2018-11-01

Vision-Based Emergency Landing of Small Unmanned Aircraft Systems

Parker Chase Lusk
Brigham Young University

Vision-Based Emergency Landing of Small Unmanned Aircraft Systems

Parker Chase Lusk

A thesis submitted to the faculty of
Brigham Young University
in partial fulfillment of the requirements for the degree of

Master of Science

Department of Electrical and Computer Engineering

Brigham Young University

ABSTRACT

Vision-Based Emergency Landing of
Small Unmanned Aircraft Systems

Parker Chase Lusk
Department of Electrical and Computer Engineering, BYU
Master of Science

Emergency landing is a critical safety mechanism for aerial vehicles. Commercial aircraft have triply-redundant systems that greatly increase the probability that the pilot will be able to land the aircraft at a designated airfield in the event of an emergency. In general aviation, the chances of always reaching a designated airfield are lower, but the successful pilot might use landmarks and other visual information to safely land in unprepared locations. For small unmanned aircraft systems (sUAS), triply- or even doubly-redundant systems are unlikely due to size, weight, and power constraints. Additionally, there is a growing demand for beyond visual line of sight (BVLOS) operations, where an sUAS operator would be unable to guide the vehicle safely to the ground.

This thesis presents a machine vision-based approach to emergency landing for small unmanned aircraft systems. In the event of an emergency, the vehicle uses a pre-compiled database of potential landing sites to select the most accessible location to land based on vehicle health. Because it is impossible to know the current state of any ground environment, a camera is used for real-time visual feedback. Using the recently developed Recursive-RANSAC algorithm, an arbitrary number of moving ground obstacles can be visually detected and tracked. If obstacles are present in the selected ditch site, the emergency landing system chooses a new ditch site to mitigate risk. This system is called **Safe2Ditch**.

Keywords: small unmanned aircraft systems, vision-based navigation, emergency landing, autonomous systems, visual target tracking, multiple object tracking, UAS traffic management

ACKNOWLEDGMENTS

I am grateful to have been associated with Brigham Young University and the MAGICC Lab for the duration of my master's degree. Phenomenal classes, professors, staff, and peers have inspired me to continually expand my own understanding and to bring others along with me.

In particular, I would like to thank my advisor Dr. Randy Beard for his example and patience. I hope that I can take with me even a fraction of his passion for digging deep into the mathematical underpinnings of topics and his ability to clearly define the geometry of problems. He has been a great guide throughout my intellectual journey at BYU.

Dr. Cammy Peterson and Dr. Tim McLain have also been wonderful examples to me of professors dedicated to serving others through education and research, and I am grateful for my discussions with both of them. Additionally, I am appreciative of other faculty members across campus with whom I have had the privilege of learning from—both in and out of the classroom.

My day-to-day interactions with countless other students have been some of the most meaningful and I am thankful for the opportunities we have had to learn together. Although there are too many to name, I specifically appreciate the insightful exchanges I have had with individuals from the MAGICC Lab about software development, control and estimation theory, computer vision, differential geometry, signals and systems, best practices, autopilot design, space exploration, and career paths. Thank you for being giants.

Finally, I am grateful for my supportive family and especially my wife for her love and encouragement throughout my graduate studies. Thanks for letting me play with robots.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Pilotless flying vehicles have existed in various forms for more than the past century, with one the earliest examples being bomb-equipped balloons used in an 1849 Austrian attack of Venice. For the next 50 years, aviation pioneers worked to demonstrate controlled flight of both manned and unmanned heavier-than-air aircraft. Known as the flying man, German engineer Otto Lilienthal built gliders and successfully flew distances of 82 to 820 ft (25 to 250 m) from 1891-96. These glider aircraft were controlled by changing the center of gravity when Lilienthal shifted his body. In 1896, Samuel P. Langley demonstrated his unmanned, steam-powered *Aerodrome No. 5*, which flew 3,300 ft (1,000 m) and had no flight controls. These efforts culminated in Orville and Wilbur Wright's piloted *Wright Flyer I*, a powered aircraft which flew 1,500 ft (450 m) in December 1903. With this demonstration, the Wright Brothers were the first to perform three-axis control of a flying vehicle.

Inspired by the successes of these early flight innovators, the United States government was eager to gain the upper hand in World War I and commissioned the Hewitt-Sperry Automatic Airplane which had its first flight in September 1917. This aircraft, known as the "flying bomb", was equipped with a Sperry Corporation gyroscopic autopilot and used radio frequency (RF) signals for remote piloting [1]. The following year, the U.S. Army commissioned a second unmanned aircraft project resulting in the Kettering Bug, which was an unmanned aerial torpedo and a precursor to the present-day cruise missile. This was arguably the world's first autonomous unmanned aerial vehicle (UAV) and could be mechanically programmed to fly a certain distance before cutting engine power, detaching the wings, and deploying the fuselage, which housed the explosives.

As technology continued to improve and military needs increased, the use of UAVs became more common during wartime. In 1962, the *Ryan Model 147* was created as a reconnaissance UAV for use in the Vietnam War. Commonly known as the "Lightning Bug", this UAV was used for low-altitude photographic and electronic aerial surveillance. As the U.S. military gained confidence in the usefulness of UAVs they were eventually weaponized into unmanned aircraft such as the Predator and Reaper drones, which are in use today.

With this brief historical sketch of the development of unmanned flight, we move on to discuss the main type of system considered in this thesis: small unmanned aircraft systems (sUAS). An unmanned aircraft system (UAS) is defined by statute as an aircraft that is operated without the possibility of direct human intervention within or on the aircraft [2]. The UAS is made up of a UAV with its sensor payload and any ground station or other infrastructure needed for communication. The designation of "small" in sUAS is applied whenever the aircraft is about the same size as a model aircraft, typically below 55 lb (25 kg). While military use has historically been the primary driving force in the development of UAS, the past 10 years have especially seen increased hobby, commercial, and civil interest in sUAS. Interestingly, the development of sUAS has been made possible by consumer electronics such as smart phones and gaming computers. These platforms have increased the availability of high-quality, low-cost micro-electro-mechanical systems (MEMS) sensors, low-power processing units, and highly-parallel computing devices such as general-purpose graphics processing units (GPGPUs) [3]. With lower economical and technological barriers to entry, a host of sUAS applications have emerged, such as infrastructure inspection, traffic monitoring, package delivery, natural disaster assessment, search and rescue activities, and others. A handful of private companies have even begun exploring larger UAS for the purpose of mobility-on-demand using autonomous personal flying vehicles.

Although the popularity and potential of sUAS continues to grow, a major hurdle for governments and industry to overcome will be the integration of sUAS into the national airspace system (NAS). In a five-year forecast published by the U.S. Federal Aviation Administration (FAA) last year, the FAA projected that the 1,142,000 sUAS units in 2016 will more than triple to 3,972,000 units by 2021 [2]. In addition to the approximately 5,000 manned aircraft in the NAS during peak times [4], integration of sUAS into the NAS raises the question of how to effectively deal with such a sharp increase of NAS traffic. To this end, NASA and the FAA have been working on a UAS traffic management (UTM) concept since 2015 [5]. UTM is a traffic management ecosystem for low-altitude sUAS operations in uncontrolled airspace (below 400 ft, 120 m). Its purpose is to connect UAS operators with real-time data from other UAS in the vicinity, weather and terrain data, and an overall NAS regulator (e.g., the FAA). This architecture aims to maximize information flow between users, giving flexibility to operators and their sUAS operations when working in low-risk or isolated areas and providing support and structure when multiple sUAS operate in densely populated areas.

1.1 Motivation for Emergency Landing Systems

Integration into the NAS as suggested by UTM requires that unmanned systems flying beyond visual line of sight (BVLOS) have much more capable on-board autonomy—not only to carry out specific missions, but to enable safety critical components such as sense and avoid, forced landing, operation in GPS degraded environments, and communication-loss mitigation. These safety concerns must be fully addressed and standardized across sUAS in the NAS. The main topic of this thesis is an emergency landing scheme for sUAS operating at low-altitudes in the NAS.

In manned aircraft, emergency landing is a crucial aspect of flight safety. When technical problems such as engine failure or structural damage occur, the pilot must quickly select a suitable landing site, which may not be at a designated airstrip. Using communication with air traffic control, a safe landing zone is chosen that is clear of people and other obstacles. In the case of uncontrolled general aviation airspace, such as FAA Class G airspace, it is the pilot's responsibility to visually locate a safe landing site clear of people and other obstacles. Performing forced landing maneuvers at low-altitude for unmanned aircraft requires the on-board system to be able to perceive the environment and make decisions on a landing site autonomously. Additionally, the size, weight, and power constraints of sUAS prevents the use of double- or triple-mode redundancy as is common in manned aircraft. This results in the need of a quick reaction time for sUAS emergency landing systems.

A successful model for autonomous interaction with the environment can be found in the sensory-based decision making process known as *situation awareness* [6]. Originating in the military aircraft pilot community, situation awareness is defined as the human process of perceiving details in the environment, comprehending how those details affect the current goal, and projecting that comprehension to what will happen in the near future [7]. This process of situation awareness allows human operators to make critical decisions in a timely and effective manner. Similarly, adding a sense of situation awareness in autonomous vehicles allows them to operate effectively in dynamic environments [8]. For example, [9] discuss adding artificial situation awareness to UAS by using automated dependent surveillance-broadcast (ADS-B) to cooperatively sense other agents while landing at prepared sites.

The goal of this thesis is to present a sUAS emergency landing framework suitable for multirotors. This framework, which originated from engineers at NASA Langley Research Center, Hampton, VA, is called *Safe2Ditch*. In this thesis, Safe2Ditch is implemented with vision capabilities and tested in both simulation and hardware. The development of the visual tracking system that gives Safe2Ditch vision capabilities is discussed in Chapter 2 and Chapter 3. The current state of the Safe2Ditch system is discussed in more detail in Chapter 4 and a conclusion is given in Chapter 5.

1.2 Summary of Contributions

In addition to the demonstration of a vision-based emergency landing system for multirotors, this thesis presents other research and engineering contributions. These contributions are listed below in two lists: first, Safe2Ditch-specific contributions, and second, general contributions that add value to the BYU MAGICC Lab.

Safe2Ditch-specific contributions:

- Python implementation of the Ditch Site Selector component of Safe2Ditch.
- Altitude-dependent tuning to increase tracking consistency during descent.
- ROS/Gazebo software-in-the-loop (SIL) environment for Safe2Ditch. Simulations use 3D models of moving people and satellite imagery to recreate test environments at Rock Canyon Park, Provo, UT and NASA Langley Research Center, Langley, VA.
- Monte Carlo SIL simulation framework to test the Safe2Ditch system from takeoff to landing in realistic environments.
- Hardware demonstrations of Safe2Ditch with the visual multiple target tracking system that tracks obstacles in potential landing zones.

General lab contributions:

- Real-time implementation of Recursive-RANSAC Tracker using C++11 and modern software development practices. Additionally, software bindings for MATLAB and Python have been produced for quick prototyping and testing of Recursive-RANSAC.
- Implementation of Visual MTT, the visual front end of R-RANSAC. Computational efficiency was enhanced by leveraging GPGPU processing via OpenCV and CUDA.
- ROS/Gazebo SIL environment for Pixhawk/APM-based multirotors.

1.3 Thesis Outline

This thesis focuses on the components needed for a reliable sUAS emergency landing system. Chapter 2 discusses the Recursive-RANSAC estimation algorithm and its real-time implementation. Recursive-RANSAC was developed in the Multiple Agent Intelligent Coordination and Control (MAGICC) lab at Brigham Young University (BYU) and has been applied in this thesis to multiple target tracking (MTT) using a camera on board a multirotor.

In Chapter 3, the Visual MTT algorithm is extended to allow consistent and continuous tracking of moving ground objects while descending using an altitude-dependent tuning scheme. Using the previously mentioned real-time Recursive-RANSAC and Visual MTT implementations, results from hardware test flights are presented that show how tracking while descending can be achieved.

Chapter 4 discusses the Safe2Ditch emergency landing system and how augmenting a pre-compiled database of potential landing sites with the visual tracker developed in Chapters 2 and 3 allows sUAS to react to non-cooperative moving obstacles while landing in populated areas. The Ditch Site Selection component of Safe2Ditch and the visual tracking system are tested with 2000 end-to-end Monte Carlo simulations and 16 hardware flights with a multirotor.

Finally, Chapter 5 concludes with a summary and future research directions of topics discussed in this thesis.

Chapter 2

The Recursive-RANSAC Tracker

Recursive-RANSAC (R-RANSAC) is an online estimation algorithm developed primarily by Peter Niedfeldt in the BYU MAGICC Lab from roughly 2012-2014 [10]. Its purpose is to estimate the states of an arbitrary number of dynamic targets based on noisy observations [11]. Follow-on work by Kyle Ingersoll [12, 13] and Patrick DeFranco [14] (2013-2015) applied R-RANSAC to vision-based multiple target tracking. In addition to visual measurements, the R-RANSAC algorithm has been applied to radar data for state estimation of UAVs [15] and for sUAS sense and avoid systems [16]. R-RANSAC has also been applied to fiber optic spectral response data by the BYU Fiber Devices Lab for tracking optical peaks caused by the reflection of fiber Bragg gratings.

Due to the applicability of R-RANSAC to different measurement data, various extensions and implementations have been created. While these implementations have been mostly sufficient for their use cases, the lack of a single efficient and well-documented implementation has been a challenge for new researchers. The purpose of this chapter is to present the theory of R-RANSAC and a real-time C++ implementation with unified notation and terminology. The C++ implementation of R-RANSAC is referred to as the **R-RANSAC Tracker**, while the core estimation algorithm is referred to as simply **R-RANSAC**. Additionally, the design and implementation of a visual measurement front end is discussed in Section 2.4 and the application of these algorithms is demonstrated in Chapter 3.

2.1 Preliminaries

In its most basic form, R-RANSAC can be thought of as a tool for regression analysis [17]. In classical regression, the goal is to fit a parametric model to observations of some underlying process given a set of inputs. The parametric model is an often idealized mathematical representation of the underlying data-generating process. Fitting a parametric model to a dataset produces an estimate of model parameters. When the parametric model is instantiated with a set of estimated model parameters, a *fitted model* is created. This fitted model can then be used to extrapolate, interpolate, or otherwise perform an analysis of the underlying process.

Let a parametric model be denoted $f(\mathbf{u}[k], \beta) \in \mathbb{R}^m$, where $\mathbf{u} \in \mathbb{R}^p$ is the input, $k \in \mathbb{R}$ is the discrete-time index, and $\beta \in \mathbb{R}^b$ represents the model parameters. We assume there is measurement noise such that the observation of this parametric model at each time step k is given by

$$\mathbf{y}[k] = f(\mathbf{u}[k], \beta) + \eta[k], \tag{2.1}$$

where $\eta \in \mathbb{R}^m$ is sampled from a zero-mean Gaussian distribution with covariance Σ_η. In addition to noisy observation of the true signal $f(\mathbf{u}[k], \beta)$, random observations called gross errors, or outliers, are detected according to a Poisson distribution with parameter λ_{GE}, where λ_{GE} is the expected number of gross errors per time step. We assume that the spatial distribution of gross errors is independent and uniform. At each time step k, a sensor scan $Z_k = Y_k \cup \Theta_k$ is produced, where $Y_k = \{\mathbf{y}[k]\}$ and Θ_k contains detected gross errors. The total number of measurements in a sensor scan is denoted ψ_k and is equal to the cardinality of the set Z_k, denoted $\psi_k = |Z_k| = |Y_k| + |\Theta_k|$. Elements of the sensor scan are unordered (i.e., it is not known which are gross errors and which are valid measurements) and are written as $Z_k = \{\mathbf{z}_j[k]\}_{j=1}^{\psi_k}$. The set of all sensors scans up to time index k is denoted as $\mathcal{Z}_k$, which can be flattened into a column vector, written as $\widetilde{\mathcal{Z}}_k$.

