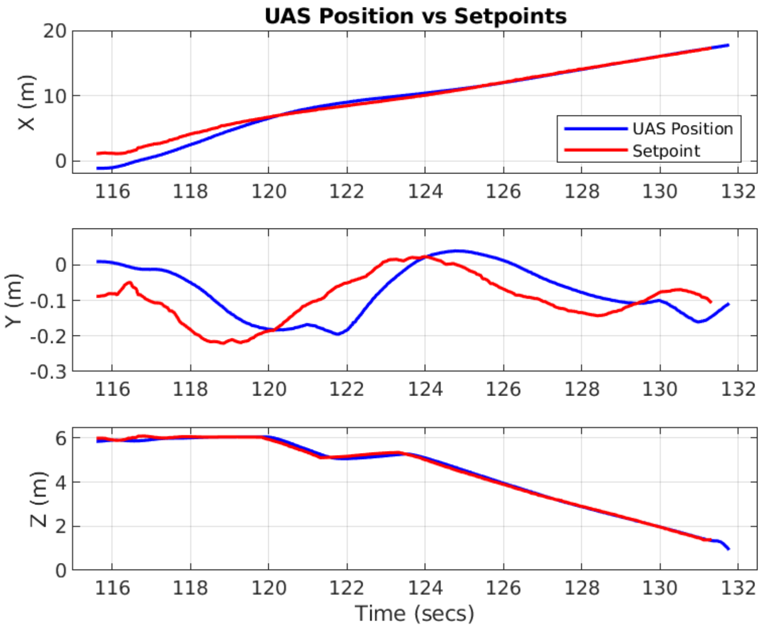


Figure 4.13: UAS position vs commanded position setpoint $\boldsymbol{P_{sp}}$ during moving landing with UGV speed of 1.0 m/s.

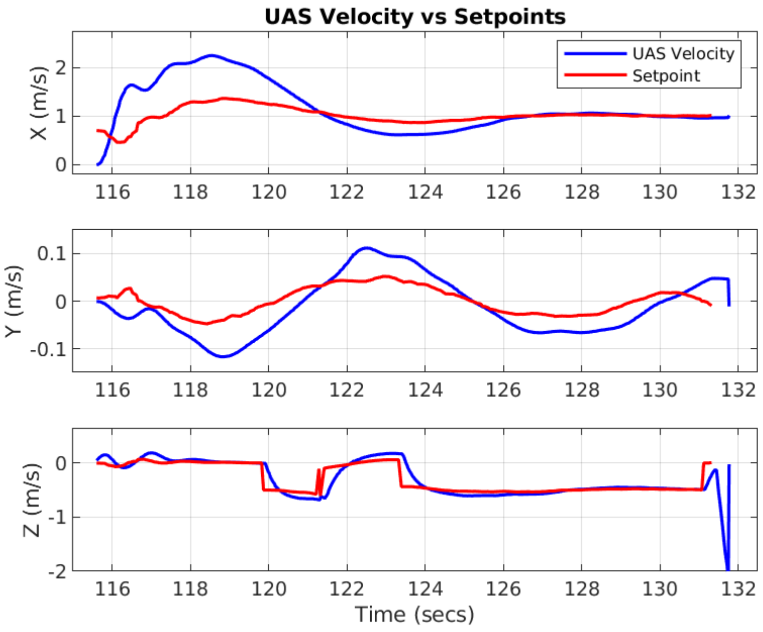


Figure 4.14: UAS velocity vs commanded velocity setpoint $\boldsymbol{V_{sp}}$ during moving landing with UGV speed of 1.0 m/s.

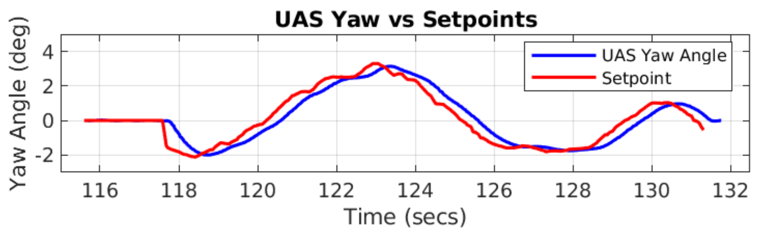


Figure 4.15: UAS yaw angle vs commanded yaw setpoint ψ_{sp} during moving landing with UGV speed of 1.0 m/s.

4.3.1.2 UGV straight line at 3.0 meters per second

In this experiment the UGV was commanded to drive forward at a constant velocity of 3.0 meters per second. The UAS was commanded to takeoff to an altitude of six meters at a position such that the UGV was out of the UAS image frame. The UGV was then commanded to start moving forward. Upon the UGV entering the bottom of the UAS image frame, the autonomous landing began. Figure 4.16 displays a 3D plot overlaying the UAS Position, $\boldsymbol{P_{sp}}$, and the UGVs path. Figure 4.17 presents the x, y, and z components of both the UAS position, as well as the x, y, and z components of $\boldsymbol{P_{sp}}$. In similar fashion figure 4.18 presents the components of UAS velocity as well as the components of $\boldsymbol{V_{sp}}$, and finally Figure 4.19 presents the UAS yaw angle and the yaw setpoint ψ_{sp}.

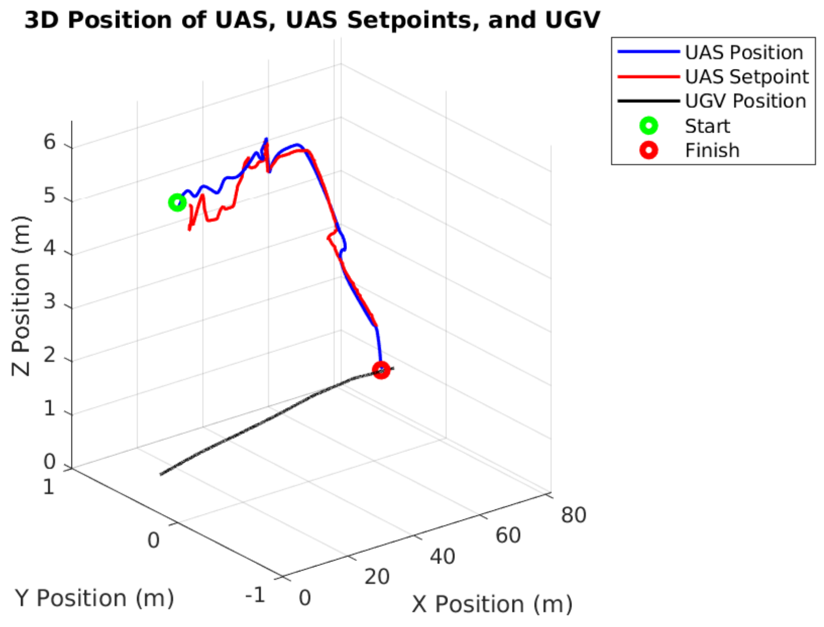


Figure 4.16: 3D Plot of the UAS position, commanded setpoint, and UGV path during moving landing with UGV speed of 3.0 m/s.

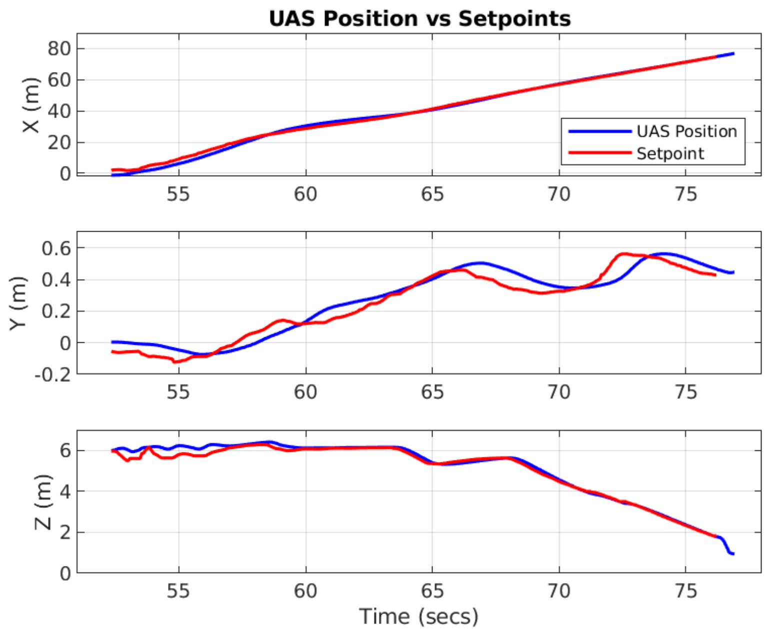


Figure 4.17: UAS position vs commanded position setpoint $\boldsymbol{P_{sp}}$ during moving landing with UGV speed of 3.0 m/s.

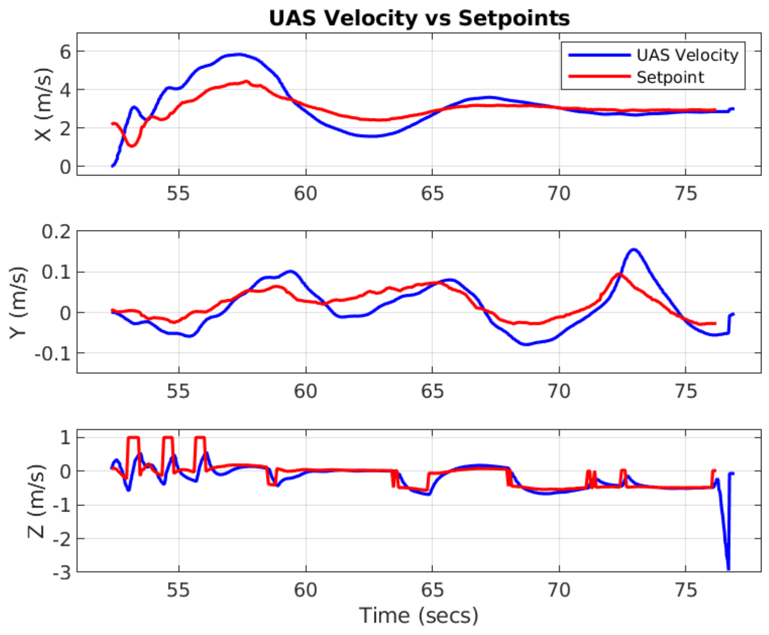


Figure 4.18: UAS velocity vs commanded velocity setpoint $\boldsymbol{V_{sp}}$ during moving landing with UGV speed of 3.0 m/s.

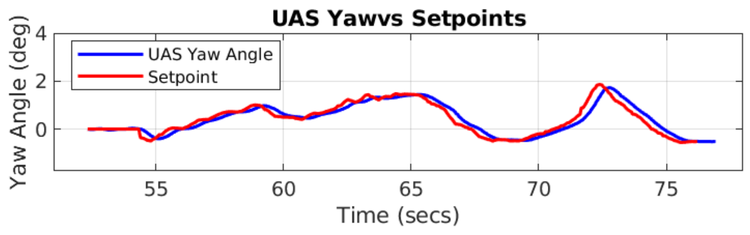


Figure 4.19: UAS yaw angle vs commanded yaw setpoint ψ_{sp} during moving landing with UGV speed of 3.0 m/s.

4.3.1.3 UGV traveling in circular path

In this experiment the UGV was commanded to drive in a circular path. Specifically the UGV was commanded to maintain a body axis forward velocity of 1.0 meter per second and a body axis angular velocity of 0.1 radians per second. This combination of translational and angular velocity commands resulted in the UGV traveling on a circular path with a radius of approximately 20 meters. The UAS was commanded to takeoff to an altitude of six meters at a position such that the UGV was out of the UAS image frame. The UGV was then commanded to start moving. Upon the UGV entering the bottom of the UAS image frame, the autonomous landing began. Figure 4.20 displays a 3D plot overlaying the UAS Position, $\boldsymbol{P_{sp}}$, and the UGVs path. Figure 4.21 presents the x, y, and z components of both the UAS position, as well as the x, y, and z components of $\boldsymbol{P_{sp}}$. In similar fashion figure 4.22 presents the components of UAS velocity as well as the components of $\boldsymbol{V_{sp}}$, and finally Figure 4.23 presents the UAS yaw angle and ψ_{sp}.

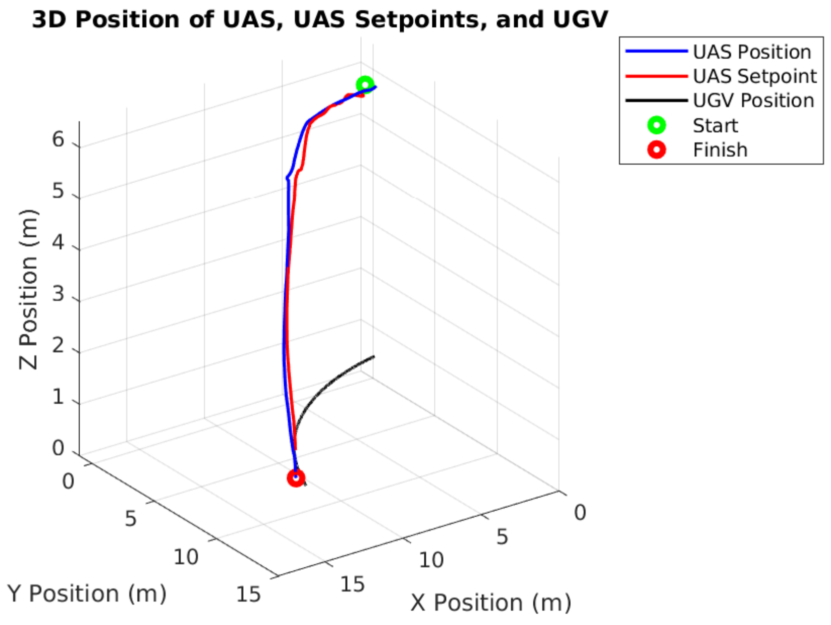


Figure 4.20: 3D Plot of the UAS position, commanded setpoint, and UGV path during moving landing with UGV traveling in a circular path.

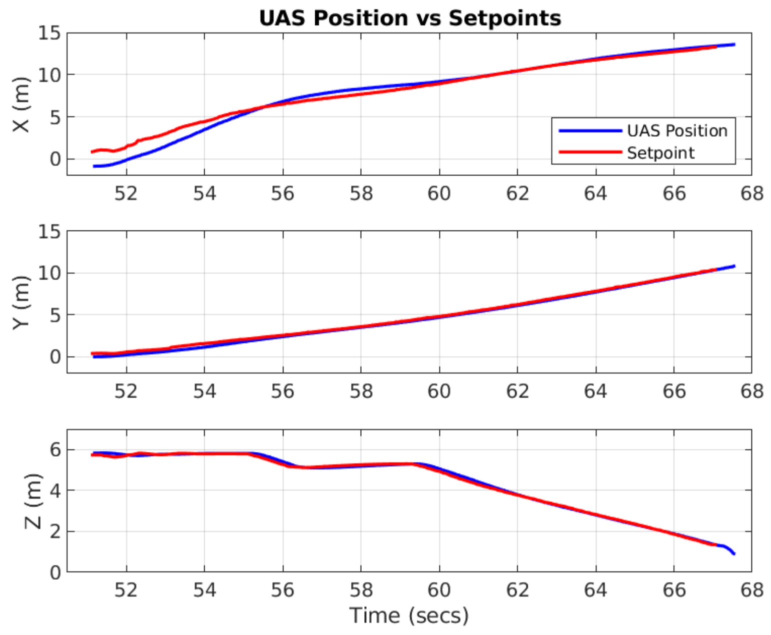


Figure 4.21: UAS position vs commanded position setpoint $\boldsymbol{P_{sp}}$ with the UGV traveling in a circular path.

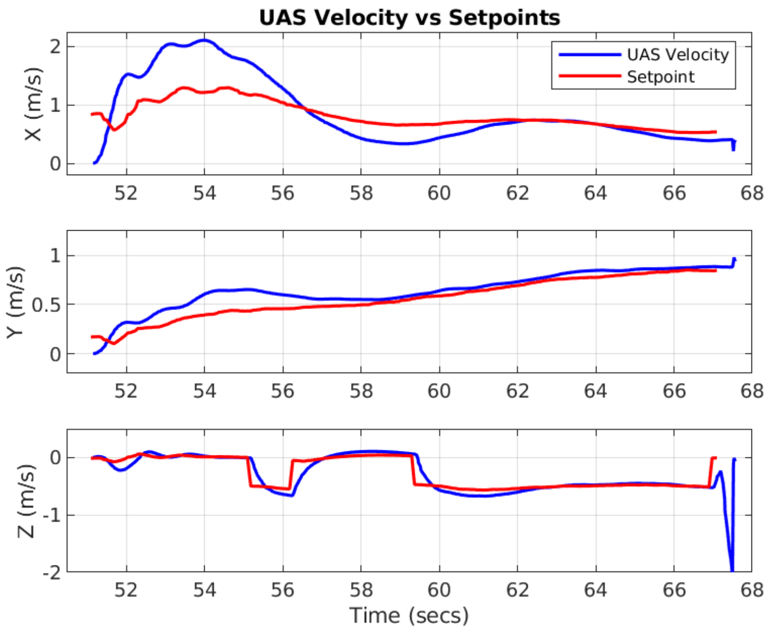


Figure 4.22: UAS velocity vs commanded velocity setpoint $\boldsymbol{V_{sp}}$ with the UGV traveling in a circular path.

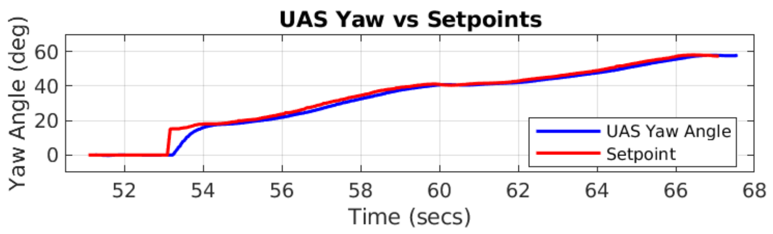


Figure 4.23: UAS yaw angle vs commanded yaw setpoint ψ_{sp} with the UGV traveling in a circular path.

4.3.1.4 Discussion

The results in the previous subsections depict successful landings in simulation with UGV speeds up to 3.0 meters per seconds. These results verify the ability to detect and localize the UGV at a sufficient enough rate to complete autonomous landings on the UGV while aligning the UAS and UGV heading. Inspection of the z components of Figure 4.17 and 4.18 at approximately 55 seconds depicts the reset of z_{targ} to z_0 and the increase of the z component of $\boldsymbol{V_{sp}}$ to 1.0 meters per second in an attempt to increase altitude and search for the UGV upon loss of UGV detections. Pauses in the descent as the UAS enters and exits the acceptance sphere around P_{targ} are also observable in the z components of Figures 4.14, 4.18, and 4.22. These results also verify that the Kalman Filter's ability to estimate the UGVs velocity as well as the utilization of this velocity estimate to estimate the UGVs heading angle, ψ. Landings at speeds greater than 3.0 meters per second were not tested but not successful in the space contained in the simulation environments. The UAS was able to track the UGV at higher speeds but was not able to complete a landing within the space of the simulation environment used.

4.3.2 Hardware

Hardware experiments were conducted in a similar fashion to the experiments in simulation. Results are presented for two different cases. Both cases had the UGV moving at 1.0 m/s, the first had the UGV travel in a straight path while the second had the UGV turn to travel on an arced path. The landing work presented became unreliable at UGV speeds greater than approximately 1.0-1.5

m/s, thus no results for faster experiments are included. Experiments were completed with both a Warthog UGV as well as the cart with the pad pictured in Figures 2.8 and 2.9 respectively. In this section the plots denoting the UGV paths are approximations of the path the UGV took as UGV data wasn't available. The values used for the UAS Position, UAS Velocity, and UAS Yaw data plots were generated from the data produced by the PX4 flight controllers EKF. The setpoint data plots were generated by the command generator algorithm described in this chapter. Discrepancies in commanded altitude and the UAS response in plots can be observed. For example, in some cases the UAS does not achieve its commanded takeoff altitude. Altitude estimation completed by PX4 was often erroneous causing these discrepancies.

Table 4.4 summarizes the landing parameters used for these outdoor hardware experiments. Figure 4.24 displays a time history of images from experiments with the Warthog UGV, and Figure 4.25 displays a time history of images from experiments with the cart and landing pad. Each time stamp contains a picture in picture view where the downward facing camera's object detection feed from the UAS is overlaid on the view of both the UGV and the UAS.

Table 4.4: Command Generator Parameters for Landings in Simulation

Parameter	**Value**
$x_{0offset}$	0.25 (m/m/s)
x_f	0.10 (m)
z_0	3.25 (m)
z_f	1.0 for UGV, 0.5 for UGV surrogate (m)

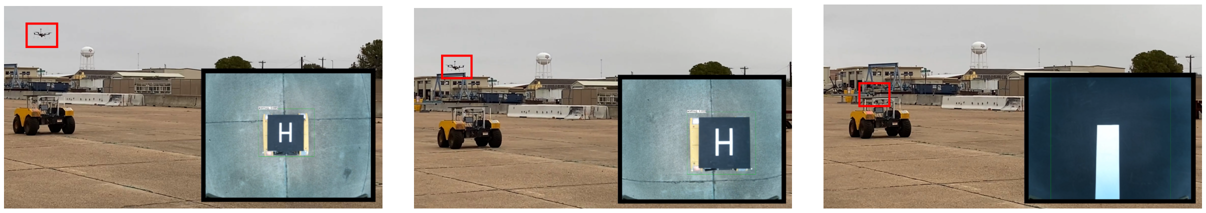


Figure 4.24: Visual time series of a landing experiment with the Warthog UGV. The UAS is marked by the red box.