Assuming that the chosen parametric model appropriately represents the observed underlying process, an estimate $\hat{\beta}$ of the unknown parameters may be found that minimizes the modeling error in some optimal sense. Using the fitted model, expected measurements can be produced as

$$\hat{\mathbf{y}}[k] = f(\mathbf{u}[k], \hat{\beta}). \tag{2.2}$$

The modeling error can then be computed in the root mean squared error (RMSE) sense as

$$RMSE = \sqrt{\frac{1}{N}\sum_{n=1}^{N}(\hat{\mathbf{y}}[n] - \mathbf{y}[n])^2}, \tag{2.3}$$

where N is the number of sensor scans and $\mathbf{y}[n]$ in this equation is understood to be the noisy observation of the true process and not clutter.

As an example, let $f(\mathbf{u}[k], \beta) = F\beta$ where $F = \begin{bmatrix}\mathbf{u}[k] & 1\end{bmatrix}$, $\beta = \begin{bmatrix}-1 & 7\end{bmatrix}^\top$, $\sigma_\eta = 0.5$, $\lambda_{\text{GE}} = 0.5$, and $\mathbf{u}[k] \sim \mathcal{U}(-10, 10)$. Thus, observations of the underlying process can be simulated as shown in Figure 2.1. For the purposes of simulating data, the underlying process is chosen to be a parametric model with a set of true parameters, β. The next two subsections present standard practices in fitting this parametric model to noisy observations, resulting in estimated model parameters, $\hat{\beta}$, and a fitted model, $f(\mathbf{u}[k], \hat{\beta})$.

2.1.1 Least Squares Parameter Estimation

The most common method for estimating parameters of a signal with constant parameters is least squares (LS). Using LS, a linear fit may be found that minimizes the sum of squared

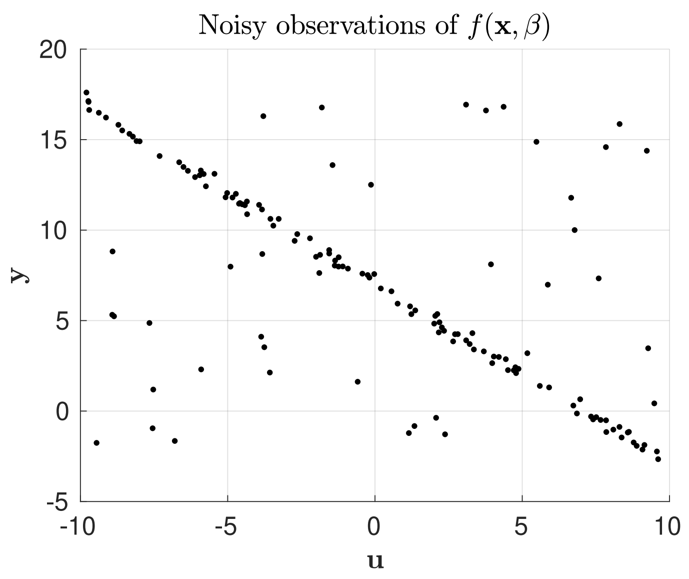


Figure 2.1: Simulated data from observation of an underlying process. Outliers are detected in addition to noisy observations of a true linear signal.

error between the fitted model and the data of an over-determined system. This estimator is written as

$$\hat{\beta}_{\text{LS}} = F^{\dagger}\widetilde{\mathcal{Z}}_k = (F^{\top}F)^{-1}F^{\top}\widetilde{\mathcal{Z}}_k, \tag{2.4}$$

where the $\cdot^{\dagger}$ operator denotes the Moore-Penrose pseudoinverse.

In the absence of gross errors, LS does well at fitting a parametric model of a line to the data, as shown in Figure 2.2a. The estimated parameters using LS with no gross errors are found to be $\hat{\beta}_{\text{LS}_1} = \begin{bmatrix} -1.00 & 7.04 \end{bmatrix}^{\top}$ with an RMSE of 0.064. However, when the observations include gross errors due to sensor processing or environment noise, the estimated parameters are $\hat{\beta}_{\text{LS}_2} = \begin{bmatrix} -0.60 & 7.06 \end{bmatrix}^{\top}$ with an RMSE of 5.12 (see Figure 2.2b). Because LS uses all of the data, the gross errors can significantly affect the fitted model. The next subsection presents a scheme for classifying gross errors as outliers and ignoring them when fitting the parametric model.

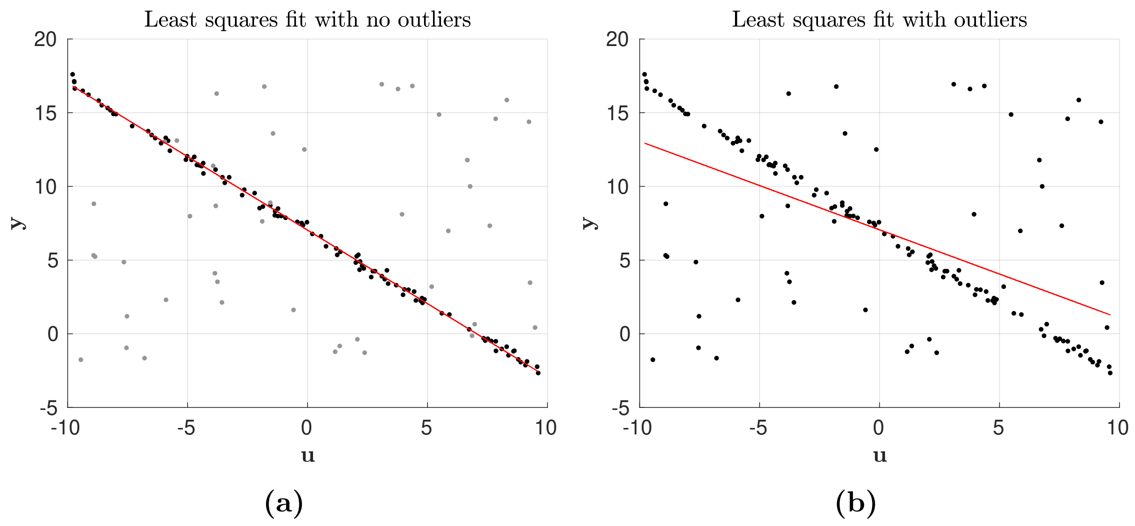


Figure 2.2: LS applied to the dataset observed according to equation (2.1). When gross errors are present in the data as in **(b)**, LS does not fit the parametric model very well.

2.1.2 Random Sample Consensus (RANSAC) Parameter Estimation

In the presence of noise, a greater amount of data is typically preferred because a better fit can be achieved. However, gross errors degrade the result, as seen from performing LS in Figure 2.2b. To increase robustness in the presence of gross errors, Fischler and Bolles [18] developed an iterative algorithm known as random sample consensus (RANSAC).

The goal of RANSAC is to use a voting scheme to classify the gross errors as *outliers* and to use the remaining data to fit the given parametric model. This is achieved by using randomly chosen minimum subsets of the data to create a temporary fitted model, called a *hypothesis model*. To fit a parametric model with parameter space $\mathbb{R}^b$ to the observations, at least b data points are required in the minimum subset. Each hypothesis model is then scored based on how consistently it explains the rest of the data and data points that are associated with a hypothesis model are called *inliers*. The hypothesis model with the most inliers wins and those inliers can then be used to perform additional smoothing if required (e.g., using LS), resulting in a final fitted model with estimated parameters $\hat{\beta}$. Concretely, the steps performed by RANSAC in a single iteration are

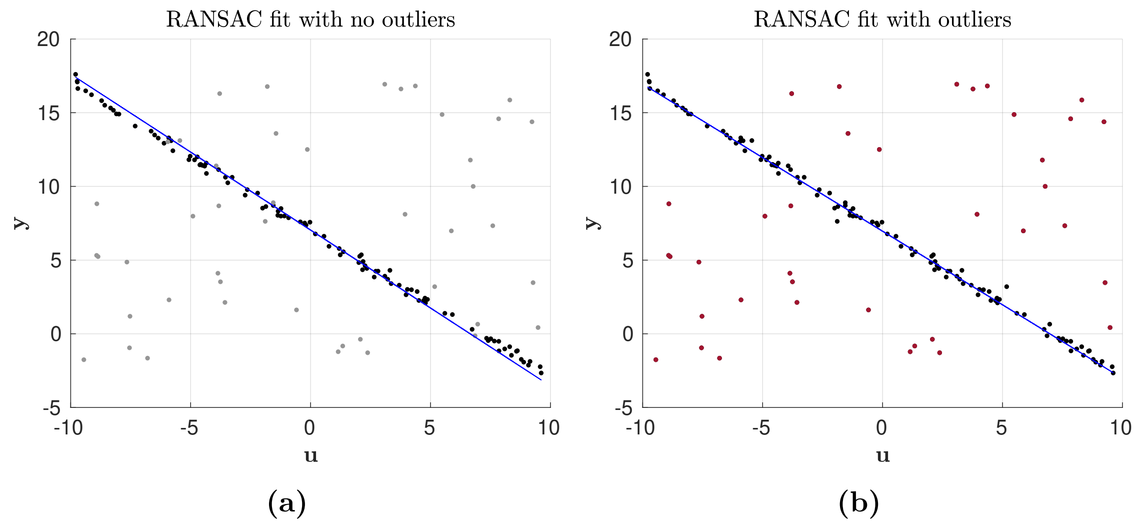


Figure 2.3: RANSAC applied to the dataset observed according to equation (2.1). Unlike LS, gross errors in the data do not affect RANSAC's parameter estimation. Note that RANSAC has classified the colored points as outliers in **(b)**.

1. Randomly sample the minimum number of data points needed to estimate model parameters, $\hat{\beta}$. This minimum subset is called the *hypothetical inliers*.

2. Create a hypothesis model using the hypothetical inliers.

3. Compare all other data points with the hypothesis model using an *inlier function* specific to the parametric model being used (e.g., Euclidean distance for an affine model or reprojection error for a homography). Any data point that fits the hypothesis model well according to the inlier function are inserted into the *consensus set*.

4. If this hypothesis model has more members in the consensus set than the result of the last iteration, then use this hypothesis model for the next iteration.

These steps are repeated for ℓ iterations or until a sufficiently good hypothesis model has been created. As previously mentioned, the consensus set of the best hypothesis model is then used to create a final fitted model.

When RANSAC is applied to the dataset without gross errors, the estimated parameters are found to be $\hat{\beta}_{\text{RANSAC}_1} = \begin{bmatrix} -1.06 & 7.04 \end{bmatrix}^\top$ with an RMSE of 0.18 (see Figure 2.3a). Alternatively, when the dataset with gross errors is used, we see that RANSAC is able to reject outliers and the estimated parameters are found to be $\hat{\beta}_{\text{RANSAC}_2} = \begin{bmatrix} -1.00 & 6.96 \end{bmatrix}^\top$ with RMSE of 0.07 (see Figure 2.3b). Note that RANSAC classified data further than 1 unit away from the resulting fitted model as outliers (denoted by color).

2.1.3 Summary

As we can see, RANSAC is a robust method for estimating the parameters of a single stationary signal in clutter. However, we would like to estimate the parameters of multiple non-stationary signals in clutter. Stationary signals are stochastic processes whose parameters such as mean and variance do not change over time. In other words, stationary signals have a constant β, while non-stationary signals have parameters that can vary over time. Fortunately, R-RANSAC allows us to estimate signals with time-varying parameters, known as *state*.

Additionally, note that the data in the previous model fitting examples was assumed to be available all at once, i.e., in a single batch. In the real-time estimation problems that we are interested in, the parameters must be estimated *online*, meaning that new data is received with each time step. An extension to LS called recursive least squares (RLS) exists that is able to efficiently incorporate new data with what is known as a rank-one update for matrix inversion [19]. The goal of R-RANSAC is to provide a recursive extension to RANSAC, just as RLS is a recursive extension to LS. The R-RANSAC algorithm is developed in the next section.

2.2 Formulation of Recursive-RANSAC

2.2.1 Problem Setup

Recursive-RANSAC is able to estimate the parameters of an arbitrary number of non-stationary processes in clutter. In target tracking, non-stationary underlying processes are dynamic targets such as people, animals, or vehicles. To apply R-RANSAC to target tracking, let $M[k]$ be the number of true targets at each time step k. Denote the state (i.e., the time-varying parameters such as position and velocity) of the i^{th} target as $\mathbf{x}_i \in \mathbb{R}^n$. The discrete-time dynamics of the i^{th} target are modeled as

$$\mathbf{x}_i[k] = A\mathbf{x}_i[k-1] + \mathbf{w}_i[k], \tag{2.5}$$

where $A \in \mathbb{R}^{n\times n}$ is the *motion model* and $\mathbf{w}_i \in \mathbb{R}^n$ is a wide-sense stationary (WSS), zero-mean Gaussian process with covariance Q that captures uncertainties in modeling. This discrete-time difference equation illustrates how the state of the underlying process evolves with time according to a given motion model.

At each time step k, an observation of each target is attempted. Noisy observations in the measurement space denoted by $\mathcal{R}_{\mathbf{y}} \subset \mathbb{R}^m$ are received via sensors. The probability that the i^{th} target is detected is modeled by the random process $\mathcal{P}_k^i(\omega)$, where the outcome ω lies in the universe of outcomes $\Omega = \{0, 1\}$ and $\omega = 0$ indicates that no measurement was received. The distribution over the universe of outcomes for the i^{th} target is assumed to be Bernoulli with parameter p_d^i. Using this model and given the current state $\mathbf{x}_i$ of the i^{th} target, measurements $\mathbf{y}_i \in \mathcal{R}_{\mathbf{y}}$ of the i^{th} target are received according to

$$\mathbf{y}_i[k] = \begin{cases} \varnothing, & \omega = 0 \\ C\mathbf{x}_i[k] + \mathbf{v}_i[k], & \omega = 1 \end{cases}, \tag{2.6}$$

where $C \in \mathbb{R}^{m \times n}$ is the *measurement model* and $\mathbf{v}_i \in \mathbb{R}^m$ is a WSS, zero-mean Gaussian process with covariance R that represents sensor noise. In other words, with probability p_d^i, a measurement for the i^{th} target is received and with probability $1 - p_d^i$ the sensor fails and no measurement is received at time step k.

In addition to a noisy observation of the i^{th} target $\mathbf{y}_i[k]$, random observations called gross errors, or outliers, are detected according to a Poisson distribution with parameter λ_{GE}, where λ_{GE} is the expected number of gross errors per time step. We assume that the spatial distribution of gross errors is independent and uniform. At each time step k, a sensor scan $Z_k = Y_k \cup \Theta_k$ is produced, where $Y_k = \{\mathbf{y}_i[k]\}_{i=1}^{M[k]}$ and Θ_k contains detected gross errors. The total number of measurements in a sensor scan is equal to the cardinality of the set Z_k, denoted $\psi_k = |Z_k| = |Y_k| + |\Theta_k|$. Elements of the sensor scan are unordered (i.e., it is not known which are gross errors and which are valid measurements of targets) and are written as $Z_k = \{\mathbf{z}_j[k]\}_{j=1}^{\psi_k}$. For convenience, define $k_N \triangleq k - N_{\text{w}} + 1$ as the initial time index of a N_{w}-length measurement window ending at the current time step k. Because storing all of the data is infeasible, let $\mathcal{Z}_k = \{Z_{k_N}, \dots, Z_k\}$ be a sliding dataset of the N_{w} most recent sensor scans. This sliding window of data can be "flattened" into a column vector, written as $\widetilde{\mathcal{Z}}_k = \begin{bmatrix} \mathbf{z}_1[k_N] & \cdots & \mathbf{z}_{\psi_{k_N}}[k_N] & \cdots & \mathbf{z}_1[k] & \cdots & \mathbf{z}_{\psi_k}[k] \end{bmatrix}^\top$.

Note that equations (2.5) and (2.6) constitute a state-space model. This state-space model has the same role as equation (2.1), rewritten here for convenience

$$\mathbf{y}[k] = f(\mathbf{u}[k], \beta) + \eta[k].$$

This equation is the noisy observation of the parametric model $f(\mathbf{u}[k], \beta)$ with constant parameters. Similar to η, the noise processes $\mathbf{w}_i$ and $\mathbf{v}_i$ in equations (2.5) and (2.6) capture the uncertainty associated with modeling and measuring. In contrast to the regression example in Section 2.1, notice how the state-space model for dynamic targets we have developed does

Table 2.1: Notation and terminology comparison of stationary vs non-stationary estimators.