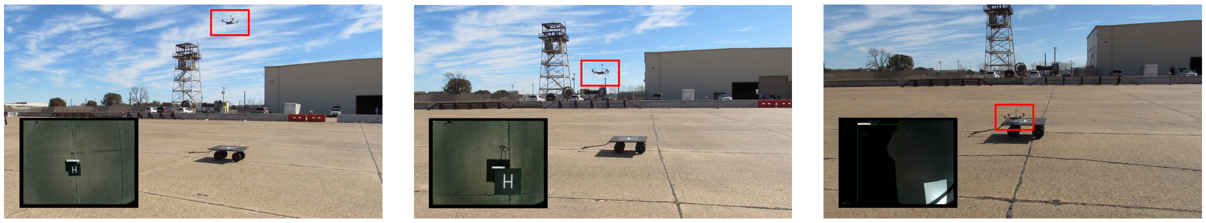


Figure 4.25: Visual time series of a landing experiment with the cart with landing pad attached. The UAS is marked by the red box.

4.3.2.1 UGV straight line 1.0 meters per second

In this experiment the cart with landing pad was pulled forward via rope in a straight line at as close to 1.0 m/s as possible. The UAS was commanded to takeoff to an altitude of four meters at a position such that the UGV was in the UAS image frame. In this case, the UAS was approximately 1.0 meter to the right of, and 0.5 meters behind the UGV. The state machine was then commanded to start the landing, and the cart began moving forward. Figure 4.30 displays a 3D plot overlaying the UAS Position, $\boldsymbol{P_{sp}}$, and a manual recreation of the UGV's approximate path. Figure 4.31 presents the x, y, and z components of both the UAS position, as well as the x, y, and z components of $\boldsymbol{P_{sp}}$. In similar fashion figure 4.32 presents the components of UAS velocity as well as the components of $\boldsymbol{V_{sp}}$, and finally Figure 4.33 presents the UAS yaw angle and the yaw setpoint ψ_{sp}.

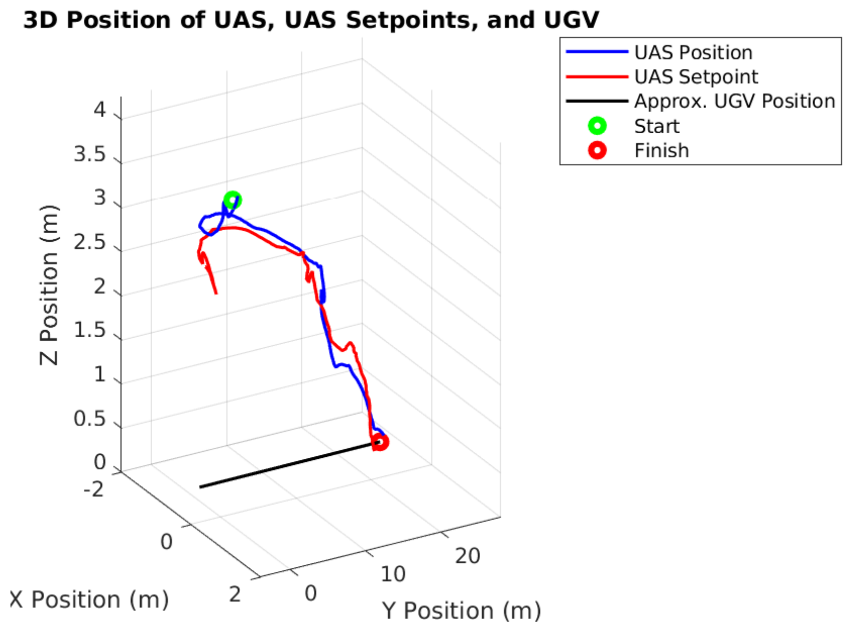


Figure 4.26: 3D Plot of the UAS position, commanded setpoint, and UGV path during moving landing with UGV traveling in a circular path.

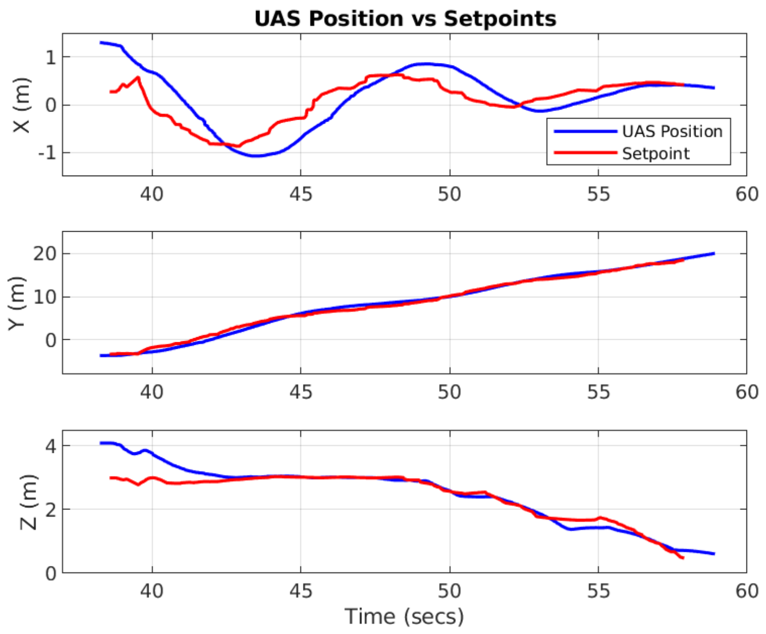


Figure 4.27: UAS position vs commanded position setpoint $\boldsymbol{P_{sp}}$ with the UGV traveling in a circular path.

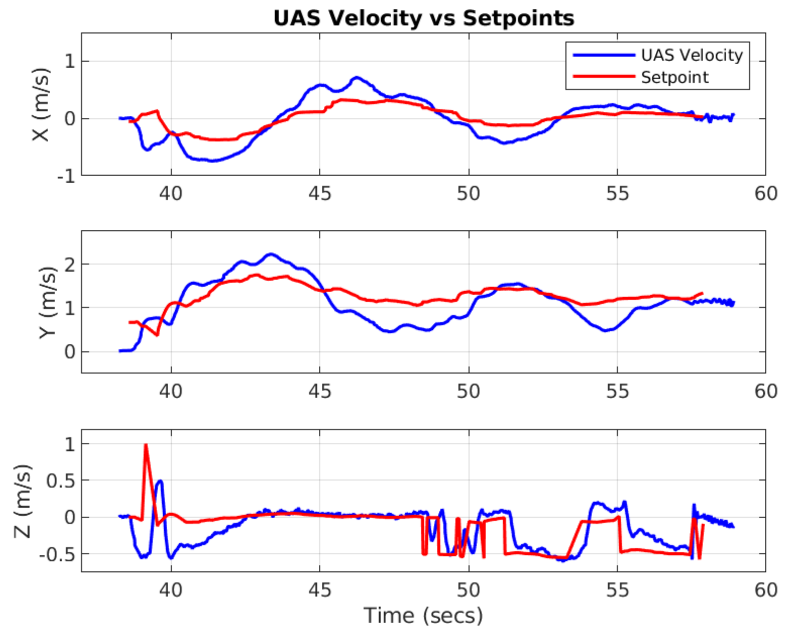


Figure 4.28: UAS velocity vs commanded velocity setpoint $\boldsymbol{V_{sp}}$ with the UGV traveling in a circular path.

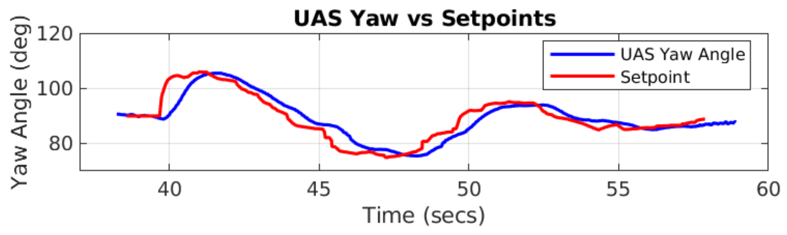


Figure 4.29: UAS yaw angle vs commanded yaw setpoint ψ_{sp} with the UGV traveling in a circular path.

4.3.2.2 UGV traveling in circular path

In this experiment the cart with landing pad was pulled forward via rope in a arced path at as close to 1.0 m/s as possible. The UAS was commanded to takeoff to an altitude of four meters at a position such that the UGV was in the UAS image frame. In this case, the UAS was approximately

0.75 meters to the right of the UGV. The state machine was then commanded to start the landing, and the cart began moving forward. Figure 4.30 displays a 3D plot overlaying the UAS Position, $\boldsymbol{P_{sp}}$, and a manual recreation of the UGV's approximate path. Figure 4.31 presents the x, y, and z components of both the UAS position, as well as the x, y, and z components of $\boldsymbol{P_{sp}}$. In similar fashion figure 4.32 presents the components of UAS velocity as well as the components of $\boldsymbol{V_{sp}}$, and finally Figure 4.33 presents the UAS yaw angle and the yaw setpoint ψ_{sp}.

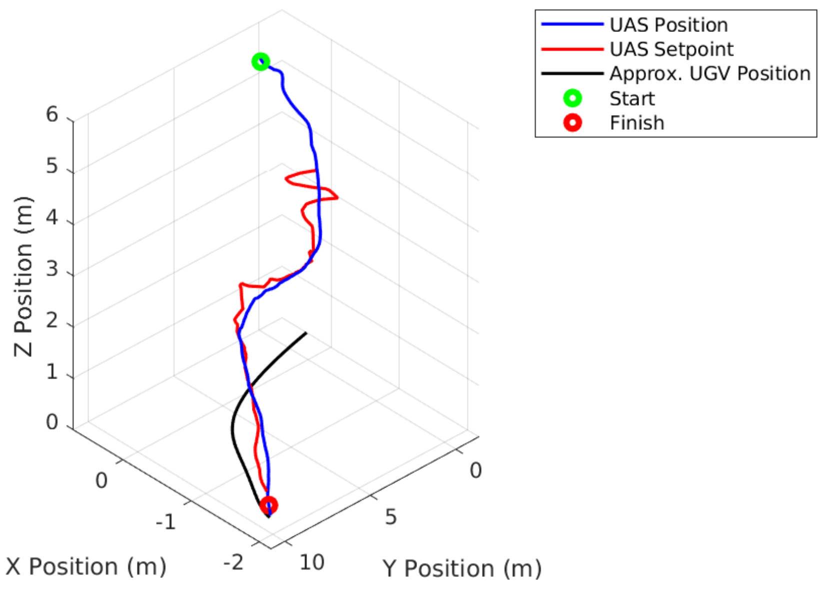


Figure 4.30: 3D Plot of the UAS position, commanded setpoint, and UGV path during moving landing with UGV traveling in a circular path.

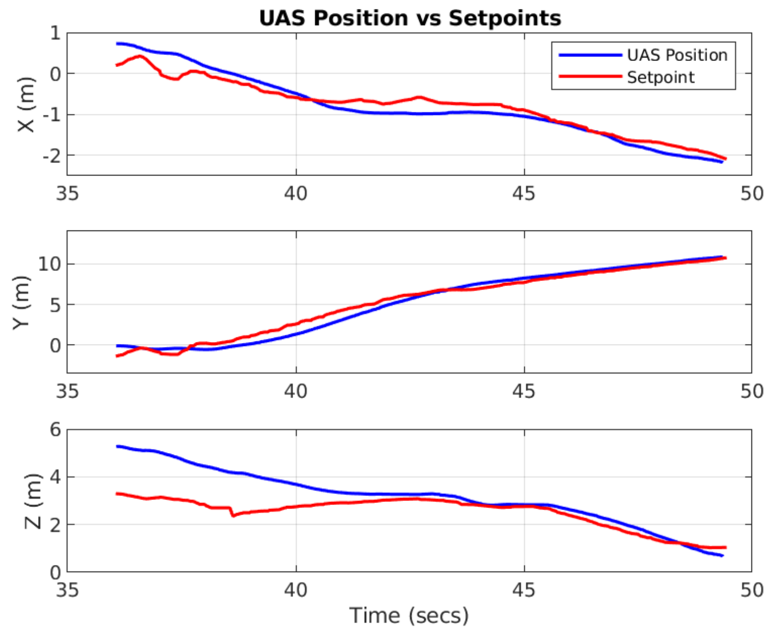


Figure 4.31: UAS position vs commanded position setpoint $\boldsymbol{P_{sp}}$ with the UGV traveling in a circular path.

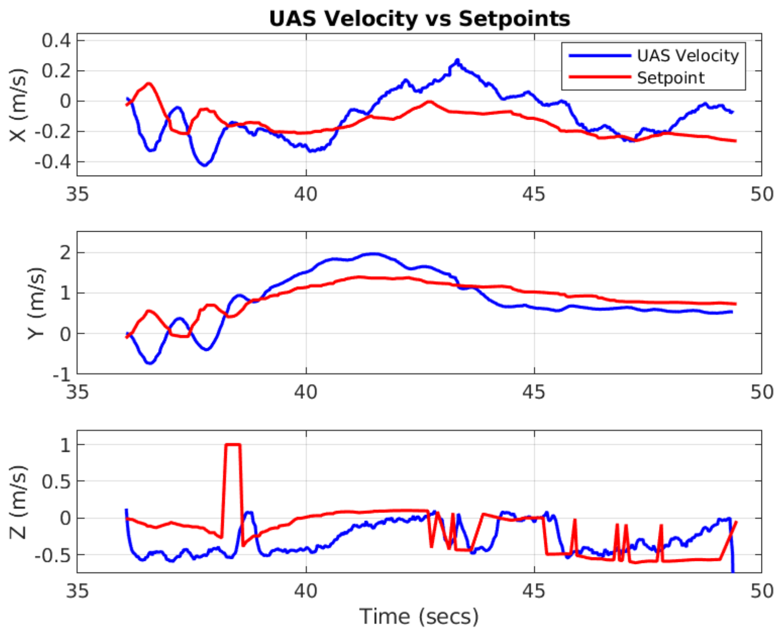


Figure 4.32: UAS velocity vs commanded velocity setpoint $\boldsymbol{V_{sp}}$ with the UGV traveling in a circular path.

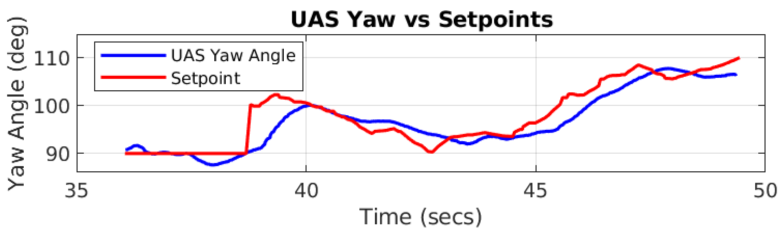


Figure 4.33: UAS yaw angle vs commanded yaw setpoint ψ_{sp} with the UGV traveling in a circular path.

4.3.2.3 Discussion

Changes in the z component of the UAS velocity can be observed to change from 0.0 to -0.5 m/s as the UAS enters and exits the acceptance sphere during the landing. Oscillations in the early part of the x and y components of $\boldsymbol{V_{sp}}$ were commonly observed during experimentation. Eventually the Kalman Filter converges to a near constant estimate of $\boldsymbol{V^n_{ugv}}$, but these oscillations in $\boldsymbol{V^n_{ugv}}$ can cause the vehicle to oscillate early in the landings, in turn causing errors with the yaw estimation. During experimentation the surrogate UGV was pulled such that it turned at as close to a constant rate as possible. Thus, the UGV's expected heading angle estimate should increase at a constant rate over time. The yaw results in Figure 4.33 depict the heading estimate decreasing from approximately 39 to 43 seconds. It is not until after 43 seconds that the UGV's heading begins estimate increases as expected. This erroneous heading estimation can be attributed to the Kalman Filter's velocity estimates not converging to a near constant velocity until approximately 43 seconds.

Compute limitations proved challenging during experimentation, as occasionally the object detection inferencing running on board would slow down to below 2 hz or fail. This verifies the work was pushing the on board compute capability to its limit. The landings were also observed to be sensitive to wind. Landings were demonstrated in winds blowing at 11 mph gusting up to 14 mph (measured with anemometer), but winds above approximately 8-10 mph significantly affected performance. On a day with winds below 6 mph a 71% success rate was observed across 7 landing attempts. Of these seven landings three landing completely on the pad, 2 resulted in the UAS on the pad but only partially, and the remaining 2 attempts resulted in the UAS missing the landing pad. A potential solution to shadows in future work could be to paint the pad a more distinctive color.

During landing experiments, the UGV's shadow caused issues as it was commonly falsely detected as the UGV. The shadow cast by the landing pad shifted the bounding box's center away from the center of the UGV landing pad causing errors in the landing target pose estimation. Depending on the sun's position this would either cause the UAS to either land near the edge of the pad or miss the landing pad entirely. To address this the training data set had to be appended with many images containing shadows. This improved shadow negation significantly, but shadows still continued to cause issues with UGV detection.

4.4 Simulation vs. Hardware

Separate object detection models were trained for use in simulation and experimentation as the UGV in simulation differs visually from the one used in experimentation. The models were trained in the same fashion and with similar amounts of training data, yet MobileNetV3 object detection proved much more accurate in simulation as bounding boxes almost always tightly conformed around the UGV. The high detection accuracy in simulation is attributed to the repeatable visual environment achievable in simulation. Factors such as UGV shadows and lighting variations caused the detections during experiments to be far noisier, producing errors in $\boldsymbol{T}^{c}_{ugv}$. The near perfect PX4 maintenance of $\boldsymbol{T}^{n}_{b}$ in simulation also aided in performance in comparison to experiments. During outdoor experimentation errors up to one meter were observed in $\boldsymbol{T}^{n}_{b}$. This introduction of noise in both $\boldsymbol{T}^{c}_{ugv}$ and $\boldsymbol{T}^{n}_{b}$ during outdoor experiments limited the top UGV speed at which experimental landings were successful to values below those achieved in simulation.

5. CONCLUSIONS

The following conclusions are made based on the methods and results presented in the previous chapters:

1. The presented work enabled vision based landings of a quadrotor UAS on a moving UGV at speeds of 3.0 meters per second in simulation and 1.0 meters per second in a straight line in simulation. Landings were also proven successful on a UGV traveling in an arced path at 1.0 meters per second in both simulation and on hardware in outdoor experiments. Outdoor experiments were successful in winds blowing 11 miles per hour and gusting to 14 miles per hour. A 71% success rate was achieved across a set of seven consecutive outdoor experiments.

2. Neural network object detection and the utilized detection localization algorithm were demonstrated to be a viable alternative to fiducial markers for autonomous landings. The presented localization algorithm achieved an iteration time of 49 milliseconds when deployed on the UAS on board computer, in comparison to the 44 milliseconds achieved with the fractal ArUcO algorithm. The presented detection localization method was also shown to be comparable to the fractal ArUCO marker in terms of accuracy.

3. All computations were able to be completed on-board the UAS where all sensing and computing components weigh less than 100 grams. This capability enables UAS to return to a UGV host without needing the UGVs or any other ground computer to aid with computations.

4. The combination of modern robotic tools and a high fidelity simulation aided this research tremendously. The ability to run the same algorithms in simulation and on hardware eased the transition from simulation to hardware where catastrophic failure can be costly. Changes were able to be tested in simulation, and rolled out onto the UAS in minutes.

6. RECOMMENDATIONS

Multiple recommendations are made regarding the research in this thesis:

1. Consider the replacement or removal of GPS to provide inertial position estimates of the UAS. This advancement to the work would enable operation in GPS denied environments. Visual Inertial Odometry offers a potential GPS replacement for UAS position data, or the command generation could be abstracted to just velocities to remove the need for UAS inertial position information. Removal of GPS reliance extends the number of environments in which the presented landing solution could be deployed.

2. Alleviate the necessity of assuming that the UGV is operating on flat and level ground would also extend the number of environments in which this landing work could be deployed. By estimating the UGVs position as the intersection of $p(t)$ and a point cloud generated from stereo vision, as opposed to the intersection $p(t)$ and the ground plane G, this landing could be deployed in environments where the UGV is not operating on flat and level ground.

3. Explore the potential for UGV/UAS communication to increase the accuracy of the vision based UGV localization. The UGVs position could also be utilized for the UAS to locate the UGV over long distances, then upon nearing the UGV the visual landing can be employed. This could be accomplished by implementing communication between the UAS and UGV such that the UGVs own state estimates could aid the landing trajectory generation. If communication is unavailable, the landing could fall back to a purely vision-based solution to avoid strict reliance on UGV/UAS communication.

4. Investigate the possibility of completing the landing target detection and localization simultaneously to reduce computational expense of the presented approach. Object detection models exist that are capable of generating three dimensional bounding boxes for detected objects [23, 24, 101]. These three-dimensional boxes offer a potential solution to combine

the detection and localization of the landing target into a single process. This single model approach could prove to be more computationally effecient that the presented detection localization method.

5. Investigate the use of gain scheduling for early and later portions of the landing trajectory. Deploying a gain schedule that allows for more aggressive UAS gains when farther from the UGV, and less aggressive gains when near the pad could improve performance and alleviate the need to tune a specific set of gains that is appropriate for the entire landing.

REFERENCES

[1] S. Garrido-Jurado, R. M.-S. F. Madrid-Cuevas, and M. Marin-Jiminez, "Automatic generation and detection of highly reliable fiducial markers under occlusion," in *Pattern Recognition*, vol. 47, pp. 2280–2292, 2014.

[2] S. Lee, T. Shim, S. Kim, J. Park, K. Hong, and H. Bang, "Vision-based autonomous landing of a multi-copter unmanned aerial vehicle using reinforcement learning," in *2018 International Conference on Unmanned Aircraft Systems (ICUAS)*, pp. 108–114, 2018.