Terminology	**LS/RANSAC**	**R-RANSAC**
model parameters	$\beta \in \mathbb{R}^b$	$\mathbf{x}_i \in \mathbb{R}^n$
parametric model	$f(\mathbf{u}[k], \beta)$	state space model $\mathbf{x}_i[k] = A\mathbf{x}_i[k-1]$ $\mathbf{y}_i[k] = C\mathbf{x}_i[k]$
fitted model	$f(\mathbf{u}[k], \hat{\beta})$	$(A, C, \hat{\mathbf{x}}_i)$
input	$\mathbf{u} \in \mathbb{R}^p$	—
observation	$\mathbf{y} \in \mathbb{R}^m$	$\mathbf{y}_i \in \mathbb{R}^m$
sensor measurement	$\mathbf{z}_i \in \mathbb{R}^m$	$\mathbf{z}_i \in \mathbb{R}^m$

not contain an input $\mathbf{u} \in \mathbb{R}^p$. This is because we are attempting to track non-cooperative targets (i.e., we do not receive the input used to command a target to move). The analog of the parametric model $f(\mathbf{u}[k], \beta)$ from before is the pair (A, C). Given an estimate of parameters $\hat{\mathbf{x}}_i$ at time step k, the parametric state-space model pair (A, C) can be instantiated to create a fitted state-space model, written as $(A, C, \hat{\mathbf{x}}_i)$. Further, the expected state at the next time step can be predicted using

$$\hat{\mathbf{x}}_i[k+1] = A\hat{\mathbf{x}}_i[k], \tag{2.7}$$

and expected measurements can be produced as

$$\hat{\mathbf{y}}_i[k+1] = C\hat{\mathbf{x}}_i[k+1]. \tag{2.8}$$

A complete notation and terminology comparison between traditional regression and estimating the state of multiple dynamic targets is given in Table 2.1.

To solve the problem of tracking an arbitrary number of dynamic targets in clutter, we develop the three principal components of the R-RANSAC Tracker in a bottom-up approach. We start with the development of RANSAC for dynamic targets, which is used to initialize fitted state-space models. These fitted models are managed by R-RANSAC, which uses a bank of Kalman filters to propagate and correct them according to new observations. Finally, we discuss the R-RANSAC Tracker, which is a top-level algorithm that handles the real-time aspect of the data collection and manipulation. These three algorithms are discussed in the following three subsections.

2.2.2 RANSAC for Dynamic Targets

At a high level, RANSAC is simply an iterative algorithm that can be used to fit various types of parametric models in the presence of outliers. To effectively use RANSAC to estimate the states of multiple targets, we must first understand how to fit a single dynamic parametric model without outliers. The result of this step is an estimate of the single target's state at the beginning of the N_{w}-length window of observations.

Before this discussion, we make an observation about motion models and their use in state-space equations. A common, low-order dynamic model used in state-space equations for estimating maneuvering targets is the nearly-constant-velocity (NCV) model [20]. Higher-order models such as the nearly-constant-acceleration (NCA) and nearly-constant-jerk (NCJ) are also used. These motion models are linear and can be written in continuous state-space form, such as

$$\dot{\mathbf{x}} = \underbrace{\begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}}_{A_{\mathrm{NCV}}} \mathbf{x}, \tag{2.9}$$

where $\mathbf{x} \triangleq \begin{bmatrix} x & y & \dot{x} & \dot{y} \end{bmatrix}^\top$ is the state of a target moving in a plane. Because there is no input in equation (2.9), the state will evolve according to the initial conditions. A derivation of the discrete motion models used in R-RANSAC can be found in Mehrotra and Mahapatra [21].

In this section, RANSAC is used to initialize new R-RANSAC tracks in a batch of noisy and cluttered data. Although the R-RANSAC algorithm in Section 2.2.3 uses NCJ motion models to propagate tracks, the implementation of the RANSAC algorithm discussed in this section uses NCV motion models to generate hypotheses, although we wish to track highly maneuverable targets. In practice, this use of NCV models to initialize new tracks is justifiable assuming that the rate at which measurements are received is high enough. In this case, even jerks fit the NCV model over the short N_w-length measurement window. Additionally, as recognized in the the computer vision community [22], it is ideal to require as few measurements in the minimum subset as possible. For a NCV model, only two position measurements are needed.

2.2.2.1 Single Target, No Clutter

Following the development in [11], consider the case where $M[k] = 1 \; \forall k$, $p_d = 1$, and $\lambda_\text{GE} = 0$. These assumptions ensure that one measurement from the single target is received per sensor scan. For the sake of notation consistent with [11], we note that there is no clutter (i.e., $|\Theta| = 0$) and so $Z_k = \{\mathbf{z}[k]\} = Y_k = \{\mathbf{y}[k]\}$. Therefore, in this section we will refer to the scan as simple Y_k, with the sliding dataset written as $\mathcal{Y}_k$ and the flattened column vector as $\widetilde{\mathcal{Y}}_k = \begin{bmatrix} \mathbf{y}[k_N] & \mathbf{y}[k_N+1] & \cdots & \mathbf{y}[k] \end{bmatrix}^\top$ Our objective is to find an estimate $\hat{\mathbf{x}}[k_N]$ of the single target's initial state given a batch of data, $\mathcal{Y}_k$.

Assuming the discrete LTI state-space model of equations (2.5) and (2.6) is consistent with the received batch of measurements, $\mathcal{Y}_k$, we can form a regression problem as we have seen before in Section 2.1. The result of this regression will be the initial target state, $\hat{\mathbf{x}}[k_N]$. We begin by stacking the measurement equation of the state-space model, which gives us a parameterized model of the measurements we expect to receive given an initial state $\mathbf{x}[k_N]$:

$$\begin{bmatrix} \mathbf{y}[k_N] \\ \mathbf{y}[k_N+1] \\ \vdots \\ \mathbf{y}[k] \end{bmatrix} = \begin{bmatrix} C\mathbf{x}[k_N] \\ C\mathbf{x}[k_N+1] \\ \vdots \\ C\mathbf{x}[k] \end{bmatrix} + \begin{bmatrix} \mathbf{w}[k_N] \\ \mathbf{w}[k_N+1] \\ \vdots \\ \mathbf{w}[k] \end{bmatrix}. \tag{2.10}$$

Note how this is a parameterized model similar to the parameterized model in Section 2.1, where the $\beta \in \mathbb{R}^b = \mathbb{R}^{nN_\mathrm{w}}$ is made up of the stacked states. By noting that the states are expected to evolve over the N_w-length dataset as

$$\begin{aligned} \mathbf{x}[k_N] &= I\mathbf{x}[k_N] \\ \mathbf{x}[k_N+1] &= A\mathbf{x}[k_N] + \mathbf{w}[k_N+1] \\ \mathbf{x}[k_N+2] &= A^2\mathbf{x}[k_N] + \mathbf{w}[k_N+2] \\ &\vdots \\ \mathbf{x}[k] &= A^{N_\mathrm{w}}\mathbf{x}[k_N] + \mathbf{w}[k], \end{aligned}$$

we can lower the dimension of the parameter space $\mathbb{R}^b$ and equation (2.10) can be simply parameterized by the initial state $\mathbf{x}[k_N]$. Let $\mathcal{O} \triangleq \begin{bmatrix} C & CA & \dots & CA^{N_\mathrm{w}} \end{bmatrix}^\top$ be an $mN_\mathrm{w} \times n$ matrix that describes the expected measurement received at time step k from an initial state $\mathbf{x}[k_N]$. If $N_\mathrm{w} = n$ then $\mathcal{O}$ is the observability matrix. Define $\xi_k \triangleq GW_k + V_k$, where $V_k \triangleq \begin{bmatrix} \mathbf{v}[k_N] & \dots & \mathbf{v}[k] \end{bmatrix}^\top$, $W_k \triangleq \begin{bmatrix} \mathbf{w}[k_N] & \dots & \mathbf{w}[k] \end{bmatrix}^\top$, and let

$$G \triangleq \begin{bmatrix} 0 & 0 & 0 & \dots & 0 \\ 0 & C & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & CA^{N_\mathrm{w}-2} & CA^{N_\mathrm{w}-3} & \dots & C \end{bmatrix}, \tag{2.11}$$

which is an $mN_\mathrm{w} \times nN_\mathrm{w}$ matrix that describes how the process and measurement noise is propagated from the initial state to the end of the window. Then, equation (2.10) can be written as

$$\widetilde{\mathcal{Y}}_k = \mathcal{O}\mathbf{x}[k_N] + \xi_k. \tag{2.12}$$

Examining the stochastic process $\xi_k \in \mathbb{R}^{mN_\mathrm{w}}$, we see that it is a zero-mean Gaussian process with covariance given by

$$\Xi_k = E[\xi_k \xi_k^\top] = GE[W_k W_k^\top]G^\top + 2E[V_k W_k^\top]G^\top + E[V_k V_k^\top] = G\mathbf{Q}G^\top + \mathbf{R}, \tag{2.13}$$

where the cross-terms cancel because $\mathbf{w}_k$ and $\mathbf{v}_k$ are independent. The covariance matrices $\mathbf{Q}$ and $\mathbf{R}$ are block-diagonal with N_w blocks of Q and R, respectively, along the main diagonal.

Using the new, low-dimensional parameterized model of equation (2.12), we wish to find the initial state $\mathbf{x}[k_N]$ that best fits the dataset. As before, if we assume there is no noise (i.e., $\mathbf{w}_k = \mathbf{v}_k = 0$), we could simply use a LS estimator. However, we can use the statistics of the the noise to find a better model fit to the noise-corrupted dataset. In other words, we can find the *most likely* initial state $\mathbf{x}[k_N]$ given the dataset, a parameterized model, and the covariance of the noise.

The probability density function (pdf) of a realization of noise ξ_k is

$$f_\xi(x) = \frac{1}{\sqrt{(2\pi)^{mN_\mathrm{w}} \det(\Xi_k)}} \exp\left(x^\top \Xi_k^{-1} x\right). \tag{2.14}$$

Thus, the likelihood of receiving the batch of measurements $\widetilde{\mathcal{Y}}_k$ given an initial state $\mathbf{x}[k_N]$ is written

$$\mathcal{L}(\widetilde{\mathcal{Y}}_k \mid \mathbf{x}[k_n]) = \frac{1}{\sqrt{(2\pi)^{mN_\mathrm{w}} \det \Xi_k}} \exp\left(-\frac{1}{2}(\widetilde{\mathcal{Y}}_k - \mathcal{O}\mathbf{x}[k_n])^\top \Xi_k^{-1}(\widetilde{\mathcal{Y}}_k - \mathcal{O}\mathbf{x}[k_n])\right), \tag{2.15}$$

where $\mathcal{L}(\cdot) \in \mathbb{R}$ is known as the likelihood function. Note that this estimator still requires a parameterized model (A, C) to fit to the data. The maximum likelihood estimate (MLE) of $\mathbf{x}[k_N]$ is found by maximizing the likelihood function over the parameters $\mathbf{x}[k_N]$. This is equivalent to maximizing the log-likelihood $\ell(\cdot) = \ln \mathcal{L}(\cdot)$, which is in a more convenient form, given by

$$\ell(\widetilde{\mathcal{Y}}_k \mid \mathbf{x}[k_n]) = -\frac{mN_\mathrm{w}}{2}\ln 2\pi - \frac{1}{2}\ln\det\Xi_k - \frac{1}{2}(\widetilde{\mathcal{Y}}_k - \mathcal{O}\mathbf{x}[k_n])^\top \Xi_k^{-1}(\widetilde{\mathcal{Y}}_k - \mathcal{O}\mathbf{x}[k_n]). \tag{2.16}$$

By taking the derivative and equating it with zero,

$$\mathbf{0} = \frac{\partial \ell}{\partial \mathbf{x}} = \mathcal{O}^\top \Xi_k^{-1}\widetilde{\mathcal{Y}}_k - \mathcal{O}^\top \Xi_k^{-1}\mathcal{O}\mathbf{x}[k_N], \tag{2.17}$$

we can find the MLE as

$$\hat{\mathbf{x}}[k_N] = (\mathcal{O}^\top \Xi_k^{-1}\mathcal{O})^{-1}\mathcal{O}^\top \Xi_k^{-1}\widetilde{\mathcal{Y}}_k. \tag{2.18}$$

The covariance of the MLE is

$$P[k_N] = E[(\hat{\mathbf{x}}[k_N])(\hat{\mathbf{x}}[k_N])^\top]$$

$$
\begin{aligned}
&= (\mathcal{O}^\top \Xi_k^{-1} \mathcal{O})^{-1} \mathcal{O}^\top \Xi_k^{-1} E[\mathcal{Y}_k \mathcal{Y}_k^\top]((\mathcal{O}^\top \Xi_k^{-1} \mathcal{O})^{-1} \mathcal{O}^\top \Xi_k^{-1})^\top \\
&= (\mathcal{O}^\top \Xi_k^{-1} \mathcal{O})^{-1} \mathcal{O}^\top \Xi_k^{-1} \Xi_k \Xi_k^{-1} \mathcal{O} (\mathcal{O}^\top \Xi_k^{-1} \mathcal{O})^{-1} \\
&= (\mathcal{O}^\top \Xi_k^{-1} \mathcal{O})^{-1} (\mathcal{O}^\top \Xi_k^{-1} \mathcal{O}) (\mathcal{O}^\top \Xi_k^{-1} \mathcal{O})^{-1} \\
&= (\mathcal{O}^\top \Xi_k^{-1} \mathcal{O})^{-1}. \qquad (2.19)
\end{aligned}
$$

Notice how the maximum likelihood estimator is similar to the least squares estimator, but weighted by the covariance of the noise in the dataset. Although we have shown how to estimate the state (i.e., the time-varying parameters) of a single non-stationary stochastic process, we have still not resolved the difficulty associated with gross errors. This is accomplished in the next section.

2.2.2.2 Multiple Targets in Clutter

We now let the number of true targets be $M[k] \geq 0$, where the i^{th} target has probability of detection $p_d^i \in [0,1]$ and $\lambda_{\text{GE}} \geq 0$. Under these conditions, multiple measurements are received each time step, and some may be clutter from the faulty sensor. As in the previous section, the objective is to estimate the initial state $\mathbf{x}_i[k_N]$ of the i^{th} target.

We use the RANSAC framework to fit up to ℓ hypotheses to the data, scoring each one to determine if it is a better hypothesis than the others. Each hypothesis is formed using a minimum subset S of $b = \frac{n}{d}$ data points, where n depends on the motion model $A \in \mathbb{R}^{n \times n}$ and d is the dimensionality of the environment. For example, when tracking targets on a plane ($d = 2$) with a nearly-constant-velocity (NCV) motion model, $A_{\text{NCV}} \in \mathbb{R}^{4\times 4}$, the minimum subset would require $b = 2$ points, which is equivalent to the number of points necessary for fitting a line to data. Given the current flattened window of measurements $\widetilde{\mathcal{Y}}_k$ and the total number of measurements $\Psi_k = |\widetilde{\mathcal{Y}}_k|$, the initial state and covariance of a hypothesis can be initialized using MLE. This is done by randomly selecting b points from the dataset and flattening them into $\widetilde{\mathcal{Y}}_k^b$. Using the scan indices of the randomly selected points, $\bar{\mathcal{O}}$, $\bar{G}$, and $\bar{\Xi}$ matrices can be constructed. These matrices have the same purpose as in Section 2.2.2.1, but they are smaller because there are only b measurements being used in $\widetilde{\mathcal{Y}}_k^b$.

After using the minimum subset to find the maximum likelihood estimate $\hat{\mathbf{x}}[k_N]$ and its covariance $P[k_N]$, this hypothesis is scored using all of the measurements in the dataset. By calculating the residual $\mathbf{r} = \widetilde{\mathcal{Y}}_k - \mathcal{O}\hat{\mathbf{x}}[k_N]$, we can determine the inliers to the hypothesis by finding which measurements have a residual magnitude less than some threshold, $\tau_{R,\text{RANSAC}}$. These inliers are stored in the consensus set χ of the hypothesis, which is used to compare hypotheses. If the current hypothesis has a greater inlier ratio, $\frac{|\chi|}{N_\text{w}}$, then it replaces the previous best hypothesis. This process is repeated for ℓ iterations, or until a hypothesis is found with an inlier ratio greater than γ.

Once the best hypothesis is determined, the initial state $\hat{\mathbf{x}}[k_N]$ and the corresponding covariance $P[k_N]$ and consensus set χ are used in a Kalman smoother to instantiate a new R-RANSAC track, denoted $\mathcal{T}_i[k] = (A, C, \hat{\mathbf{x}}_i[k], P_i[k])$.