[3] Y. Zhang, Y. Yu, S. Jia, and X. Wang, "Autonomous landing on ground target of uav by using image-based visual servo control," in *2017 36th Chinese Control Conference (CCC)*, pp. 11204–11209, 2017.

[4] F. Cocchioni, E. Frontoni, G. Ippoliti, S. Longhi, A. Mancini, and P. Zingaretti, "Visual based landing for an unmanned quadrotor," *Journal of Intelligent Robotic Systems*, vol. 84, pp. 511–528, 2015.

[5] Y. Xu, Z. Liu, and X. Wang, "Monocular vision based autonomous landing of quadrotor through deep reinforcement learning," in *2018 37th Chinese Control Conference (CCC)*, pp. 10014–10019, 2018.

[6] A. M. Almeshal and M. R. Alenezi, "A vision-based neural network controller for the autonomous landing of a quadrotor on moving targets," *Robotics*, vol. 7, no. 71, 2018.

[7] 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)*, pp. 115–123, 2018.

[8] R. Acuna, D. Zhang, and V. Willert, "Vision-based uav landing on a moving platform in gps denied environments using motion prediction," in *2018 Latin American Robotic Symposium,*

2018 Brazilian Symposium on Robotics (SBR) and 2018 Workshop on Robotics in Education (WRE), pp. 512–521, IEEE, January 2018.

[9] D. Wickramanayake, R. Rajasooriya, K. Ranawella, and D. Karunarathne, “Landing a quadcopter on to a moving landing target using computer vision,” 01 2019.

[10] J. Wubben, F. Fabra, C. T. Calafate, T. Krzeszowski, J. M. Marquez-Barja, J.-C. Cano, and P. Manzoni, “Accurate landing of unmanned aerial vehicles using ground pattern recognition,” *Electronics*, vol. 8, no. 12, 2019.

[11] S. Nogar, “Autonomous landing of a uav on a moving ground vehicle in a gps denied environment,” in *IEEE Symposium on Safety, Security, and Rescue Robotics (Online Conference)*, 2020.

[12] E. Olson, “Apriltag: A robust and flexible visual fiducial system,” in *IEEE International Conference on Robotics and Automation*, 2011.

[13] H. Voos and B. Nourghassemi, “Nonlinear control of stabilized flight and landing for quadrotor uavs,” 2009. http://www.issi.uz.zgora.pl/ACD_2009/program/Papers/44_ACD_2009.pdf.

[14] A. Paris, B. Lopez, and J. How, “Dynamic landing of an autonomous quadrotor on a moving platform in turbulent wind conditions,” in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, 2020.

[15] A. Howard, Z. Menglong, C. Bo, K. Dmitry, W. Weijun, W. Tobias, A. Marco, and A. Hartwig, “Mobilenets: Efficient convolutional neural networks for mobile vision applications,” April 2017.

[16] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, “Mobilenetv2: Inverted residuals and linear bottlenecks,” in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 4510–4520, 2018.

[17] A. Howard, R. Pang, H. Adam, Q. Le, M. Sandler, B. Chen, W. Wang, L.-C. Chen, M. Tan, G. Chu, V. Vasudevan, and Y. Zhu, “Searching for mobilenetv3,” pp. 1314–1324, 10 2019.

[18] J. Kim, K. Lin, S. M. Nogar, D. Larkin, and C. M. Korpela, “Detecting and localizing objects on an unmanned aerial system (uas) integrated with a mobile device,” in *2021 International Conference on Unmanned Aircraft Systems (ICUAS)*, pp. 738–743, 2021.

[19] M. Bleyer, C. Rhemann, and C. Rother, “Extracting 3d scene-consistent object proposals and depth from stereo images,” in *Computer Vision – ECCV 2012* (A. Fitzgibbon, S. Lazebnik, P. Perona, Y. Sato, and C. Schmid, eds.), pp. 467–481, 2012.

[20] H. Jiang, “Finding approximate convex shapes in rgbd images,” in *Computer Vision – ECCV 2014* (D. Fleet, T. Pajdla, B. Schiele, and T. Tuytelaars, eds.), pp. 582–596, 2014.

[21] D. Maturana and S. Scherer, “Voxnet: A 3d convolutional neural network for real-time object recognition,” in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 922–928, 2015.

[22] G. Zhang, H. Li, and F. Wenger, “Object detection and 3d estimation via an FMCW radar using a fully convolutional network,” in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2020.

[23] R. Juránek, A. Herout, M. Dubská, and P. Zemcík, “Real-time pose estimation piggybacked on object detection,” in *2015 IEEE International Conference on Computer Vision (ICCV)*, pp. 2381–2389, 2015.

[24] X. Chen, K. Kundu, Z. Zhang, H. Ma, S. Fidler, and R. Urtasun, “Monocular 3d object detection for autonomous driving,” in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2147–2156, 2016.

[25] J. Sochor, J. paňhel, and A. Herout, “Boxcars: Improving vehicle fine-grained recognition using 3d bounding boxes in traffic surveillance,” *ArXiv*, vol. abs/1703.00686, 2017.

[26] D. Lee, H. Lim, H. J. Kim, Y. Kim, and K. J. Seong, “Adaptive image-based visual servoing for an underactuated quadrotor system,” *Journal of Guidance, Control, and Dynamics*, vol. 35, no. 4, 2012.

[27] O. H. Boyers, "An evaluation of detection and recognition algorithms to implement autonomous target tracking with a quadrotor," Master's thesis, Rhodes University, Grahamstwon, South Africa, 11 2013.

[28] D. Falanga, A. Zanchettin, A. Simovic, J. Delmerico, and D. Scaramuzza, "Vision-based autonomous quadrotor landing on a moving platform," in *2017 IEEE International Symposium on Safety, Security and Rescue Robotics (SSRR)*, pp. 200–207, 2017.

[29] X. Liu, S. Zhang, J. Tian, and L. Liu, "An onboard vision-based system for autonomous landing of a low-cost quadrotor on a novel landing pad," *Sensors*, vol. 19, no. 21, 2019.

[30] U. Shriki, O. Gal, and Y. Doytsher, "Drone autonomous landing on a moving maritime platform using machine learning classifiers," *International Journal of Data Science and Advanced Analytics (ISSN 2563-4429)*, vol. 2, pp. 30–35, Dec. 2020.

[31] A. M. Almeshal and M. R. Alenezi, "A vision-based neural network controller for the autonomous landing of a quadrotor on moving targets," *Robotics*, vol. 7, no. 4, 2018.

[32] B. Lee, V. Saj, M. Benedict, and D. Kalathil, "A deep reinforcement learning control strategy for vision-based ship landing of vertical flight aircraft," in *AIAA AVIATION 2021 FORUM*.

[33] L. Wang and X. Bai, "Quadrotor autonomous approaching and landing on a vessel deck," *Journal of Intelligent Robotic Systems*, vol. 92, 2018.

[34] S. B. S. R. A. M. Yi Feng, Cong Zhang, "Autonomous landing of a uav on a moving platform using model predictive control," in *MDPI Drones*, vol. 34, 2018.

[35] L. Meier, D. Honegger, and M. Pollefeys, "Px4: A node-based multithreaded open source robotics framework for deeply embedded platforms," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 6235–6240, 2015.

[36] B. C.M., *A Pattern Recognition and Machine Learning*. Springer, 2006.

[37] M. Nielsen, *Nueral Networks and Deep Learning*. CA, USA. vol 25: Determination Press San Francisco, 2015.

[38] K. Hornik, M. Stinchcombe, and H. White, “Multilayer feedforward networks are universal approximators,” *Neural networks*, vol. 2, no. 5, pp. 359–366, 1989.

[39] K. Hornik, “Approximation capabilities of multilayer feedforward networks,” *Neural networks*, vol. 4, no. 2, pp. 251–257, 1991.

[40] W. S. McCulloch and W. H. Pitts, “Imagenet classification with deep convolutional neural networks,” *The Bulletin of Mathematical Biophysics*, vol. 5, no. 4, pp. 115–133, 1943.

[41] G. P. Zhang, “Nueral networks for classification: A survey,” *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 30, no. 4, 2000.

[42] G. Cybenko, “Approximation by superpositions of a sigmoidal function,” *Math. Control Signal Systems*, vol. 2, pp. 303–314, 1989.

[43] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[44] S. Sharma, S. Sharma, and A. Athaiya, “Activation functions in nueral networks,” *International Journal of Engineering Applied Sciences and Technology*, vol. 4, pp. 310–316, April 2020.

[45] A. Kaparthy, “Cs231n convolutional neural networks for visual recognition,” *Neural Netowrk*, vol. 1, 2016.

[46] P. Druzhkov and V. Kustikova, “A survey of deep learning methods and software tools for image classification and object detection,” *pattern Recognition and Image Analysis*, vol. 26, no. 1, pp. 9–15, 2016.

[47] C. Szegedy, A. Toshev, and D. Erhan, “Deep neural networks for object detection.” Google, Inc., 2013.

[48] R. S. Jia Deng, Wei Dong, L.-J. Li, and F.-F. Li, “Imagenet: A large-scale hierarchical image database,” in *IEEE Conference on Computer Vision and Pattern Recognition*, pp. 248–255, 2009.

[49] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in neural information processing systems*, pp. 1097–1105, 2012.

[50] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Region-based convolutional networks for accurate object detection and segmentation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 38, no. 1, pp. 142–158, 2016.

[51] R. Girshick, "Fast r-cnn," *Proceedings of the IEEE International Conference on Computer Vision*, pp. 1440–1448, 2015.

[52] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, no. 6, pp. 1137–1149, 2017.

[53] Z. Zou, Z. Shi, Y. Guo, and J. Ye, "Object detection in 20 years: A survey," *ArXiv*, vol. abs/1905.05055, 2019.

[54] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[55] J. Redmon and A. Farhadi, "Yolo9000: Better, faster, stronger," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[56] J. Redmon and A. Farhadi, "Yolov3: An incremental improvement," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[57] W. Liu, D. Anguelov, D. Erhan, C. S. S. Reed, and C.-Y. Fu, "Ssd: Single shot multibox detector," in *European Conference on Computer Vision*, pp. 21–37, Springer International Publishing, 2016.

[58] L. Liu, W. Ouyang, X. Wang, P. Fieguth, J. Chen, X. Liu, and M. Pietkainen, "Deep learning for generic object detection: A survey," *International Journal of Computer Vision*, vol. 128, pp. 261–318, 2020.

[59] M. J. Shafiee, B. Chywl, F. Li, and A. Wong, "Fast YOLO: A fast you only look once system for real-time embedded object detection in video," *arXiv Computing Research Repository*, 2017.

[60] R. Huang, J. Pedoeem, and C. Chen, "Yolo-lite: A real-time object detection algorithm optimized for non-gpu computers," in *2018 IEEE International Conference on Big Data (Big Data)*, pp. 2503–2510, 2018.

[61] J. Valasek, H. C. Lehman, and V. G. Goecks, "Online intelligent motion video guidance for unmanned air system ground target surveillance," in *AIAA SciTech*, 2019.

[62] P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features," in *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*, vol. 1, 2001.