2.2.3 R-RANSAC Core

The core R-RANSAC algorithm is responsible for using RANSAC for track initialization, managing existing tracks, and determining when a track is good. At each time step k, a new measurement scan Z_k and a transformation T_k is received. These inputs are used to perform the following high-level tasks: (1) track transformation, (2) track propagation, (3) measurement classification, (4) RANSAC initialization using outliers, (5) track correction, and (6) track management. Each of these steps are discussed below.

2.2.3.1 Model Transformation

In this thesis, data primarily comes from vision processing and so the transformation $T_k \in \mathbb{R}^{3\times 3}$ represents a homography, which is a type of perspective projection (see Section 2.4). Alternatively, the transformation $T_k \in \text{SE}(3)$ could represent a 3D rotation and translation in a 4×4 homogeneous matrix. Using the transformation T_k, the data from the previous $N_\text{w} - 1$ scans are transformed into the same coordinate frame as the new scan. By expressing all data in the same coordinate frame, the R-RANSAC Tracker can used when the sensor is rigidly moving with respect to the surveillance region $\mathcal{R}$. Additionally, any R-RANSAC models have their state $\hat{\mathbf{x}}_i$ and covariance P_i transformed using T_k.

2.2.3.2 Propagation

Once all data is in the same coordinate frame, the R-RANSAC Tracker then runs the R-RANSAC estimation algorithm on the updated dataset $\mathcal{Z}_k$ which contains the latest sensor scan Z_k. The first step is to propagate the state and covariance of each R-RANSAC track forward by one time step so that we can predict where new measurements will be. This propagation step is performed using the standard Kalman filter equations for the i^{th} track,

$$\hat{\mathbf{x}}_i^+[k] = A\hat{\mathbf{x}}_i[k-1] \tag{2.20}$$

$$P_i^+[k] = AP_i[k-1]A^\top + Q. \tag{2.21}$$

2.2.3.3 Measurement Classification

Once the predicted state of each track is available, this prediction is used to classify the newest scan of measurements $Z_k \in \mathcal{Z}_k$ as either inliers or outliers. For each of the i R-RANSAC tracks in memory, the expected measurement $\hat{\mathbf{y}}_i[k]$ for the current time step is found using equation (2.8). Then, the Euclidean distance from $\hat{\mathbf{y}}_i[k]$ to the j^{th} measurement $\mathbf{z}_j[k] \in Z_k$ is computed. If this distance is less than the inlier region, defined as a circle with radius τ_R, then the j^{th} measurement is flagged as an inlier to the i^{th} track. These inliers will be used later in the correction step of R-RANSAC. Note that a single measurement could be counted as an inlier to multiple tracks.

2.2.3.4 RANSAC Initialization

If a measurement $\mathbf{z}_j[k]$ is not marked as an inlier to any of the i R-RANSAC tracks, then it is used with RANSAC to initialize a new track (see Section 2.2.2). Instead of randomly selecting b measurements from the flattened dataset $\widetilde{\mathcal{Y}}_k$, the outlier measurement is used and only $b-1$ measurements are randomly chosen from $\bar{\mathcal{Z}}_k$. The function call to the RANSAC routine looks like

$$\mathcal{T} \leftarrow \text{RANSAC}(\bar{\mathcal{Z}}_k, \mathbf{z}_j[k]), \tag{2.22}$$

which returns a new R-RANSAC track $\mathcal{T}$ that will be added to the set of $\mathcal{M}$ tracks $\mathbb{T}$ that R-RANSAC manages. This step is key in allowing operator-free tracker initialization.

2.2.3.5 Correction

Next, each R-RANSAC track is updated using measurements from the current scan that were classified as inliers. The standard Kalman update is given by

$$\begin{aligned} S_i[k] &= CP_i^+[k]C^\top + R && (2.23)\\ K_i[k] &= P_i^+[k]C^\top S_i^{-1}[k] && (2.24)\\ \hat{\mathbf{x}}_i[k] &= \hat{\mathbf{x}}_i^+[k] + K_i[k](\zeta - \hat{\mathbf{y}}_i[k]) && (2.25)\\ P_i[k] &= (I - K_i[k]C)P_i^+[k], && (2.26) \end{aligned}$$

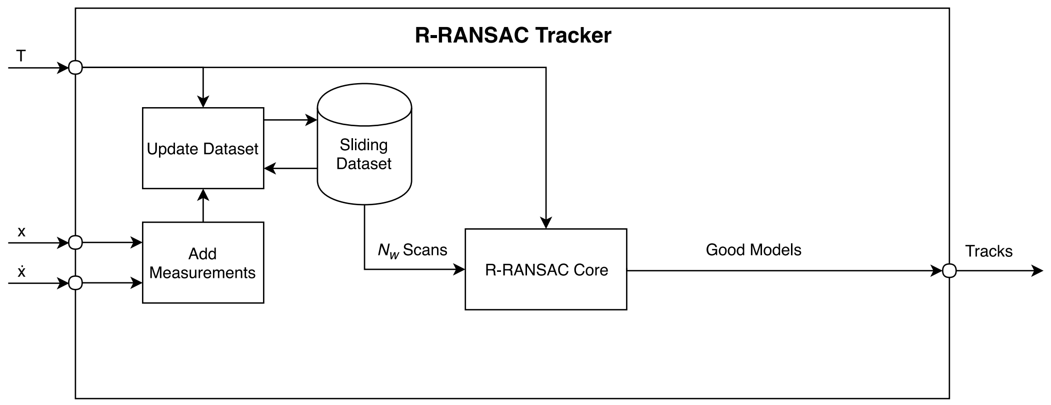


Figure 2.4: Architecture of the R-RANSAC Tracker.

where $S_i[k]$ is the residual covariance, $K_i[k]$ is the optimal Kalman filter gain, and ζ is a valid measurement received from the underlying process. However, because the true data association is unknown in multiple target tracking with gross errors, we use a synthetic measurement residual. The synthetic residual is found by weighting each residual $\mathbf{z}_j[k] - \hat{\mathbf{y}}_i[k]\ \forall j \in \mathcal{I}$ by the normalized likelihood that $\mathbf{z}_j[k]$ would have been received. The likelihood is given by the pdf of a Gaussian with mean $\hat{\mathbf{y}}_i[k]$ and covariance $S_i[k]$. The sum of these weighted residuals gives the synthetic residual s and the state of the i^{th} track is then updated as

$$\hat{\mathbf{x}}_i[k] = \hat{\mathbf{x}}_i^+[k] + K_i[k]s. \tag{2.27}$$

2.2.3.6 Model Management

The last task for R-RANSAC to perform is model management, which prunes tracks that have not received measurements in some time and merges tracks that have similar state. This step allows R-RANSAC to be run with little or no operator intervention and the pruning and merging parameters are tunable.

2.2.4 R-RANSAC Tracker

In addition to real-time tracking of multiple dynamic targets, the so-called R-RANSAC Tracker enables R-RANSAC to be used from a moving sensor. Because R-RANSAC operates on a sliding window of data, it is important for all of the data to be expressed in the same coordinate frame. Using the transform T_k, the R-RANSAC Tracker manages this sliding dataset and transforms older sensor scans into the same coordinate frame as the newest scan. The implementation of the R-RANSAC Tracker is discussed in the following section.

2.3 Implementation

To use Recursive-RANSAC in real-time on-board an autonomous vehicle, it was implemented in `C++11` with modern syntax. The goal of this implementation is to be as much of a one-size-fits-all implementation as is reasonable in a research setting. Having a single codebase allows documentation, issues, questions, and new features to benefit all who use it. In this section, we give a high-level overview of the R-RANSAC Tracker codebase with suggestions on how to maintain it.

The project is implemented as a `C++11` shared library, similar to the style of OpenCV [23] or Eigen [24], as opposed to a collection of source file that are compiled with every project that uses R-RANSAC. Using a shared library allows users to compile R-RANSAC once, install it on their machine, and then use multiple software packages that use the same installation of R-RANSAC. This pattern is used frequently to manage software versions and to lessen compilation cost. Additionally, this pattern accentuates the organization of R-RANSAC and how it is a sensor-modality-agnostic algorithm.

A directory listing of the R-RANSAC Tracker (sometimes referred to as `librransac`) is given in Figure 2.5. In addition to providing the standard `C++` interface through the public header files, bindings for MATLAB and Python 2.7/3 are provided. Using the `CMake` build

```
rransac/
├─ benchmarks/
├─ bindings/
│  ├─ python/
│  └─ matlab/
├─ cmake/
├─ debug/
├─ include/rransac/
│  ├─ access_type.h
│  ├─ core/
│  │  ├─ common.h
│  │  ├─ measurement.h
│  │  ├─ model.h
│  │  ├─ motion_model.h
│  │  ├─ parameters.h
│  │  ├─ ransac_extras.h
│  │  ├─ ransac.h
│  │  ├─ rransac.h
│  │  ├─ sources.h
│  │  └─ utils.h
│  ├─ tracker.h
│  └─ version.h
├─ src/
└─ tests/
```

Figure 2.5: Directory listing of the R-RANSAC Tracker codebase. Only the header files are shown.

system as described in the `librransac` wiki, these extra bindings can be built and installed on the users machine. These bindings are useful for rapid prototyping and quickly interfacing with other types of data.

The public application programming interface used by other software pacakges developing with `librransac` can be found in the `tracker.h` header file. The other header files in `core` are useful for understanding how `librransac` works, but should not be accessed by other software packages. Additionally, the `access_type.h` header file allows software packages to use their own data structure for passing in measurements. Using the template provided, an `AccessType` class is created that specifies how to use the foreign data structure. Internally, `librransac` takes this data and converts it into a `rransac::core::Measurement`, where a scan of measurements is stored as `std::vector<rransac::core::MeasurementPtr>`.

Although not discussed in this thesis, `librransac` allows measurements to come from multiple sources. The i^{th} measurement source is equipped with its own noise statistics, represented in the source covariance matrix $R^{(i)}$. More information about the theory of multiple measurement sources and the centralized information fuser can be found in [25]. Because of this implementation addition, the `parameters.h` header file enables the user to add specific measurement sources that measurements are obtained from. A minimal example of using `librransac` to add a scan of measurements from a single measurement source is given in Listing 2.1.

2.4 Application: Visual Tracking

In this thesis, multiple target tracking (MTT) is performed with a camera to allow a descending multirotor to image potential landing zones and detect moving obstacles. Using the C++ implementation of R-RANSAC discussed in the preceding section and the Robot Operating System (ROS) [26], a visual tracker called **Visual MTT** is implemented for real-time, online processing.

```
#include <rransac/tracker.h>
int main() {
    rransac::Tracker tracker;
    rransac::core::Parameters params;

    // id 0, has velocities, pos and vel variances
    params.add_source(0, true, 5, 1);

    tracker.set_parameters(params);

    // do some processing so that source 0 obtains measurements
    ...

    // Add measurements from source 0; see wiki for Data Accessor info
    tracker.add_measurements<EigenVector2dAccess>(pos, vel, 0);

    // Run the R-RANSAC Tracker
    auto models = tracker.run();
}
```

Listing 2.1: Minimal example of interfacing with `librransac`.

Chapter 3

Visual Multiple Target Tracking From a Descending Aerial Platform[1]

3.1 Introduction

Autonomous vehicles are quickly becoming ideal platforms in research, commercial, military, and civil applications. Typical autonomous systems include self-driving cars, personal air vehicles, and small unmanned aircraft systems (sUAS). As these vehicles are integrated into our society and infrastructure, an increased level of autonomy will be required—both for safety and for mission capability. This higher level of autonomy will allow vehicles to perceive their environment and act within certain parameters to achieve their goal. This construct of sensory-based decision making is modeled by *situation awareness* [6].

Situation awareness (SA) is a term that originated in the military aircraft pilot community and is defined as the human process of perceiving details in the environment, comprehending how those details affect the current goal, and projecting that comprehension to what will happen in the near future [7]. This process of SA allows human operators to make critical decisions in a timely and effective manner. Similarly, adding a sense of SA in autonomous vehicles allows them to operate effectively in dynamic environments [8]. The main contribution of this paper is in enhancing the SA perception stage of sUAS through altitude-dependent visual multiple target tracking during a descent.

[1]This chapter is a modified version of the paper published in the 2018 American Control Conference, written by Parker C. Lusk and Randal W. Beard [27]

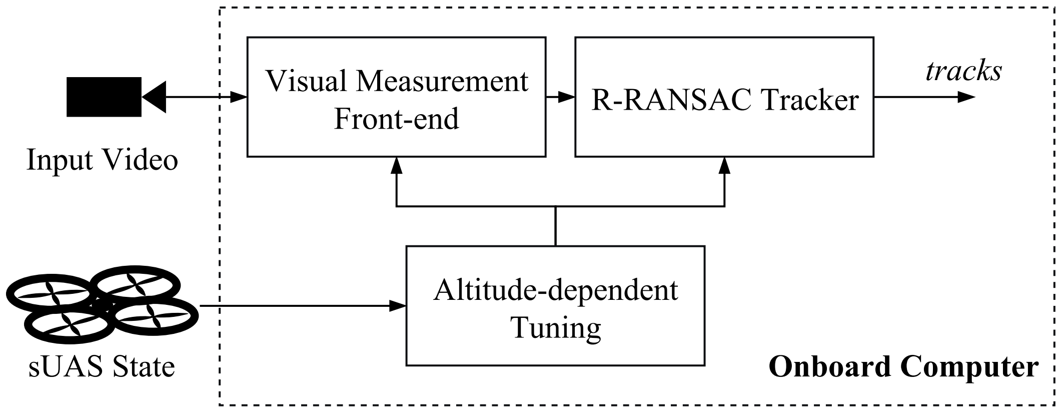


Figure 3.1: Visual tracking system architecture. The altitude of the sUAS is used to tune the parameters of the Recursive-RANSAC visual tracker during a descent. Real-time tracking is performed with an onboard NVIDIA Jetson TX2.

Beyond visual line of sight (BVLOS) operation is an important aspect of truly autonomous vehicles. The challenge of understanding the environment and avoiding obstacles at the beginning and end of a sUAS flight has been referred to as the "first/last 50 feet problem" [5]. Specifically, difficulties arise when sUAS must descend and possibly land in an environment with obstacle uncertainty, such as in the case of forced landings or package delivery [28]. Situation awareness is crucial for these tasks to minimize the risk of damage to moving ground obstacles. In work by McAree et al. [9], the authors simulate UAS during landing at prepared sites in the presence of multiple agents and state uncertainty. The SA of the UAS is enhanced using automated dependent surveillance-broadcast (ADS-B) for cooperative sensing of other agents. The landing approach taken by Scherer et al. [29] uses lidar for non-cooperative sensing to autonomously land a full-scale helicopter at unprepared sites. Mejias et al. [30] demonstrate a camera-based landing zone detection algorithm for UAS in emergency landing scenarios. In our work, a visual multiple target tracker is used to perceive information about moving ground targets. We focus on the target tracking information needed to refine an unprepared landing site that has already been selected by other means.

Visual target tracking has been an active area of computer vision research [31, 32, 33]. Tracking can focus on single or multiple objects and can take place on a static or moving platform. Initialization of tracking typically falls in the category of tracking-by-detection or detection-free tracking. Tracking-by-detection performs frame-to-frame association of detections generated by pre-trained object classifiers [34], feature descriptors [35], or even a simple corner detector [36]. Conversely, detection-free tracking must be manually initialized, as in the work of Kalal et al. [37].

Visual tracking gives sUAS situation awareness and can enable many autonomous applications. Cameras are ideal sensor packages for sUAS because they are low-cost, small, and rich with visual information. In the work of Thomas et al. [38], a small quadrotor with a downward-facing camera is used to track and follow a single object with known geometry. Teulière et al. [39] also track and follow a single object using a template image, removing the need of prior target geometry. A color-based tracker and a particle filter are used, allowing tracking robustness in partial or full occlusion of the target selected by the user. Pestana et al. [40] performs visual servoing from a sUAS by tracking a single user-specified object.

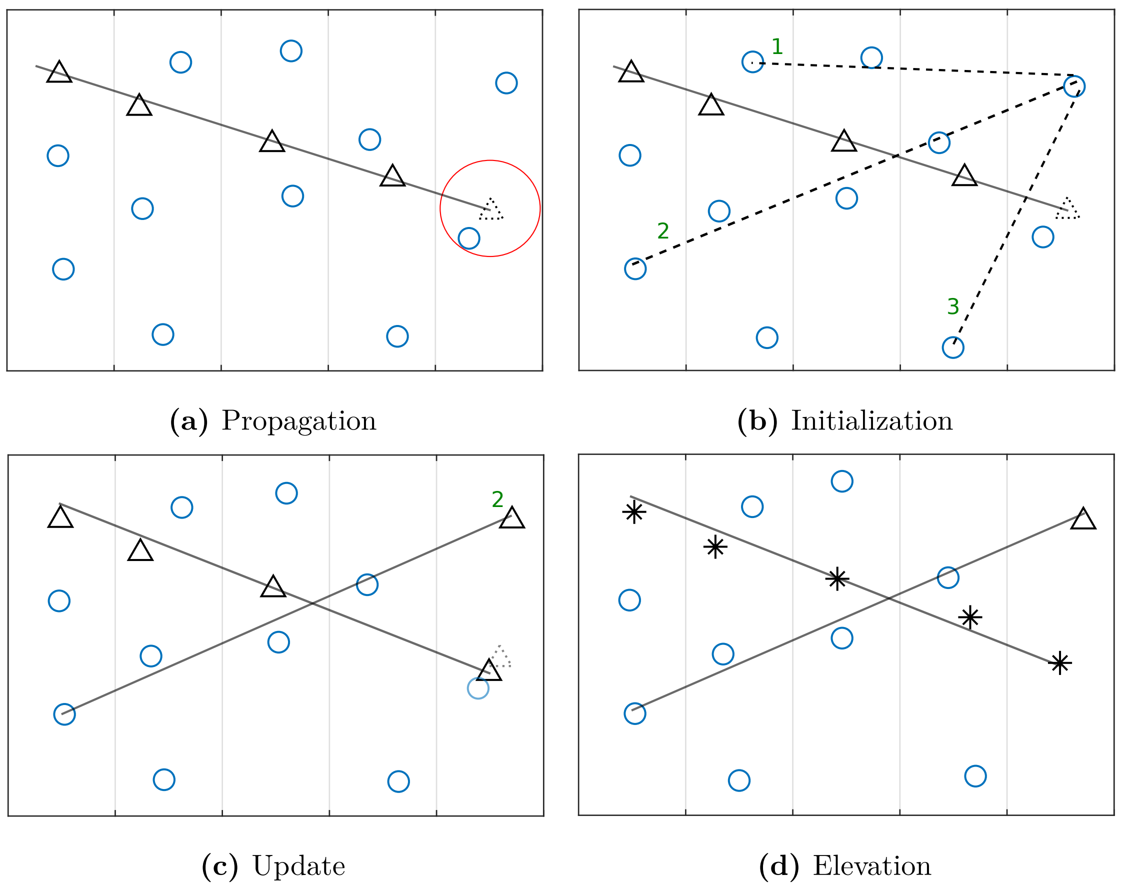
Multiple object tracking (MOT) from a moving camera is of particular interest in sUAS applications because it adds robustness to surveillance techniques where the number of targets or the nature of the camera motion is not known beforehand. Rodriguez-Canosa et al. [41] use Parallel Tracking and Mapping (PTAM) [42] for motion estimation which is then used to create an artificial optical flow field. The difference between Lucas-Kanade optical flow and the artificial flow field exposes dynamically moving objects; however, this technique requires initialization using a marker map and may struggle to detect objects moving with similar velocity to the multirotor. Jiang and Cao [43] are able to track multiple objects in post-processed aerial video using detections based on background modeling, but do not use Bayesian filtering for track management. Li et al. [44] successfully track other sUAS for sense and avoid applications in post-processed aerial footage using a similar visual tracking front-end to ours; however, they assume targets are non-deformable which limits the types of objects that can be tracked. Teutsch and Krüger [45] post-process aerial videos and demonstrate traffic surveillance and tracking from a constant altitude. Their approach is most similar to our tracking pipeline, but assumes constant altitude stable flight over structured environments, which results in smooth video input.

We perform real-time visual multiple object tracking onboard a sUAS using an algorithm known as the Recursive-RANSAC (R-RANSAC) visual tracker. In contrast to existing tracking solutions, our algorithm runs on a moving platform, requires no manual initialization, can run in real-time during a flight, and can track an arbitrary number of moving objects in clutter. In addition, we propose an altitude-dependent tuning scheme that increases track continuity during a descent, as shown in Figure 3.1. This coupling of aircraft state and tracker increases the situation awareness of the sUAS allowing future work to autonomously plan safe trajectories during the “last 50 feet” [5].

The organization of this paper is as follows. Section II gives an overview of R-RANSAC and the visual measurement front-end. In Section III we present the proposed tuning scheme to increase track continuity during a descent. Section IV discusses the hardware implementation and demonstration. Tracking results are reported and discussed in Section V and future research directions are given in Section VI.

3.2 Visual Tracking

As shown in Figure 3.1, the visual multiple-target tracker used in this work consists of two major components: the visual measurement front end and the Recursive-RANSAC (R-RANSAC) tracker. Using the visual tracker in conjunction with an altitude-dependent tuning scheme allows the algorithm to continuously track objects as the sUAS descends. We discuss these components in the following subsections.



(a) Propagation **(b)** Initialization **(c)** Update **(d)** Elevation

Figure 3.2: The four steps of R-RANSAC demonstrated on a surveillance region in $\mathbb{R}^2$. Timesteps are denoted by the grey vertical lines, the current timestep is rightmost. Measurements ($\circ$) may be clutter or from targets. (a) Hypothesis models ($\triangle$) are predicted forward in time (dotted). New measurements are associated as inliers or outliers. (b) Outliers are used to generate RANSAC hypotheses (1, 2, 3). (c) Inlier measurements are used to correct the model prediction. The RANSAC hypothesis with the most support (hypothesis 2) is stored as a model. (d) Models that have good support and have been tracked for a while without too many missed detections are elevated to a track ($*$).

3.2.1 Recursive-RANSAC Tracker

R-RANSAC is an online estimation algorithm capable of tracking an arbitrary number of objects in clutter [11, 10, 12, 46]. Measurements are contained in the surveillance region $\mathcal{R}$ of the system, where $\mathcal{R} \subset \mathbb{R}^2$ in this work. Using the incoming measurements, the algorithm forms hypothesis models that fit the specified motion dynamics. At every timestep, the following four tasks are performed, as shown in Figure 3.2.

Model propagation: Existing models are propagated forward with nearly constant jerk motion dynamics. The new scan of measurements are classified as inliers or outliers to each predicted model position based on Euclidean distance. Measurements within the inlier region defined by a circle of radius τ_R are classified as inliers.

Model initialization: For each measurement that is an outlier to all models, a RANSAC-based initialization step is performed to find models that fit nearly constant velocity dynamics. Only the best RANSAC hypothesis model is kept.

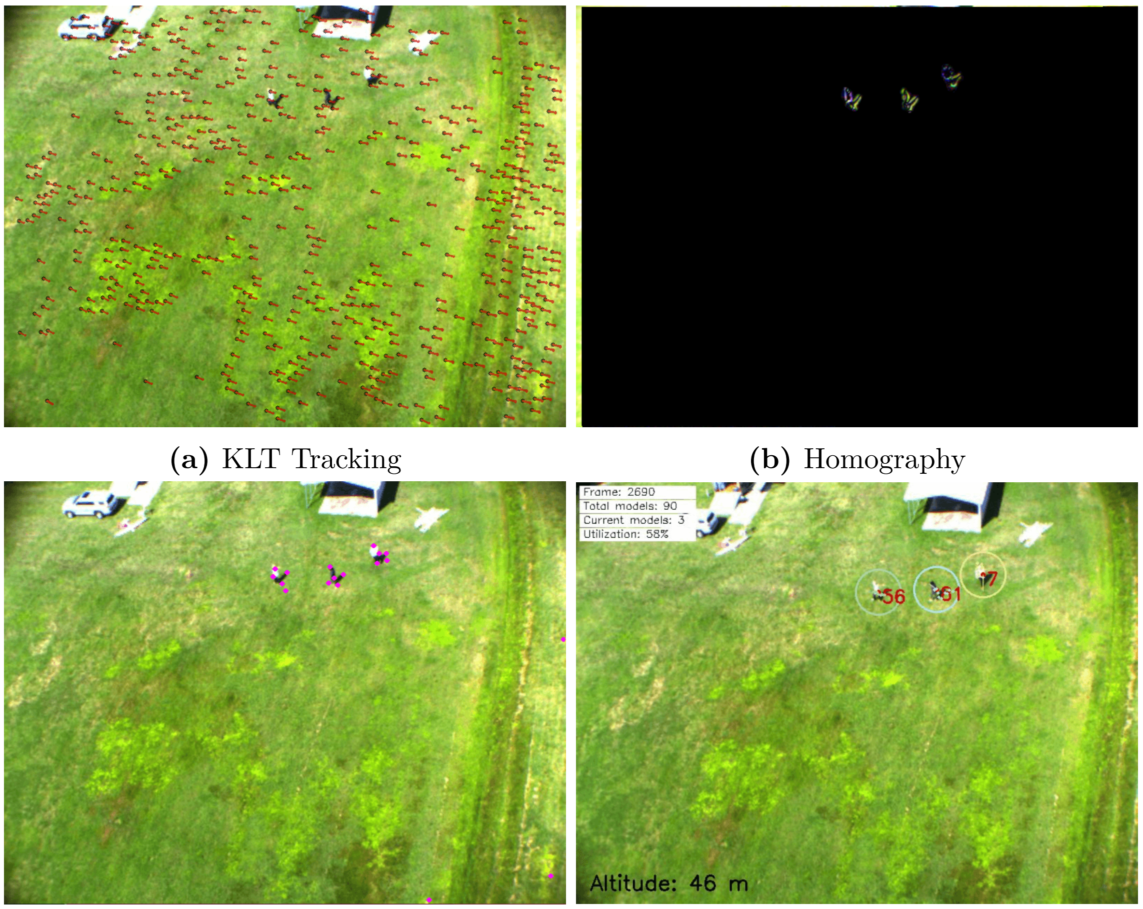
Model update: At the end of each timestep, each model uses its associated inliers to perform a Kalman update.

Model elevation: A model is elevated to a track once it has survived τ_T iterations without having more than τ_{CMD} consecutive missed detections. This step is also where models with poor support are pruned and models with similar positions are merged if within τ_x and τ_y of each other.

R-RANSAC's strength lies in its ability to initialize new hypothesis models from noisy data and subsequently manage those models without operator intervention. This allows the use of computationally cheap computer vision algorithms with less precision. Additionally, R-RANSAC is not strictly a computer vision algorithm; it can filter measurements from diverse sensor modalities [15]. The work of fusing different measurement sources in R-RANSAC is currently being investigated. For more information about the R-RANSAC derivation we refer the reader to [11].

3.2.2 Visual-Measurement Front End

In this work, R-RANSAC receives data from a visual-measurement front end. The vision processing is done with a calibrated camera in a three-step pipeline to (i) find feature corre-



(a) KLT Tracking **(b)** Homography

(c) Moving Features **(d)** Tracking Results

Figure 3.3: The three steps (a)-(c) of the visual measurement front-end with the resulting tracks (d) from R-RANSAC. Images are taken from sequence **run2a**. Feature correspondences from (a) are used to estimate a homography. Note how the homography-compensated difference image in (b) masks out the feature motion resulting from camera motion and exposes independently moving objects.

spondences between images, (ii) compute a homography, and (iii) detect true object motion. The input video rate is controlled by the *frame stride* parameter which dictates how many frames to skip. For example, with incoming video at 30 fps, $stride = 3$ results in 10 Hz processing.

(i) *Feature management*: At each timestep k, features from the last image X_{k-1} are propagated forward into the current image as X_{k-1}^{+} using optical flow. Feature correspondences (X_{k-1}, X_{k-1}^{+}) are sent as input to the next step in the pipeline for further processing. A new set of features X_k are then found using the Shi-Tomasi corner detection method for the current image $\mathcal{I}_k$. These features will be propagated on the next iteration. This step is known as Kanade-Lucas-Tomasi (KLT) tracking and is depicted in Figure 4.4a.

(ii) *Homography generation*: Using the feature correspondences (X_{k-1}, X_{k-1}^+) from the KLT tracker, a perspective transformation H known as a homography is estimated using a RANSAC-based scheme. This step is crucial for moving platform tracking because it allows the set of features X_{k-1} and X_k to be represented in the same coordinate frame through image registration. The quality of a homography estimation between camera views can be visualized via difference imaging (see Figure 4.4b), defined as

$$\mathcal{D}_k = \mathcal{I}_k - \mathcal{I}_{k-1}^+ = \mathcal{I}_k - H\mathcal{I}_{k-1}. \tag{3.1}$$

Note that the R-RANSAC visual tracker only makes use of KLT features and that the difference image $\mathcal{D}_k$ is only computed when assessing the homography estimation quality.

(iii) *Moving object detection*: Equipped with a homography and a set of feature correspondences, the velocity of each of the feature points can be calculated as

$$V = X_{k-1}^+ - HX_{k-1}. \tag{3.2}$$

If the homography estimate is good, then the velocity of static features will be nearly zero, leaving behind the motion of independent objects only, as shown in Figure 4.4c. Measurements ($z_j = [x, y, v_x, v_y]^\top$) of independently moving objects are defined as feature points that have a velocity magnitude within predefined thresholds, given by

$$Z = \{(x_i, v_i) \in X_{k-1}^+ \times V : \tau_{v_{\min}} \leq v_i \leq \tau_{v_{\max}}\}.$$

This scan of measurements is then given to R-RANSAC to estimate the position of targets. Figure 4.4d shows the tracking results.

Note that the calibration parameters of the camera are used to undistort the features extracted in step (i). Further, the camera matrix is used to project features from 2D pixel space to the *normalized image plane* in 3D space where coordinates are normalized such that the depth is unity. This results in tracker parameters that are less sensitive to differences in calibrated cameras and allows tuning to be done in more intuitive units, as described below.

3.3 Altitude-Dependent Tuning

Track continuity is an important attribute of situationally aware systems. This attribute implies that moving targets maintain a unique track ID throughout its lifetime. As the aerial vehicle changes altitude, objects will change in size with respect to the camera field of view. This can cause tracks to fragment into multiple IDs.

To maintain track continuity during a UAS descent, we propose using the vehicle altitude to tune parameters of the R-RANSAC visual tracking system. The relevant tuning parameters for R-RANSAC are the inlier region τ_R and the absolute difference threshold for model merging, τ_x and τ_y. The visual front-end feature velocity thresholds $\tau_{v_{\min}}$ and $\tau_{v_{\max}}$ are also tuned during flight. Denote the UAS altitude as h. The parameters are then

$$\tau_R = \frac{s}{2h}; \quad \tau_x = \tau_y = \frac{d_{\text{merge}}}{h} \tag{3.3}$$

$$\tau_{v_{\min}} = \frac{v_{\min}}{h}; \quad \tau_{v_{\max}} = \frac{v_{\max}}{h}, \tag{3.4}$$

where the tuning parameters are: s, the object size in meters; d_{merge}, the distance for model merging in meters; $v_{\min}$ and $v_{\max}$ the minimum and maximum target velocities in meters per second.

3.4 Hardware Demonstration

The R-RANSAC visual tracker software with altitude-dependent tuning is demonstrated in real time during four flight tests. A GPU implementation is discussed that enhances the tracking ability of the algorithm. The flight scenario is described, along with performance metrics for multiple object tracking.

3.4.1 GPU Implementation

The most computationally expensive portion of the R-RANSAC visual tracker is the KLT optical-flow feature tracking. Leveraging OpenCV libraries allowed for a GPU implementation of the visual front end. Although able to run on mobile CPU machines, using a GPU gives the visual tracker extra processing time between each frame, allowing more features to be extracted at a higher frame rate (i.e., lower *stride*). It also allows processing time for higher level functions such as mapping, localization, and path planning.