[63] U. Caluori and S. Klaus, "Glyph recognition by pattern matching on-the-fly generated patterns," in *IEEE 21ist International Conference on Software, Telecommunications, and Computer Networks-(SoftCOM 2013)*, 2013.

[64] A. Martinez, "Fisherfaces," *Scholarpedia*, vol. 6, no. 2, p. 4282, 2011.

[65] L. Arreola, G. Gudiño, and G. Flores, "Object recognition and tracking using haar-like features cascade classifiers: Application to a quad-rotor uav," in *International Conference on Unmanned Systems*, 2019.

[66] C. Kanellakis and G. Nikolakopoulos, "Survey on computer vision for uavs: Current developments and trends," *Journal of Intelligent and Robotic Systems*, vol. 87, p. 141–168, 2017.

[67] J. G. Ferrer, "A follow-me algorithm for ar.drone using mobilenet-ssd and pid control," Master's thesis, Universitat de Barcelona, Computer Science, Barcelona, Spain, 6 2018.

[68] A. Rohan, M. Rabah, and S.-H. Kim, "Convolutional neural network-based real-time object detection and tracking for parrot ar drone 2," *IEEE Access*, vol. 7, pp. 69575–69584, 2019.

[69] J. Lee, J. Wang, D. Crandall, S. Sabanovic, and G. Fox, "Real-time, cloud-based object detection for unmanned aerial vehicles," in *IEEE International Conference on Robotic Computing*, 2017.

[70] H. H. A. Kadouf and Y. M. Musafah, "Colour-based object detection and tracking for autonomous quadrotor uav," *International Conference on Mechatronics*, vol. 53, 2013.

[71] J. Rodriguez, C. Castiblanco, I. Mondragon, and J. Colorado, "Low-cost quadrotor applied for visual detection of landmine-like objects," in *2014 International Conference on Unmanned Aircraft Systems (ICUAS)*, pp. 83–88, 2014.

[72] L. Meng, Z. Peng, J. Zhou, J. Zhang, Z. Lu, A. Bausmann, and Y. Du, "Real-time object detection of ground objects based on unmanned vehicle remote sensing with deep learning: Application in excavator detection for pipeline saftey," *Remote Sensing*, vol. 12, no. 182, 2020.

[73] J. R. B. del Rosario, E. J. C. Azarraga, M. J. C. Chiu, A. J. C. Del Rosario, A. B. S. Jarabelo, and A. A. Bandala, "Development of a vision based parking monitoring system using quadrotor uav," in *2020 IEEE 12th International Conference on Humanoid, Nanotechnology, Information Technology, Communication and Control, Environment, and Management (HNICEM)*, pp. 1–6, 2020.

[74] A. Pérez, P. Chamoso, V. Parra, and A. J. Sánchez, "Ground vehicle detection through aerial images taken by a uav," in *17th International Conference on Information Fusion*, pp. 1–6, 2014.

[75] H. Xie, J. Li, and K. Low, "Dynamic output feedback image-based visual servoing of rotorcraft uavs," in *2017 International Conference on Unmanned Aircraft Systems (ICUAS)*, pp. 1361–1367, 2017.

[76] M. Saqib, S. Daud Khan, N. Sharma, and M. Blumenstein, "A study on detecting drones using deep convolutional neural networks," in *2017 14th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS)*, pp. 1–5, 2017.

[77] V. G. Goecks, G. Woods, and J. Valasek, “Combining visible and infrared spectrum imagery using machine learning for small unmanned aerial system detection,” in *Automatic Target Recognition XXX*, vol. 11394, pp. 198 – 207, International Society for Optics and Photonics, SPIE, 2020.

[78] P. M. Wyder, Y.-S. Chen, A. J. Lasrado, R. J. Pelles, R. Kwiatkowski, E. O. A. Comas, R. Kennedy, A. Mangla, Z. Huang, X. Hu, Z. Xiong, T. Aharoni, T.-C. Chuang, and H. Lipson, “Autonomous drone hunter operating by deep learning and all-onboard computations in gps-denied environments,” *PLoS ONE*, vol. 14, 2019.

[79] G. Schall, J. Newman, and D. Schmalstieg, “Rapid and accurate deployment of fiducial markers for augmented reality,” in *10th Computer Vision Workshop 2005*, 2005.

[80] M. Kalaitzakis, B. Cain, S. Carroll, A. Ambrosi, C. White, and N. Vitzilaios, “Fiducial markers for pose estimation,” *Journal of Intelligent Robotic Systems*, vol. 101, no. 71, 2021.

[81] J. Springer, “Autonomous landing of a multicopter using computer vision,” Master’s thesis, Reykjavik University, Department of Computer Science, Reykjavik, Iceland, 6 2020.

[82] H. Kato and M. Billinghurst, “Marker tracking and hmd calibration for a video-based augmented reality conferencing system,” in *IEEE and ACM International Workshop on Augmented Reality (IWAR ’07)*, 1999.

[83] M. Rohs, “Real-World Interaction with Camera Phones,” in *Ubiquitous Computing Systems*, 2004.

[84] D. Flohr and J. Fischer, “A Lightweight ID-Based Extension for Marker Tracking Systems,” in *Eurographics Symposium on Virtual Environments, Short Papers and Posters*, 2007.

[85] M. Fiala, “Artag, a fiducial marker system using digital techniques,” in *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR’05)*, vol. 2, pp. 590–596 vol. 2, 2005.

[86] M. Fiala, “Designing highly reliable fiducial markers,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 32, no. 7, pp. 1317–1324, 2010.

[87] J. Wang and E. Olson, “Apriltag 2: Efficient and robust fiducial detection,” in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 4193–4198, 2016.

[88] S. Garrido-Jurado, R. Muñoz-Salinas, F. Madrid-Cuevas, and R. Medina-Carnicer, “Generation of fiducial marker dictionaries using mixed integer linear programming,” *Pattern Recognition*, vol. 51, pp. 481–491, 2016.

[89] F. J. Romero-Ramirez, R. Muñoz-Salinas, and R. Medina-Carnicer, “Speeded up detection of squared fiducial markers,” *Image and Vision Computing*, vol. 76, pp. 38–47, 2018.

[90] F. Romero-Ramirez, R. Muñoz-Salinas, and R. Medina-Carnicer, “Fractal markers: a new approach for long-range marker pose estimation under occlusion,” in *IEEE Digital Object Detection*, 04 2019.

[91] M. Popova, “Visual servoing for a quadrotor uav in target tracking applications,” in *AIAA Guidance, Navigation, and Control Conference*, AIAA, January 2015.

[92] T. Yang, Q. Ren, F. Zhang, B. Xie, H. Ren, J. Li, and Y. Zhang, “Hybrid camera array-based uav auto-landing on moving ugv in gps-denied environment,” *Remote Sensing*, vol. 10, no. 11, 2018.

[93] A. Carrio, J. Tordesillas, S. Vemprala, S. Saripalli, P. Campoy, and J. P. How, “Onboard detection and localization of drones using depth maps,” *IEEE Access*, vol. 8, pp. 30480–30490, 2020.

[94] Y. Hong, J. Liu, Z. Jahangir, S. He, and Q. Zhang, “Estimation of 6d object pose using a 2d bounding box,” *Sensors*, vol. 21, no. 9, 2021.

[95] J. Gall, A. Yao, N. Razavi, L. Van Gool, and V. Lempitsky, “Hough forests for object detection, tracking, and action recognition,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 33, no. 11, pp. 2188–2202, 2011.

[96] C. Redondo-Cabrera, R. López-Sastre, and T. Tuytelaars, "All together now: Simultaneous detection and continuous pose estimation using a hough forest with probabilistic locally enhanced voting," in *BMVC Computer Science*, 2014.

[97] M. Ozuysal, V. Lepetit, and P. Fua, "Pose estimation for category specific multiview object localization," in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 778–785, 2009.

[98] R. J. López-Sastre, T. Tuytelaars, and S. Savarese, "Deformable part models revisited: A performance evaluation for object category pose estimation," in *2011 IEEE International Conference on Computer Vision Workshops (ICCV Workshops)*, pp. 1052–1059, 2011.

[99] K. He, L. Sigal, and S. Sclaroff, "Parameterizing object detectors in the continuous pose space," in *Computer Vision – ECCV 2014* (D. Fleet, T. Pajdla, B. Schiele, and T. Tuytelaars, eds.), pp. 450–465, 2014.

[100] T.-T. Do, M. Cai, T. T. Pham, and I. Reid, "Deep-6dpose: Recovering 6d object pose from a single rgb image," *ArXiv*, vol. abs/1802.10367, 2018.

[101] S. Song and M. Chandraker, "Joint sfm and detection cues for monocular 3d localization in road scenes," in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 3734–3742, 2015.

[102] J. Ku, M. Mozifian, J. Lee, A. Harakeh, and S. L. Waslander, "Joint 3d proposal generation and object detection from view aggregation," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 1–8, 2018.

[103] S. Hutchinson, G. Hager, and P. Corke, "A tutorial on visual servo control," *IEEE Transactions on Robotics and Automation*, vol. 12, no. 5, pp. 651–670, 1996.

[104] D. Kragic, H. Christensen, and F. A, "Survey on visual servoing for manipulation," *Comput. Vis. Act. Percept. Lab. Fiskartorpsv*, vol. 15, 02 2002.

[105] M. Zarudzki, H.-S. Shin, and C.-H. Lee, "An image based visual servoing approach for multi-target tracking using an quad-tilt rotor uav," in *International Conference on Unmanned Aircraft Systems (ICUAS)*, pp. 781–790, IEEE, June 2017.

[106] D. Zheng, H. Wang, and W. Chen, "Image-based visual tracking of a moving target for a quadrotor," in *11th Asian Control Conference (ASCC)*, pp. 198–203, ASCC, January 2017.

[107] Z. Cao, X. Chen, Y. Yu, J. Yu, X. Liu, C. Zhou, and M. Tan, "Image dynamics-based visual servoing for quadrotors tracking a target with a nonlinear trajectory observer," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 50, pp. 376–384, January 2020.

[108] J. Wang, C. Sadler, C. F. Montoya, and J. C. Liu, "Optimizing ground vehicle tracking using unmanned aerial vehicle and embedded apriltag design," in *2016 International Conference on Computational Science and Computational Intelligence (CSCI)*, pp. 739–744, 2016.

[109] J. G. Ferrer, "A follow-me algorithm for ar drone using mobilenet-ssd and pid control," Master's thesis, Universitat de Barcelona Department of Applied Mathematics and Analy, Barcelona, Spain, 6 2020.

[110] S. Li and D.-Y. Yeung, "Visual object tracking for unmanned aerial vehicles: A benchmark and new motion models," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 31, Feb. 2017.

[111] M. G. Popova, "Visual servoing for a quadrotor uav in target tracking applications," Master's thesis, University of Toronto Department of Aerospace Engineering, Toronto, Canada, 11 2015.