An increase of feature correspondences between frames allows for a more robust homography estimation process. A homography describes the perspective transformation of a plane from one view to another. In low-altitude scenarios there is often more depth in the scene from tall structures and trees, causing parallax and making it more difficult to find a single plane to describe the entire image. However, moving objects are most often found on local planes (e.g., streets, grass, etc). With more feature points used in the homography estimation process, there will be a higher likelihood of choosing the local plane of the targets and rejecting the parallax points as outliers. This increases the ability of the visual front end to maintain detection of independent object motion.

Using the flight test data, we perform a comparison of real-time tracking efficiency on various CPU and GPU platforms. Efficiency is expressed in terms of *utilization*, which is measured per frame as the ratio of time spent processing to total time available ($\frac{1}{\text{fps}}$).

3.4.2 Flight Scenario

The selected sUAS for testing is the 3DR Y6 multirotor with an onboard NVIDIA Jetson TX2, as shown in Figure 4.7. Four flight tests were performed to demonstrate the R-RANSAC visual tracker in real-time. Each scenario lasted 2 minutes starting at an altitude of 122 meters and ending at 20 meters. The descent was controlled by four pre-programmed waypoints resulting in a 45° descent. The multirotor stops at each waypoint for 10 seconds.

During the descent, targets move at various rates in the camera field of view. Using the Robot Operating System (ROS), all stages of the visual tracking system are recorded, including vehicle state and tracking results. ROS manages the initialization and communication of the software as soon as the flight computer is powered on.

3.4.3 Performance Metrics for Multiple Object Tracking

To measure the performance of multiple object tracking, we use the CLEAR MOT metrics, MOTP and MOTA [47]. These metrics provide an overall performance measure of tracker precision (MOTP) and accuracy (MOTA). MOTP measures the ability of the tracker to precisely estimate the true position of objects. In this work, an intersection-over-union (IOU) ratio of bounding boxes is used; thus, a MOTP score of 1.00 represents perfect precision. MOTA gives a measure of how well the tracker detects objects and their trajectories, being penalized for false positives, track mismatches, and missed detections. A given track is associated with ground truth if there is at least 10% overlap (IOU) between bounding boxes.

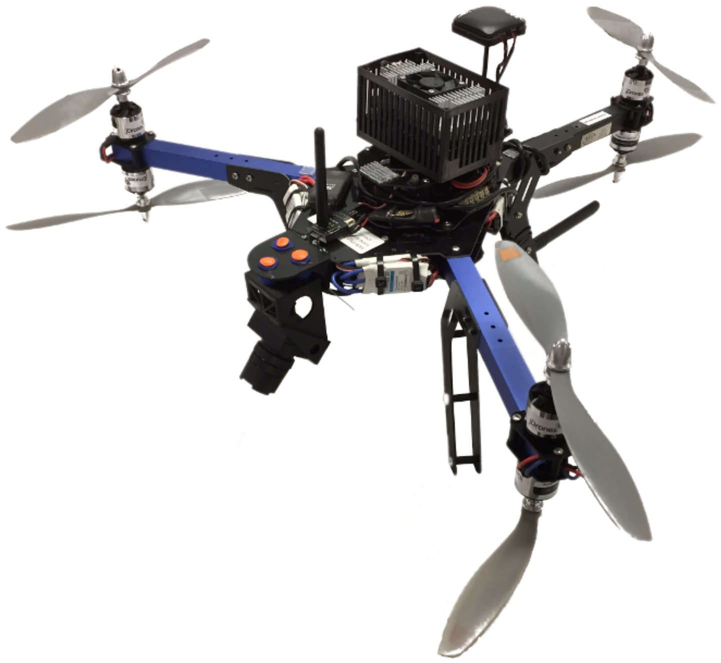


Figure 3.4: 3DR Y6 multirotor used in hardware demonstration. The Pixhawk autopilot runs APM:Copter firmware. The camera has a resolution of 800×600 at 30fps and is mounted at a $45°$ angle.

In addition to the CLEAR MOT metrics, we provide a measure of *track coverage* for each ground truth object in the flight test. It is defined as the ratio of total frames tracked to total frames present (and moving); thus, it is a means of breaking down the MOTA score to each object. False positive (FP) coverage is defined as the ratio of false positives to the number of frames in the sequence.

To calculate these metrics, ground truth is required. Flight video was manually annotated using a VATIC-inspired JavaScript implementation[2] [48].

3.5 Tracking Results

3.5.1 Tracking Performance

The results of the CLEAR MOT tracking analysis are found in Table 3.1. The arrows in the heading indicate if low (↓) or high (↑) ratios are preferred. The overall results for each test flight sequence are given as well as a breakdown according to alitude ranges dictated by the four waypoints (122 m, 84 m, 53 m, 20 m). Targets 1–3 are people and target 4 is a small RC vehicle. Runs 1 and 2 are smoother and more visually consistent, while runs 3 and 4 are more difficult because of harsh lighting, strong winds, and many strong edges in the scene. This difficulty is reflected in the MOTA and MOTP scores.

Note that scores tend to improve as the altitude decreases. This is expected because moving objects have more detail in the frame, allowing more features to be extracted and tracked. We expect that scores would improve further if we employed a feature parallax detection and rejection scheme.

For the smoother video in runs 1 and 2, the visual tracker performs very well. This is in contrast to the erratic, wind-induced camera motion in runs 3 and 4. We note that the most apparent difference in the two pairs of sequences is that runs 1 and 2 tended to have the targets in the middle of the frame while the targets were more often on the edge of the frame in runs 3 and 4. This caused the targets to frequently move in and out of the camera field of view, incurring track fragmentation and missed detection costs. We suggest that using an object detection method (feature descriptors, template matching, etc.) or tracking measurements in the 3D world coordinate frame would mitigate these issues.

[2]Source code can be found at `github.com/plusk01/vatic.js`
[3]Video results can be found at `https://youtu.be/UIlvXSdVvqA`

3.5.2 GPU vs CPU Utilization

Figure 3.5 shows the results of running the four video sequences on three different machines while holding the parameters constant. Each machine has an associated pair of bars, with the left representing CPU-only and the right GPU-enabled. The **i7Mobile** is a Gigabyte Brix computer with only a CPU. An **i7Desktop** machine is listed for comparison. Each bar breaks down how much time is spent in each part of the tracking pipline during one period ($\frac{1}{\text{fps}}$). The algorithm is running in real-time if utilization is under 100%.

Note that the **i7Mobile** computer can on average run in real-time with a frame stride of 3; however, there is almost no extra processing time for higher-level tasks. The **TX2** on the other hand performs comfortably in real-time for strides of 1, 2, and 3. We choose to process at $stride = 3$ (10 Hz) so that there is more target motion between frames at higher altitudes and so there are plenty of computational resources available for future higher-level tasks.

3.6 Conclusion

In this paper we have demonstrated a visual multiple target tracker running in real time and onboard a descending multirotor. With a robust target tracking solution, the situation awareness of autonomous vehicles is increased. A utilization analysis shows that extra processing time is available for tracking improvements and higher-level tasks such as path planning and control. Future work will include parallax compensation, world frame tracking, and trajectory optimization for ground target avoidance during landing.

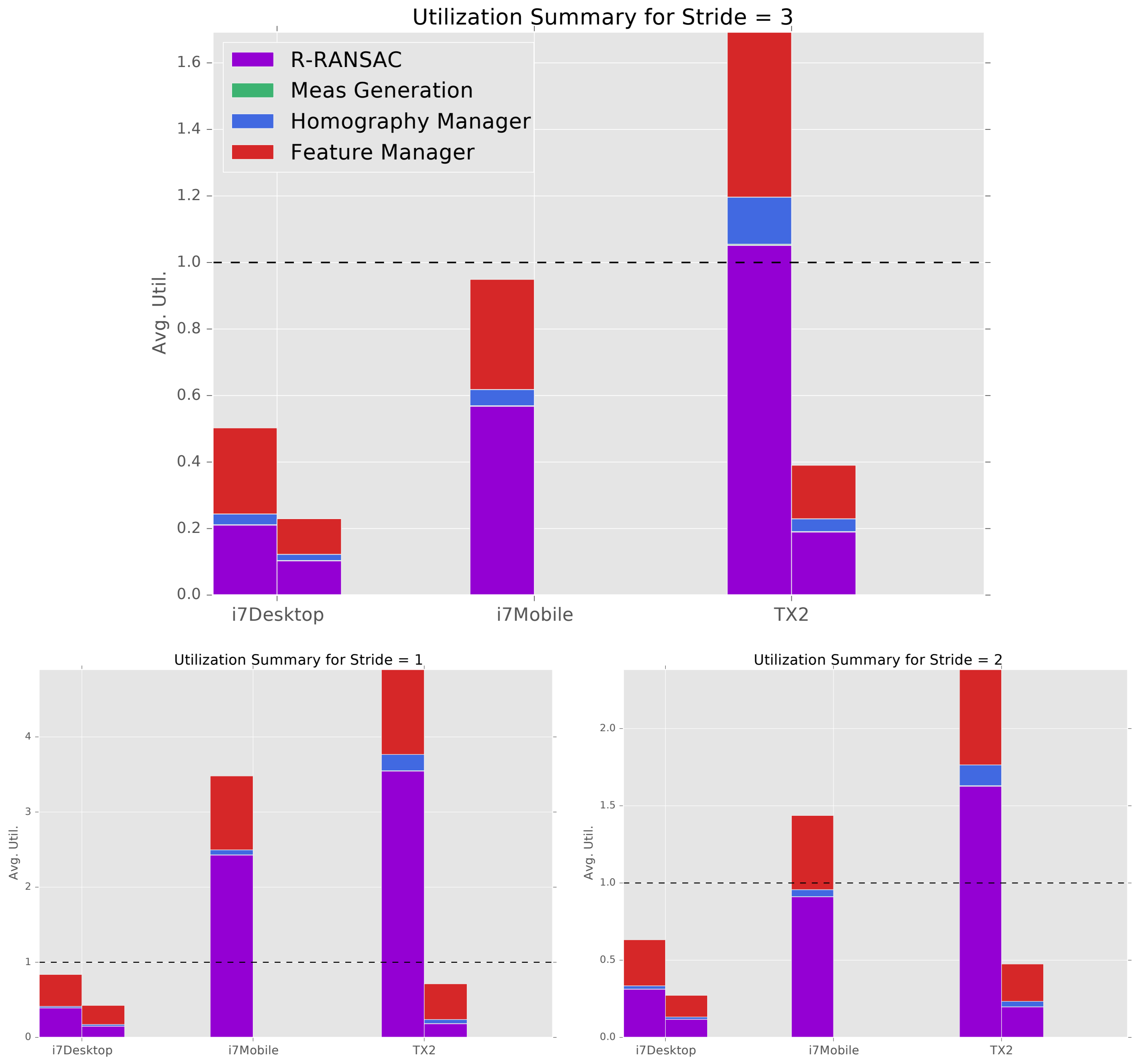


Figure 3.5: R-RANSAC visual tracker utilization for frame strides of 1, 2, and 3. Each of the three bar pairs represent a single machine. The left bar in each pair shows the utilization on CPU-only tracking and the right bar shows the utilization with the GPU enabled (if available). The tracker is running in real time if utilization is below the dotted line (100%).

Table 3.1: Tracking Performance[3]

Sequence	Duration [s]	Altitudes [m]	MOTA ↑	MOTP ↑	FP ↓	Target Coverage ↑			
						1	2	3	4
run1a	**123**	**122 to 20**	**0.75**	**0.52**	**0.28**	**1.00**	**0.94**	**0.87**	
	49	122 to 84	0.66	0.53	0.43	1.00	0.98	0.82	
	37	84 to 53	0.81	0.52	0.17	1.00	0.92	0.88	
	37	53 to 20	0.78	0.52	0.22	1.00	0.87	0.93	
run2a	**120**	**122 to 20**	**0.90**	**0.59**	**0.10**	**0.99**	**0.97**	**0.91**	
	49	122 to 84	0.91	0.52	0.13	1.00	0.98	0.97	
	36	84 to 53	0.89	0.67	0.13	1.00	0.94	0.91	
	35	53 to 20	0.90	0.60	0.05	0.99	1.00	0.83	
run3a	**123**	**122 to 25**	**0.47**	**0.39**	**0.27**	**0.93**	**0.55**		
	50	122 to 85	0.42	0.48	0.54	0.98	0.63		
	40	85 to 49	0.53	0.36	0.11	0.91	0.35		
	33	49 to 25	0.48	0.29	0.10	0.90	0.73		
run4a	**120**	**123 to 25**	**0.25**	**0.30**	**0.39**	**0.62**			**0.33**
	53	123 to 83	0.45	0.30	0.38	0.74			0.33
	38	83 to 49	0.10	0.28	0.40	0.33			0.14
	29	49 to 25	0.23	0.32	0.38	0.66			0.59

Chapter 4

Safe2Ditch: Emergency Landing for Small Unmanned Aircraft Systems[1]

4.1 Introduction

Technology and computing capabilities continue to allow small unmanned aircraft systems (sUAS) to be used in sophisticated applications such as infrastructure monitoring, medical services delivery, search and rescue, natural disaster assessment, package delivery, atmospheric observation, etc. Small UAS are well suited for these tasks because of their low cost, quick deployment time, and ability to perform hazardous work. However, to fully realize the economic and societal benefits of these civil and commercial applications, a structure must be put in place to integrate sUAS into the National Airspace System (NAS) [49].

Many efforts are currently being made in this regard, both in terms of policy and technology [50, 51, 52]. Most notably is the UAS Traffic Management (UTM) effort being led by NASA and the United States Federal Aviation Administration (FAA) [5]. UTM is a traffic management ecosystem for low-altitude (below 400 ft, 120 m) sUAS operations in uncontrolled airspace. Its purpose is to connect UAS operators with real-time data from other UAS in the vicinity, weather and terrain data, and an overall NAS regulator (e.g., the FAA). This architecture aims to maximize information flow between users, giving flexibility to operators and their sUAS operations when working in low-risk or isolated areas and providing support and structure when multiple sUAS operate in densely populated areas.

[1]This chapter is a modified version of the paper submitted to the Journal of Intelligent & Robotic Systems, written by Parker C. Lusk, Patricia C. Glaab, Louis J. Glaab, and Randal W. Beard [27]

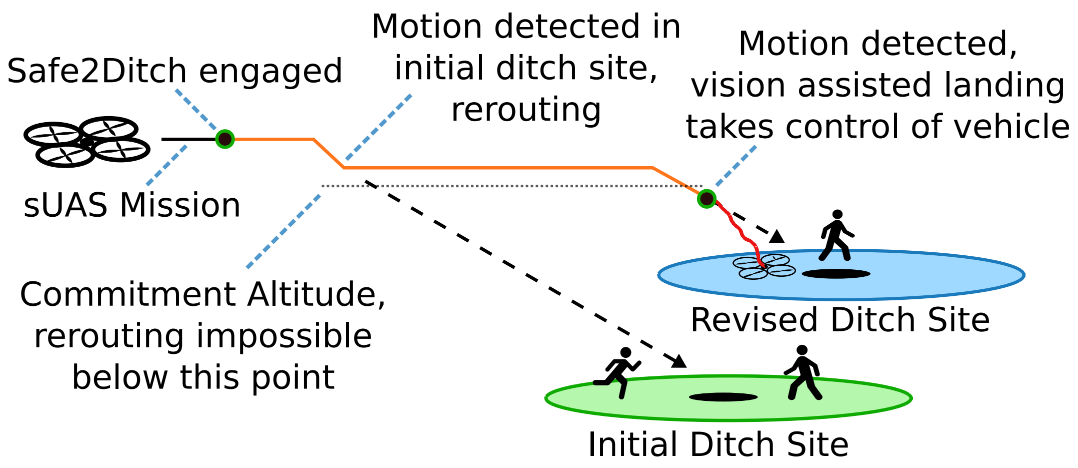

Safe2Ditch engaged
Motion detected in initial ditch site, rerouting
Motion detected, vision assisted landing takes control of vehicle
sUAS Mission
Commitment Altitude, rerouting impossible below this point
Revised Ditch Site
Initial Ditch Site

Figure 4.1: Notional flight scenario in which Safe2Ditch would be used.

Integration into the NAS as suggested by UTM requires that unmanned systems flying beyond visual line of sight (BVLOS) have much more capable on-board autonomy—not only to carry out specific missions, but to enable safety critical components such as sense and avoid, forced landing, operation in GPS degraded environments, and communication-loss mitigation. These safety concerns must be fully addressed and standardized across sUAS in the NAS. The main contribution of this paper is an emergency landing scheme for sUAS operating at low-altitudes in the NAS.

In manned aircraft, emergency landing is a crucial aspect of flight safety. When technical problems such as engine failure or structural damage occur, the pilot must quickly select a suitable landing site that might not be at a designated airstrip. Using communication with air traffic control, a safe landing zone is chosen that is clear of people and other obstacles. In the case of uncontrolled general aviation airspace, such as FAA Class G airspace, it is the pilot's responsibility to visually locate a safe landing site clear of people and other obstacles. Performing forced landing maneuvers at low-altitude for unmanned aircraft requires the on-board system to be able to perceive the environment and make decisions on a landing site autonomously. Additionally, the size, weight, and power constraints of sUAS prevents the use of double- or triple-mode redundancy as is common in manned aircraft. This results in the need of a quick reaction time for sUAS emergency landing systems.

A successful model for autonomous interaction with the environment can be found in the sensory-based decision making process known as *situation awareness* [6]. Originating in the military aircraft pilot community, situation awareness is defined as the human process of perceiving details in the environment, comprehending how those details affect the current goal, and projecting that comprehension to what will happen in the near future [7]. This process of situation awareness allows human operators to make critical decisions in a timely and effective manner. Similarly, adding a sense of situation awareness in autonomous vehicles allows them to operate effectively in dynamic environments [8]. For example, McAree and Chen [9] discuss adding artificial situation awareness to UAS by using automated dependent surveillance-broadcast (ADS-B) to cooperatively sense other agents while landing at prepared sites.

To better discuss the need for increased autonomy and situation awareness, we define the sUAS emergency landing problem in the following series of steps.

1. **Event detection.** During normal operation of an sUAS mission, an emergency event may be received from various sources. In the context of NAS integration, the UTM system may broadcast a land directive for the purposes of aircraft deconfliction or other safety reasons. Other examples of emergency events include vehicle fault detection such as low power, damaged motors, etc.

2. **Landing site selection.** During an emergency, the availability of a designated landing site is unlikely. In that case, safety systems for sUAS must be able to identify a suitable location for landing or crashing. This could include locations such as building tops, trees, lakes, parks, or roads.

3. **Path planning.** Once a new emergency landing site is selected, an efficient path to the desired location must be designed. Examples include optimizing trajectories for power consumption, time to land, or observability constraints. If a sense-and-avoid

system is present, a reactive path planning scheme may be used to safely guide the vehicle through air traffic. Additionally, a 3D obstacle map could be used to remain within UTM-cleared airspace or to plan paths around static obstacles such as buildings or trees.

4. **Robust control.** Depending on the vehicle and the nature of the emergency event, specialized autopilot routines may need to be used to maintain maneuverability of damaged vehicles. Additionally, it is the goal of the controller to ensure that paths created by the planner are followed.

5. **Tactical maneuvering.** As the sUAS makes its final approach to the impromptu landing site, there are likely to be dynamic ground obstacles such as cars, animals, and people that were not accounted for in the path planning stage. The goal of tactical maneuvering is to react to these non-cooperative ground obstacles using on-board sensing so that the vehicle can land without causing harm to the ground obstacles or itself. Depending on the population density of the chosen landing site, this step of emergency landing can be the most challenging in regards to safety and is known as the "last 50 feet" concern by UTM [5].

Safe2Ditch is a concept for an on-board crash management system that continuously monitors vehicle health and mission objectives. This paper focuses on real-time landing site selection and builds upon [27] to introduce visual situation awareness into the site selection process. While other methods discussed in Section 4.2 have designed landing site selection schemes, this paper incorporates visual tracking so that the current state of potential ditch sites can be assessed while still at higher altitudes. Simulation and hardware experiments demonstrate the ability of Safe2Ditch to quickly select the best landing site using visual tracking.

The paper is organized as follows. Section 4.2 reviews current solutions relevant to forced landing of small unmanned aircraft systems. Section 4.3 discusses each of the emergency landing steps in the context of the Safe2Ditch architecture. Section 4.4 discusses the vision processing and target tracking algorithms that were designed and implemented to inform Safe2Ditch about obstacle motion on the ground. In Section 4.5, the Safe2Ditch system is tested in simulation and in hardware. A discussion of the results are also given. We conclude with final thoughts and next steps in Section 4.6.

4.2 Related Work

4.2.1 Emergency Landing

One of the earliest fully-integrated emergency landing systems to be flown on-board an unmanned aerial vehicle was by Scherer et al. [29]. In real-time, the system demonstrated the ability to detect, select, and navigate a full-scale unmanned helicopter to an unprepared landing site. Using lidar for perception and on-board computing resources, the system is able to build an online 3D point cloud of the terrain surrounding the full-scale unmanned helicopter. This terrain model is then used to evaluate how well a 3D model of the landing gear fits at different locations. Once the contact of the landing gear to the terrain model is nearly level, the landing site is determined and the helicopter autonomous navigates and lands there.

In Mejias and Fitzgerald [30], the authors developed a visual landing site detection system using a method based on image segmentation. Applied to video data from a Cessna 172 flying at approximately 1500 ft (450 m) above ground level, the detection system is able to find large potential landing sites. Warren et al. [53] use a similar system that incorporates digital elevation models (DEM) and inertial measurement unit (IMU) data to perform dense 3D reconstruction using a structure-from-motion technique. Shen et al. [54] also developed a visual landing site detection system for manned fixed-wing flight to assist pilots in choosing an appropriate landing site, but was only tested on simulated data. Note that the aforementioned works do not explicitly handle cases of motion in the potential landing sites.

While these automated systems were successful for larger aircraft, in this paper we target small UAS with limited size, weight and power constraints. Additionally, the operational envelop that we are targeting is below 400 ft (120 m) and with only one minute to land once the emergency is triggered. Eendebak et al. [55] demonstrates a vision-based landing site selection on post-processed video acquired from a handheld camera at 100 ft (30 m). Using a background estimation filter, foreground elements are exposed and a graph-cut-based segmentation method is used to create a 2D obstacle map which allows the safest landing site to be chosen. However, the use of background estimation and subtraction requires slow, smooth flight and the sUAS must hover in place while potential landings sites are being imaged. In the work of Mackay et al. [56], landing site identification for multirotors is proposed using a plane-fitting algorithm on RGB-D point-cloud data.

Existing commercial solutions are based heavily on parachute technology, such as ParaZero [57]. These passive systems are inexpensive and effective at ensuring a safe descent, but leave the sUAS vulnerable to wind and crippled in cooperative sense-and-avoid capability. Other sUAS simply rely on the human operator to steer the drone to a clear landing zone as it slowly descends [58].

Research in other facets of the sUAS emergency landing problem have also been progressing. Di Donato and Atkins [59] proposed enhancing the site-selection process with the addition of non-traditional information such as real-time mobile phone activity to assist in avoiding densely occupied areas. In contrast, our work assumes a database of pre-selected ditch sites and focuses on refining site selection based on real-time motion. The work of Coombes et al. [60] provides a landing site reachability analysis of a fixed-wing sUAS that is gliding due to engine failure. Mueller and D'Andrea [61] demonstrate fault-tolerant control of a quadrotor with complete loss of various propellers.

4.2.2 Visual Target Tracking

Visual target tracking is an active area of computer vision research [62, 32]. Three defining characteristics of visual trackers are (1) their ability to track from a stationary versus moving camera, (2) their ability to track single versus multiple objects, and (3) detection-free tracking which must be manually initialized versus tracking-by-detection which can self-initialize tracks. In this paper, we use an on-board visual multiple object tracker that can self-initialize tracks moving independently from the vehicle's motion.

In a survey of vision-based techniques used on UAS, Kanellakis and Nikolakopoulos [63] discuss various works focusing on aerial surveillance and tracking of ground objects. Rodríguez-Canosa et al. [41] use Parallel Tracking and Mapping (PTAM) for motion estimation which is then used to create an artificial optical flow field. The difference between Lucas-Kanade optical flow and the artificial flow field exposes dynamically moving objects; however, this technique requires initialization using a marker map and may struggle to detect objects moving with similar velocity to the multirotor. Li et al. [44] successfully track other sUAS for sense-and-avoid applications in post-processed aerial footage; however, they assume targets are non-deformable which limits the types of objects that can be tracked (e.g., non-biological objects). Jiang and Cao [43] are able to track multiple objects in post-processed aerial video using detections based on background modeling, but do not use Bayesian filtering for track management. Teutsch and Krüger [45] also track multiple objects in post-processed aerial videos and demonstrate traffic surveillance and tracking from a constant altitude. Their approach is most similar to the tracking pipeline used in our work, but assumes constant altitude stable flight over structured environments, which results in smooth video input. Finally, Thomas et al. [38] demostrate how the output of a single object visual tracker can be used to control a quadrotor to follow a single moving object.

4.3 Safe2Ditch Overview

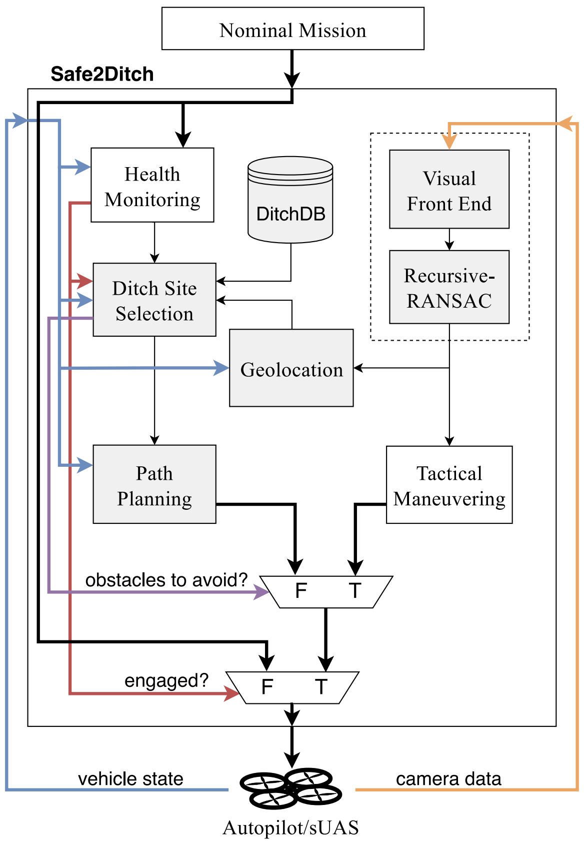
Safe2Ditch is a sUAS crash management system. Its goal is to provide emergency landing capability to either rotorcraft or fixed-wing autonomous vehicles. By communicating with the vehicle's sensors and autopilot, the set of core Safe2Ditch algorithms are able to react to emergency situations. To allow widespread access to the safety that this system provides, Safe2Ditch is designed as a flexible framework capable of interfacing with a wide variety of commercial off-the-shelf (COTS) components.

4.3.1 Architecture

To manage each of the emergency landing steps as defined in Section 4.1, Safe2Ditch has a number of subsystem components as shown in Figure 4.2. This paper addresses and demonstrates the six components shaded with light grey, while health monitoring and tactical maneuvering are left for future work. The function and interdependence of each subsystem is described below in the context of the notional emergency scenario depicted in Figure 4.1. As the system manages forced landings and crashes, the vehicle's priorities are to

1. avoid people,
2. avoid damaging property, and
3. avoid damaging itself.

We assume that once Safe2Ditch is engaged, the vehicle is only able to descend or maintain altitude.



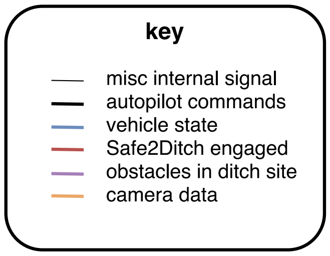


Figure 4.2: Safe2Ditch system architecture. This paper addresses and demonstrates the six light grey components. When Safe2Ditch is not engaged, the nominal mission waypoints are passed unchanged.

4.3.1.1 Health Monitoring

The Safe2Ditch system is continuously running for the duration of the sUAS mission. During normal operation, Safe2Ditch is simply monitoring vehicle health and communication channels that may relay an emergency land directive. For example, the battery or fuel required for the mission completion may be insufficient due to unexpected headwinds, a fuel leak, or poor battery health. Alternatively, the vehicle state determined by the autopilot may deviate significantly from the internal dynamic model of the health monitoring systems. If either of these or other performance anomalies occur, the health monitoring system engages Safe2Ditch to manage the emergency.

4.3.1.2 Ditch Site Selection

Once Safe2Ditch is engaged, a ditch site is selected. This step is managed by the Ditch Site Selector subsystem, which is connected to the ditch site database (DitchDB) and the visual tracking subsystem as shown in Figure 4.2. A pre-loaded database provides optional ditch site locations with key descriptors. The current position of the sUAS and the estimated time to land from the health monitoring system inform which ditch sites are within range. If no sites are in range, the vehicle immediately begins its descent and uses all of its remaining energy for tactical maneuvering. If pre-vetted sites are within range, triaging by the Ditch Site Selector (DSS) selects the optimal site based on predicted vacancy, size of the site, terrain factors, and other weighted factors. Specific values for the weight factors are a subject of ongoing research. The flight prototype places the highest value on predicted vacancy because the rotorcraft has no runway length requirement and because the test range is outfitted with many ditch site options within the vehicle's range. Simulation runs using larger real-world scenarios may indicate more emphasis on range to conserve energy for tactical maneuvering. Once the ditch site is selected, its location is passed to the path planner.

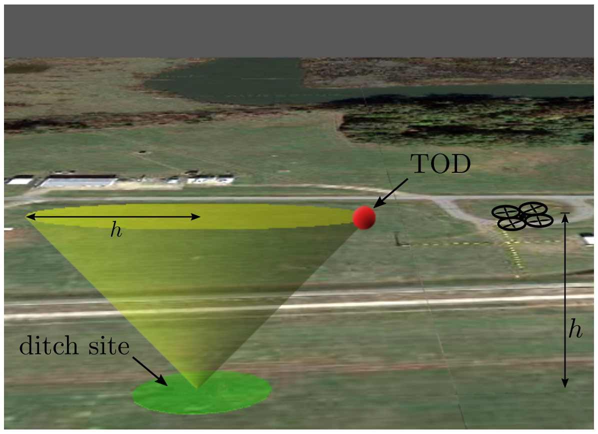


Figure 4.3: Annotated visualization of a ditch site (green). The Safe2Ditch path planner creates a TOD point (red) so that imaging of the ditch site for obstacles is maximized.

The use of a pre-loaded database has precedence in commercial aviation where airport and airspace information is pre-loaded for use by the flight management system (FMS) for the planned region for a given flight. This allows the system to have a continually updated set of information, and provides opportunity for governance. For example, future databases could provide optional ditch sites as well as avoidance areas.

4.3.1.3 Path Planning

Using the position of the selected ditch site, the path planner generates an efficient path for landing. An efficient path minimizes the landing time and maximizes the amount of time that the ditch site can be imaged by the on-board camera. By increasing the amount of time that the camera can image the ditch site, the visual tracking subsystem can better inform the Ditch Site Selector of any motion in potential ditch sites. Therefore, for a multirotor, an efficient path can be achieved by spending the majority of the landing time descending at an angle equal to the camera mounting angle. As discussed in Section 4.5, the vehicle used for flight tests has a camera that is mounted at a 45-degree angle toward the ground.

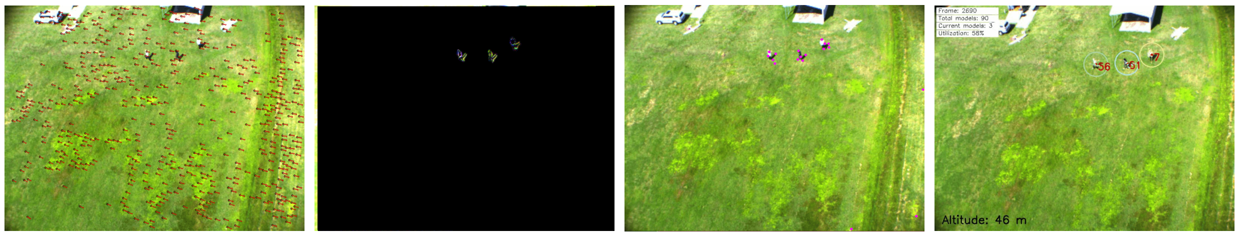
To maximize the time descending at the optimum imaging angle, an intermediate waypoint is inserted between the vehicle's current position and the ditch site. This waypoint becomes the top-of-descent (TOD) point, as shown in Figure 4.3. For the 45-degree camera mount, this is conveniently located at a distance equal to the vehicle's current altitude.