[112] J. Thomas, J. Welde, G. Loianno, K. Daniilidis, and V. Kumar, "Autonomous flight for detection, localization, and tracking of moving targets with a small quadrotor," *IEEE Robotics and Automation Letters*, vol. 2, no. 3, pp. 1762–1769, 2017.

[113] C. J. C. H. Watkins and P. Dayan, "Q-learning," *Machine Learning*, vol. 8, pp. 279–292, 1992.

[114] C. Fu, R. Duan, D. Kircali, and E. Kayacan, “Onboard robust visual tracking for uavs using a reliable global-local object model,” *Sensors*, vol. 16, no. 9, 2016.

[115] M.-K. Hu, “Visual pattern recognition by moment invariants,” *IRE Transactions on Information Theory*, vol. 8, no. 2, pp. 179–187, 1962.

[116] S. Saripalli and G. Sukhatme, “Landing on a moving target using an autonomous helicopter,” *International Conference on Field and Service Robotics*, vol. 24, 04 2004.

[117] J. Gomez-Avila, C. Lopez-Franco, A. Y. Alanis, N. Arana-Daniel, and M. Lopez-Franco, “Ground vehicle tracking with a quadrotor using image based visual servoing,” *2nd IFAC Conference on Modelling, Identification and Control of Nonlinear Systems MICNON 2018*, vol. 51, no. 13, pp. 344–349, 2018.

[118] R. Polvara, S. Sharma, J. Wan, A. Manning, and R. Sutton, “Vision-based autonomous landing of a quadrotor on the perturbed deck of an unmanned surface vehicle,” *Drones*, vol. 2, no. 2, 2018.

[119] D. Han, J. Kim, and J. Kim, “Deep pyramidal residual networks,” in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 6307–6315, 2017.

[120] T.-J. Yang, A. Howard, B. Chen, X. Zhang, A. Go, M. Sandler, V. Sze, and H. Adam, “Netadapt: Platform-aware neural network adaptation for mobile applications,” *ArXiv*, vol. abs/1804.03230, 2018.

[121] J. Hu, L. Shen, and G. Sun, “Squeeze-and-excitation networks,” in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 7132–7141, 2018.

[122] P. Ramachandran, B. Zoph, and Q. V. Le, “Searching for activation functions,” *ArXiv*, vol. abs/1710.05941, 2017.

[123] S. Elfwing, E. Uchibe, and K. Doya, “Sigmoid-weighted linear units for neural network function approximation in reinforcement learning,” *ArXiv*, vol. abs/1702.03118, 2017.

[124] D. Hendrycks and K. Gimpel, “Gaussian error linear units (gelus),” *ArXiv*, vol. abs/1606.08145, 2016.

[125] “Clearpath robotics warthog,” https://clearpathrobotics.com/warthog-unmanned-ground-vehicle-robot/, Accessed 2021-08-26.

[126] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, “TensorFlow: Large-scale machine learning on heterogeneous systems,” 2015. Software available from tensorflow.org.

[127] “ModalAI VOXL m500,” https://www.modalai.com/products/voxl-m500?variant=31790290599987, Accessed 2021-07-26.

[128] “Roboflow,” https://roboflow.com.

[129] Autonomous Systems Lab ETH Zurich, “image_undistort,” https://github.com/ethz-asl/image_undistort.

[130] R. Szeliski, “Computer vision: Algorithms and applications.” http://szeliski.org/Book/drafts/SzeliskiBook_20100903.pdf.

[131] R. I. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*. Cambridge University Press, ISBN: 0521540518, second ed., 2004.

[132] “Snapdragon 820 Mobile Platform,” https://www.qualcomm.com/products/snapdragon-820-mobile-platform, Accessed 2021-07-26.

[133] D. Brescianini, M. Hehn, and R. D’Andrea, “Nonlinear quadrocopter attitude control technical report,” in *Institute for Dynamic Systems and Control (IDSC) ETH Zurich*.

[134] R. E. Kalman and R. S. Bucy, "New Results in Linear Filtering and Prediction Theory," *Journal of Basic Engineering*, vol. 83, pp. 95–108, 03 1961.

[135] FilterPy, "Filterpy kinematic kalman filter," https://github.com/rlabbe/filterpy.

[136] Smach, "State machine ros package," http://wiki.ros.org/smach.

[137] J. K. Haas, "A history of the unity game engine," 2014.

[138] "Pixhawk flight controllers," https://pixhawk.org/products/.

[139] "Modalai flight core flight control unit," https://www.modalai.com/products/flight-core.

[140] "Arm cortex-m7 32-bit," https://www.st.com/en/microcontrollers-microprocessors/stm32-32-bit-arm-cortex-mcus.html.

[141] "Voxl computer," https://www.modalai.com/products/voxl-flight-deck-r1?variant=31790279196723.

[142] "Voxl flight," https://www.modalai.com/products/voxl-flight?variant=31636287094835.

[143] Stanford Artificial Intelligence Laboratory et al., "Robot Operating System," https://www.ros.org.

[144] G. Bradski, "The OpenCV Library," *Dr. Dobb's Journal of Software Tools*, 2000.

[145] D. Merkel, "Docker: lightweight linux containers for consistent development and deployment," *Linux journal*, vol. 2014, no. 239, p. 2, 2014.

[146] MAVROS, "Mavros ros package," http://wiki.ros.org/mavros.

[147] "Invensense icm-20602 imu," https://invensense.tdk.com/products/motion-tracking/6-axis/icm-20602/.

[148] "Invensense icm-42688 imu," https://invensense.tdk.com/products/motion-tracking/6-axis/icm-42688-p/.

[149] "Bosch bmi0888 imu," https://www.bosch-sensortec.com/products/motion-sensors/imus/bmi088/.

[150] "Bosch bmp388 barometer," https://www.bosch-sensortec.com/products/environmental-sensors/pressure-sensors/bmp388/.

[151] "Pixhawk 4 gps module," http://www.holybro.com/product/pixhawk-4-gps-module/.

[152] "Ovm7251 bw cmos vga sensor," https://www.ovt.com/products/ovm7251-raia-r1/.

[153] "Sony imx214 cmos color sensor," https://www.mouser.com/datasheet/2/897/ProductBrief_214_-1289331.pdf.

APPENDIX A

UNMANNED AIR SYSTEM DESCRIPTION

Appendix A provides details on the UAS used for experimental demonstration of the vision-based autonomous landing presented. It contains an overview of the flight control unit (FCU), the on-board companion computer, and details regarding sensors on-board the UAS. These details are aimed to provide context on the capabilities of the UAS, and to be used as a frame of reference for the performance achieved in the presented autonomous landing research. The UAS used for this work was the VOXL® m500 developed by ModalAI® [127]. This vehicle was selected for its suite of visual sensors, and its companion computer. The on-board companion computer contains a GPU which aids in the acceleration of artificial intelligence computations, such as the use of neural network object detectors. Figure 6.1 sourced from [127] displays the VOXL® m500 UAS.

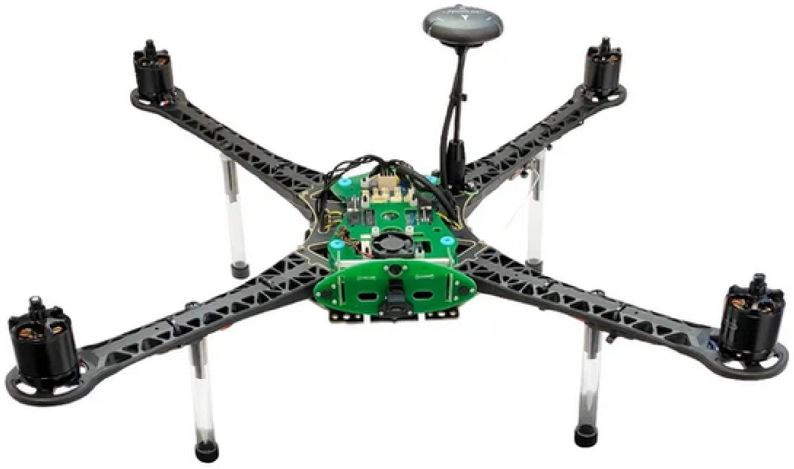


Figure A.1: VOXL® m500 UAS from ModalAI®; reprinted from [127].

A.1 Physical Description

The VOXL® m500 is a four rotor multi-rotor UAS, also known as a quadrotor. Table 6.1 below summarizes the physical descriptions measurements provided by the manufacturer ModalAI® [127].

Table A.1: Size and Weight Specifications of VOXL® m500 [127].

Specification	**VOXL m500**
Weight	1075 grams
Payload Capacity	1 kg (effects endurance)
Endurance	Approximatley 20 minutes
Length	15.5 inches
Width	15.5 inches
Height	8 inches
Propeller Diameter	10 inches

Minor modification to the vehicle was neccessary for this work. From the manufacturer the suite of visual sensors on board the VOXL® m500 UAS faces forward during flight. As this work aims to land on a UGV, the cameras need to be downward facing. To accomplish this, the entire suite of visual sensors was removed, rotated 90 degrees into a downward facing orientation, and reattached using a custom designed 3D printed part.

A.2 Flight Control Unit

The VOXL® m500 runs the open source PX4 flight controller [35]. PX4 is an open source flight control software for UAS and other unmanned vehicles. PX4 exists as an ecosystem containing the software and hardware needed to manually or autonomously fly UAS.

A.2.1 Hardware

While PX4 can run on Linux computers as well as off the shelf Pixhawk® style flight controllers [138], the VOXL® m500 employs the ModalAI ® Flight Core® [139]. The ModalAI® Flight Core® is a PX4 flight controller based on the STM32F7 ARM micro-controller [140]. Flight Core® contains three on-board Inetial Measurement Units (IMUs) and a barometer.

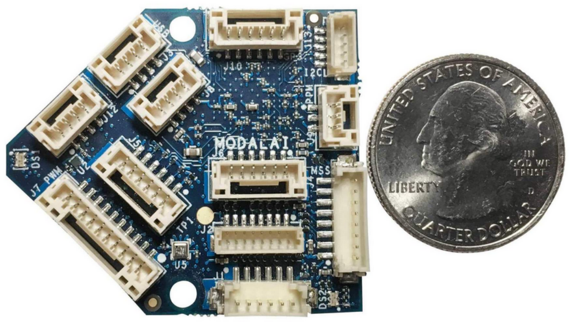


Figure A.2: ModalAI® Flight Core® FCU; reprinted from [127].

A.2.2 Software

PX4 software handles much of the overhead of UAS operation, allowing researchers to focus on developing autonomous behaviors. PX4 handles critical functions such as managing data from all on-board sensors, UAS state estimation, position, velocity and attitude control, as well as safety and fail safe management. All of these features combined enable manual and autonomous flight of UAS out of the box.

While capable of standalone flight, PX4 also allows for a companion computer to send commands either over a serial connection on-board the vehicle, or over wireless communication from a ground station computer. The use of external computers enable to deployment of more computational expensive autonomous systems such as vision-based control.

Another key feature of PX4 software is the ability to build PX4 within a desktop environment.

This enables PX4 to be used in simulation environments, aiding in autonomy development as new methods can be validated in simulation before they are tested on hardware.

A.3 Companion Computer

The VOXL® m500 also uses an on-board companion computer from ModalAI®, the ModalAI® VOXL® [141]. The VOXL® m500 UAS actually utilizes a single printed circuit board (PCB) containing both the Flight Core® FCU and the VOXL® companion computer, as shown in Figure 6.3 [142].

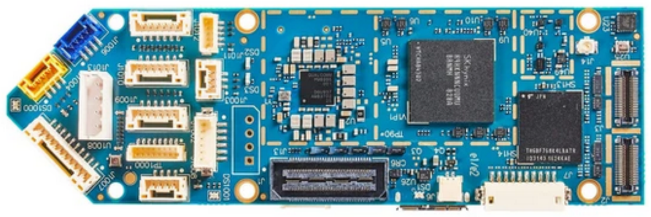


Figure A.3: ModalAI® VOXL® Flight, a combination of the Flight Core® FCU and the VOXL® companion computer; reprinted from [127].

A.3.1 Hardware

The VOXL® computer is built around the Qualcom® Snapdragon™ 821 chip-set [132] which contains a quad-core Qualcom® Kyro™ CPU with speeds up to 2.15 GHz, a Qualcom® Adreno™ 530 GPU with speeds of up to 624 MHz, 4 gigabytes of LPDDR4 random access memory (RAM), and 32 gigabytes of internal storage. The addition of a GPU aids in the acceleration of artificial intelligence and vision processes such as object detection and visual inertial odometry (VIO). Wi-Fi is supported by VOXL® enabling wireless communication by either acting as a stand alone network, or by connecting to an existing network. VOXL® is also able to support both radio and cellular communication for long range applications. The companion computer is also responsible for managing all visual sensors on-board the UAS via its three Mobile Industry Processor Interface

(MIPI) connectors, and hosts two IMU sensors in addition to the three IMUs on the FCU.

A.3.2 Software

Specifics on various software such as the autonomous landing algorithm, detection localization algorithm, and the object detection are covered in their respective chapters. This section serves as a brief summary of the major software tools running on-board the VOXL® computer to provide insight as to what was accomplished with the defined hardware. VOXL® uses a Yocto Linux operating system, enabling the use of common software tools used in UAS autonomy such as PX4 [35], ROS [143], OpenCV [144], and Docker [145]. This thesis uses ROS as the main framework for the autonomous landing, as its structure and features ease the development of robotic systems. VOXLs® integration with ROS eases the passing of image data between ROS nodes for visual servoing. MAVROS [146] is a ROS package that takes the relevant data from PX4 and publishes it to ROS topics, even further aiding in autonomy development as now image and FCU data is easily accessible by any ROS node. For example, the detection localization process takes place inside a ROS node that relies on both image data and altitude data frrom MAVROS. MAVROS also facilitates the communication of commands from the off-board computer to the FCU for off board control of the UAS. A ROS based Smach state machine also runs on-board the VOXL® to facilitate high level behavioral control. Finally, a TensorFlowLite MobileNetV3 object detection model is deployed on the VOXL® for detection of the UGV.

A.4 Sensors

A.4.1 Data Sensors

The Flight Core® FCU contains three on-board inertial measurment units (IMUs) for aid in state estimation, as well as a barometer used for altitude estimation. Flight Core® supports additional sensors that are commonly used for UAS autonomy. This work utilizes a GPS receiver for position estimation as well as a laser rangefinder to aid in altitude estimation. The VOXL® companion computer also ships with two additional IMUs. Table A.2 summarizes all on-board sensors.

Table A.2: Sensors On Board the VOXL® m500 [127].

Sensor	Name
FCU-IMU 1	ICM-20602 [147]
FCU-IMU 2	ICM-42688 [148]
FCU-IMU 3	BMI088 [149]
FCU-Barometer	BMP388 [150]
GPS	Pixhawk 4 GPS Module [151]

A.4.2 Visual Sensors

The VOXL® ships with four on-board visual sensors. The only visual sensor used in this work is a full color high resolution Sony IMX214 camera. The remaining sensors are all Omivision OVM7251 black and white sensors [152]. One is a higher field of view version denoted as the tracking camera, and the other two are used to generate stereo disparity images for stereo vision applications. Table A.3 summarizes details regarding all on-board visual sensors.

Table A.3: Visual Sensors On Board the VOXL® m500 [127].

Sensor	Name	Shutter	Resolution	Framerate
Hires	Sony IMX214 [153]	Rolling	4224 x 3200	Up to 60
Tracking	Omnivision OVM7251 [152]	Global	640 x 480	30, 60, 90, 120
Stereo	Omnivision OVM7251 [152]	Global	640 x 480	Up to 60

APPENDIX B

KALMAN FILTER SPECIFICS

The Kalman Filter described in Chapter 4 is based on a three-dimensional constant velocity kinematic model.

B.1 State Transition Matrix

Expressing the states evolution over time with constant velocity can be expressed with the kinematic equations described in Equation B.1.

$$\begin{aligned} x_{t+1} &= x_t + \dot{x}_t \Delta t \\ \dot{x}_{t+1} &= \dot{x}_t \\ y_{t+1} &= y_t + \dot{y}_t \Delta t \\ \dot{y}_{t+1} &= \dot{y}_t \\ z_{t+1} &= z_t + \dot{z}_t \Delta t \\ \dot{z}_{t+1} &= \dot{z}_t \end{aligned} \tag{B.1}$$

The formulation of these equations into matrix form yields the state transition matrix F such that $\boldsymbol{x_{t+1}} = F\boldsymbol{x_t}$ as depicted in Equation B.2.

$$\begin{bmatrix} x_{t+1} \\ \dot{x}_{t+1} \\ y_{t+1} \\ \dot{y}_{t+1} \\ z_{t+1} \\ \dot{z}_{t+1} \end{bmatrix} = \begin{bmatrix} 1 & \Delta t & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & \Delta t & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & \Delta t \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_t \\ \dot{x}_t \\ y_t \\ \dot{y}_t \\ z_t \\ \dot{z}_t \end{bmatrix} \tag{B.2}$$

B.2 Process Noise Covariance Matrix

The process noise utilized in this filter is discrete, that is it is different at each time step but constant between time steps. For the constant velocity kinematic Kalman Filter used, the process noise covariance matrix is described in Equation B.3 where $COV()$ denotes covariance.

$$Q = \begin{bmatrix} COV(x,x) & COV(x,\dot{x}) & COV(x,y) & COV(x,\dot{y}) & COV(x,z) & COV(x,\dot{z}) \\ COV(\dot{x},x) & COV(\dot{x},\dot{x}) & COV(\dot{x},y) & COV(\dot{x},\dot{y}) & COV(\dot{x},z) & COV(\dot{x},\dot{z}) \\ COV(y,x) & COV(y,\dot{x}) & COV(y,y) & COV(y,\dot{y}) & COV(y,z) & COV(y,\dot{z}) \\ COV(\dot{y},x) & COV(\dot{y},\dot{x}) & COV(\dot{y},y) & COV(\dot{y},\dot{y}) & COV(\dot{y},z) & COV(\dot{y},\dot{z}) \\ COV(z,x) & COV(z,\dot{x}) & COV(z,y) & COV(z,\dot{y}) & COV(z,z) & COV(z,\dot{z}) \\ COV(\dot{z},x) & COV(\dot{z},\dot{x}) & COV(\dot{z},y) & COV(\dot{z},\dot{y}) & COV(\dot{z},z) & COV(\dot{z},\dot{z}) \end{bmatrix} \tag{B.3}$$

By assuming that the states are only dependent on states of the same axis (either X, Y,or Z) many of the terms go to zero. The diagonal terms can also be expressed as variances, denoted as $V()$. The resulting matrix Q is defined in equation B.4.

$$Q = \begin{bmatrix} V(x) & COV(x,\dot{x}) & 0 & 0 & 0 & 0 \\ COV(\dot{x},x) & V(\dot{x}) & 0 & 0 & 0 & 0 \\ 0 & 0 & V(y) & COV(y,\dot{y}) & 0 & 0 \\ 0 & 0 & COV(\dot{y},y) & V(\dot{y}) & 0 & 0 \\ 0 & 0 & 0 & 0 & V(z) & COV(z,\dot{z}) \\ 0 & 0 & 0 & 0 & COV(\dot{z},z) & V(\dot{z}) \end{bmatrix} \tag{B.4}$$

Equation B.5, B.6, and B.7 describes the evaluation of a few terms in Equation B.4, where $E()$ denotes expected value. The remaining terms are calculated in the same fashion as the terms in Equation B.5. The position and velocity variance and convariance are expressed in terms of random acceleration variance of the model, σ_a. The expected values used are determined by utilizing the acceleration update component of constant acceleration kinematic equations.

$$V(x) = \sigma_x^2 = E(x^2) - \mu_x^2 = E((\frac{1}{2}a\Delta t^2)^2) - (\frac{1}{2}\mu_a\Delta t^2)^2 = \frac{\Delta t^4}{4}(E(a^2) - \mu_a^2) = \frac{\Delta t^4}{4}\sigma_a^2 \quad \text{(B.5)}$$

$$V(\dot{x}) = \sigma_{\dot{x}}^2 = E(\dot{x}^2) - \mu_{\dot{x}}^2 = E((a\Delta t)^2) - (\mu_a\Delta t)^2 = \Delta t^2(E(a^2) - \mu_a^2) = \Delta t^2\sigma_a^2 \quad \text{(B.6)}$$

$$\begin{aligned} COV(x,\dot{x}) = COV(\dot{x},x) = E(x\dot{x}) - \mu_x\mu_{\dot{x}} = E(\frac{1}{2}a\Delta t^2 a\Delta t) - (\frac{1}{2}\mu_a\Delta t^2\mu_a\Delta t) = \\ \frac{\Delta t^3}{2}(E(a^2) - \mu_a^2) = \frac{\Delta t^3}{2}\sigma_a^2 \end{aligned} \quad \text{(B.7)}$$

Using the aforementioned terms, the final formulation of Q exists as described in Equation B.8, where σ_a is the user defined variance mentioned in Section 4.2.2.3, Equation 4.4.

$$Q = \begin{bmatrix} \frac{\Delta t^4}{4}\sigma_a^2 & \frac{\Delta t^3}{2}\sigma_a^2 & 0 & 0 & 0 & 0 \\ \frac{\Delta t^3}{2}\sigma_a^2 & \Delta t^2\sigma_a^2 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{\Delta t^4}{4}\sigma_a^2 & \frac{\Delta t^3}{2}\sigma_a^2 & 0 & 0 \\ 0 & 0 & \frac{\Delta t^3}{2}\sigma_a^2 & \Delta t^2\sigma_a^2 & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{\Delta t^4}{4}\sigma_a^2 & \frac{\Delta t^3}{2}\sigma_a^2 \\ 0 & 0 & 0 & 0 & \frac{\Delta t^3}{2}\sigma_a^2 & \Delta t^2\sigma_a^2 \end{bmatrix} \quad \text{(B.8)}$$