4.3.1.4 Tactical Maneuvering

As the vehicle descends toward the selected ditch site, newly detected obstacles in the landing area may cause Safe2Ditch to consider another ditch site as better suited for landing. However, at some altitude, the sUAS will not have enough energy to reroute to other ditch sites and must commit to land. This commitment altitude is calculated by comparing the remaining energy of the sUAS with the distance needed to reach other ditch sites. If the sUAS does not enough power to reach another ditch site, the better-suited ditch site is disregarded and the sUAS continues its path to the original location. In these cases, Safe2Ditch is forced to choose a suboptimal ditch site that contains moving ground objects. To maintain safety and prevent harm to the moving ground obstacles, Safe2Ditch alters its 45-degree straight-line path and begins to use a control scheme that tactically maneuvers around moving ground obstacles in the ditch site. This vision-based control scheme is left as future work.

4.3.1.5 Visual Tracking Subsystem

The visual tracking subsystem is made up of the *visual front end* and the *Recursive-RANSAC* blocks shown in Figure 4.2. Using only the vehicle's on-board camera, the visual tracking subsystem can estimate the position and higher-order dynamics of moving ground obstacles, as well as provide awareness of non-planar ground objects such as fences or trees. This image-based tracking information is then geolocated and used to inform the Ditch Site Selector of motion in the ditch site. Additionally, the image-based tracking information could be used for vision-based control techniques to perform tactical maneuvering. The visual tracking and geolocation components are discussed in more detail in Section 4.4.



(a) KLT Tracking **(b)** Homography **(c)** Moving Features **(d)** Tracking Results

Figure 4.4: The three steps (a)-(c) of the visual measurement front end with the resulting tracks (d) from R-RANSAC. Images are taken from video processed on-board a descending multirotor. Feature correspondences from (a) are used to estimate a homography. Note how the homography-compensated difference image in (b) masks out the feature motion resulting from camera motion and exposes independently moving objects.

4.4 Visual Multiple Target Tracking

To enable autonomy in dynamic environments, a visual tracking subsystem is integrated into the Safe2Ditch architecture. This visual target tracking component plays two roles: (1) to inform the Ditch Site Selector (see Section 4.3.1.2) as it is scoring ditch sites, and (2) for tactical avoidance during the final landing maneuver. The work presented in this paper builds on our previous work of designing a vision-based multiple target tracking system for use on a descending platform [27].

The visual target tracking subsystem allows tracking of multiple moving targets from the on-board camera of the sUAS. Combining a visual measurement source with an online estimation back end known as Recursive-RANSAC, a robust tracking filter is created that requires no operator intervention to initialize or manage target tracking. Target tracks produced by Recursive-RANSAC are then geolocated and projected onto a flat-earth model to estimate the 3D position of the visually tracked targets.

4.4.1 Camera Geometry

To aid in downstream tasks, the measurement processing of the visual tracker is performed using normalized image coordinates as opposed to pixels. Consider the geometry of a pinhole

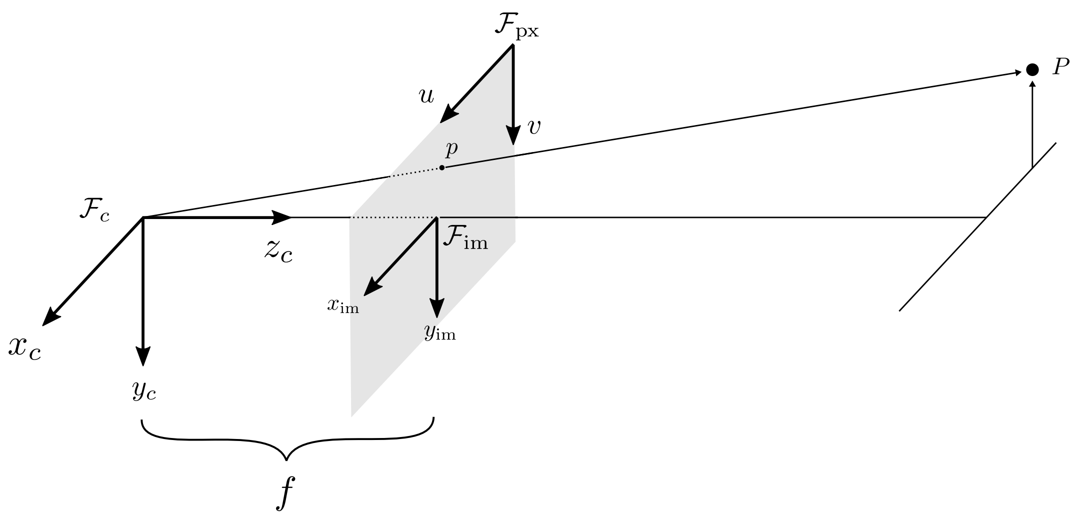


Figure 4.5: Geometry of a pinhole camera with both the pixel plane $\mathcal{F}_{px}$ and the normalized image plane $\mathcal{F}_{im}$ shown. The normalized image plane is found where the focal length $f = 1$ and the origin is aligned with the principal point.

camera model, as shown in Figure 4.5. For simplicity of the following discussion, the camera frame $\mathcal{F}_c$ is identified with the inertial frame $\mathcal{F}_i$. Suppose that the point P exists in 3D space and can be expressed in the inertial frame as $P^i = \begin{bmatrix} x^i & y^i & z^i \end{bmatrix}^\top$. The perspective projection equations for imaging the point P are given by

$$u = f x^{im} + c_x = f \frac{x^i}{z^i} + c_x \tag{4.1}$$

$$v = f y^{im} + c_y = f \frac{y^i}{z^i} + c_y, \tag{4.2}$$

where u and v are respectively the x and y pixels, f is the focal length in pixels, c_x, c_y are the pixel offsets to the camera's principal point, and x^{im}, y^{im} are the coordinates of P expressed on the normalized image plane. Note that the focal length and the principal point offsets may be found via camera calibration. In homogeneous coordinates, the normalized image coordinates can be written as the 3-vector

$$p^{im} = \begin{bmatrix} x^{im} & y^{im} & 1 \end{bmatrix}^\top . \tag{4.3}$$

Therefore, each u, v pixel received from the camera sensor can be transformed into the normalized image plane using equations (4.1) and (4.2).

The use of normalized image coordinates enables parameter tuning to be generalized to different visual systems and different scenarios, using the camera calibration of each camera to calculate the normalized image coordinates. In particular, this allows altitude-dependent tuning of the visual tracking algorithm as presented in [27]. Additionally, unit bearing vectors can be constructed from the camera data by normalizing p^{im}. These bearing vectors are used in this work for target geolocation, as discussed in Section 4.4.4.

4.4.2 Visual Measurement Front End

The purpose of the visual measurement front end is to process incoming video data and generate measurements that can be fed to the Recursive-RANSAC Tracker. Because of the robustness of the Recursive-RANSAC algorithm, we value many low-quality measurements over few high-quality measurements. Motivated by this heuristic, the vision processing is performed with a calibrated camera in a three-step pipeline to (i) find feature correspondences between images, (ii) compute a homography, and (iii) detect true object motion. Each of these steps are briefly described below. More information can be found in [27].

(i) *Feature management*: At each timestep k, features from the last image X_{k-1} are propagated forward into the current image as X_{k-1}^{+} using optical flow. Feature correspondences (X_{k-1}, X_{k-1}^{+}) are sent as input to the next step in the pipeline for further processing. A new set of features X_k are then found using the Shi-Tomasi corner detection method for the current image $\mathcal{I}_k$. These features will be propagated in the next iteration. This step is known as Kanade-Lucas-Tomasi (KLT) tracking and is depicted in Figure 4.4a.

(ii) *Homography generation*: Using the feature correspondences (X_{k-1}, X_{k-1}^{+}) from the KLT tracker, a perspective transformation H known as a homography is estimated. This step is crucial for tracking on-board a sUAS because it allows the set of features X_{k-1} and X_k to be represented in the same coordinate frame through image registration. The quality of a homography estimation between camera views can be visualized via difference imaging, as shown in Figure 4.4b. Note that the visual tracking subsystem only makes use of KLT features and that the difference image is only computed when assessing the homography estimation quality.

(iii) *Moving object detection*: Equipped with a homography and a set of feature correspondences, the velocity of each of the feature points can be calculated as

$$V = X_{k-1}^{+} - HX_{k-1}.$$

If the homography estimate explains the motion of static features in the image plane well, then the velocity of static features will be nearly zero, leaving behind the motion of independent objects only, as shown in Figure 4.4c. Measurements ($z_j = [x, y, v_x, v_y]_j^\top \in Z_j$) of independently moving objects are defined as feature points that have a velocity magnitude within predefined thresholds, given by

$$Z_{k-1} = \{(x_i, v_i) \in X_{k-1}^{+} \times V : \tau_{v_{\min}} \leq v_i \leq \tau_{v_{\max}}\}.$$

This scan of measurements is then used by Recursive-RANSAC to estimate the position, velocity, and acceleration, and jerk of targets.

4.4.3 The Recursive-RANSAC Tracker

Recursive-RANSAC is an online estimation algorithm capable of tracking an arbitrary number of objects in clutter [11, 46]. Measurements are received in the surveillance region $\mathcal{R}$ of the system, where $\mathcal{R} \subset \mathbb{R}^2$ in this work. Recursive-RANSAC uses the random sample consensus (RANSAC) algorithm to quickly initialize hypothesis models that best fit the

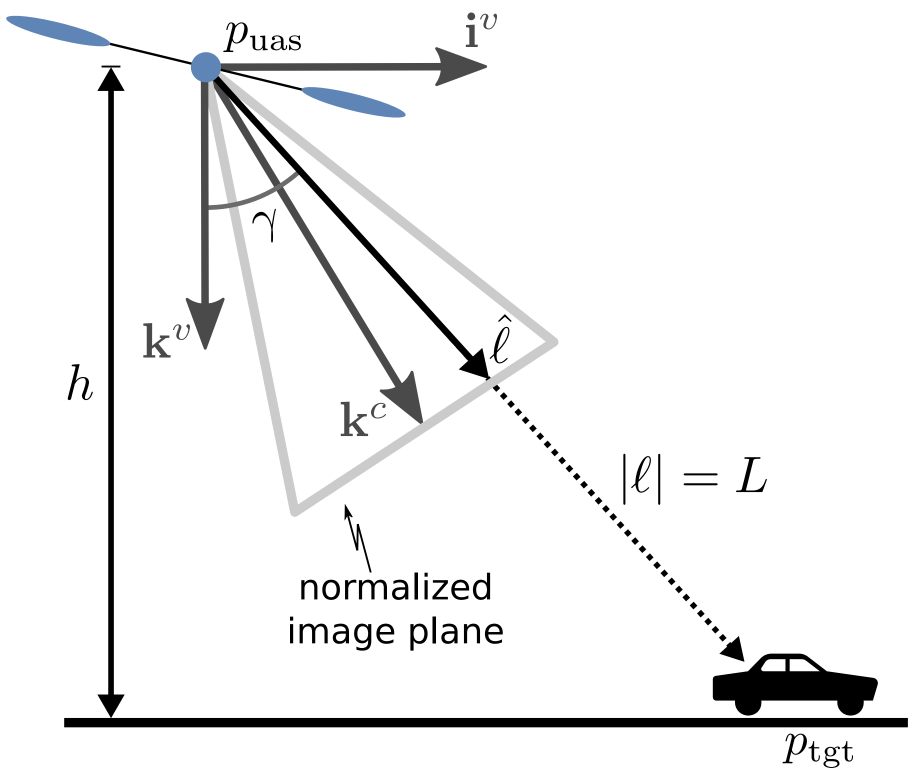


Figure 4.6: North-Down view of the geolocation problem from a multirotor. We would like to find an estimate of p_{tgt} expressed in the inertial frame. Given p^i_{uas}, the camera's attitude and a normalized line-of-sight vector $\hat{\ell}^c$ from the camera, p^i_{tgt} can be calculated.

current measurements in a maximum-likelihood sense. Once a model is initialized, a bank of Kalman filters is used to propagate each model forward based on nearly-constant jerk kinematics and a data association step as new measurements are received. The tracks found by Recursive-RANSAC can be seen in Figure 4.4d.

The benefits of using Recursive-RANSAC are found in its ease of implementation, lightweight computational requirements, robustness in rejecting outliers, and its ability to initialize and manage models without an operator [12].

4.4.4 Target Geolocation

Once Recursive-RANSAC produces target tracks, the position p_{uas} of the sUAS is used to estimate their inertial position. Consider the target p_{tgt} shown in Figure 4.6. Using the geometry of the camera and a flat-earth model of the inertial frame, the range to the target can be estimated [64].

Given a target track p^{im} expressed in the normalized image plane as in equation (4.3), a normalized line-of-sight (LOS) vector expressed in the camera frame can be constructed as $\hat{\ell}^c = \frac{p^{im}}{|p^{im}|}$. Using the unit LOS vector $\hat{\ell}^c$ and the height above ground level h of the sUAS, the objective is to estimate the inertial position of the target assuming a level ground plane. The inertial position of the target can be written as

$$p^i_{\text{tgt}} = p^i_{\text{uas}} + L\hat{\ell}^i, \tag{4.4}$$

where L is the range from the sUAS to the ground target. Because cameras are bearing-only sensors, the range must be estimated. First, we construct a rotation matrix that allows us to express all of the quantities in the same basis. This rotation matrix is composed of the rotation from the camera $\mathcal{F}_c$ to the body $\mathcal{F}_b$ and the rotation of the body to the inertial frame $\mathcal{F}_i$. Therefore, the LOS vector in the inertial frame is

$$\hat{\ell}^i = R^i_b R^b_c \hat{\ell}^c, \tag{4.5}$$

with $R^i_b, R^b_c \in \text{SO}(3)$. The range L can be calculated using the cosine of γ, the angle between the inertial $\mathbf{k}^i$-axis and the unit LOS vector $\hat{\ell}^i$. Using the following equality

$$\hat{\ell}^i \cdot \mathbf{k}^i = \cos\gamma = \frac{h}{L}, \tag{4.6}$$

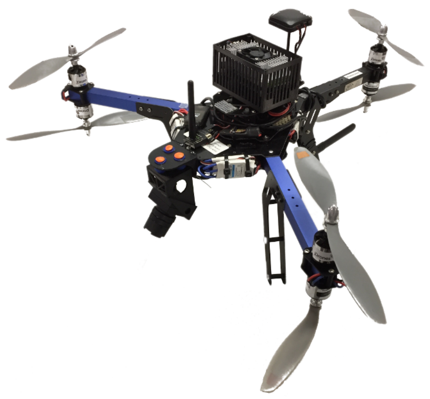


Figure 4.7: The 3DR Y6 multirotor used in hardware experiments. The Pixhawk autopilot runs APM:Copter firmware. The camera has a resolution of 800×600 at 30 fps and is mounted at a 45-degree angle.

the range can be written

$$L = \frac{h}{\hat{\ell}^i \cdot \mathbf{k}^i}. \tag{4.7}$$

Thus, the inertial position of the target at time t is given by combining equations (4.4), (4.5) and (4.7). The inertial position of moving ground obstacles can then be used to inform the Ditch Site Selector during an emergency landing, as discussed in Section 4.3.1.2.

Table 4.1: Hardware details

Component	Description
Platform	3DR Y6
Autopilot	3DR Pixhawk with APM v3.5.4
Sensors	GPS, IMU, barometer, magnetometer
RGB Camera	800×600 @ 30 fps, HFOV $\approx 34°$
Processor	NVIDIA Jetson TX2

4.5 Experimentation

To demonstrate the obstacle-aware Ditch Site Selection component of the Safe2Ditch system, simulation and hardware experiments were performed. The multirotor used for hardware tests is shown in Figure 4.7 with details in Table 4.1.

4.5.1 Implementation

The Safe2Ditch system is implemented in C++ and Python for real-time execution onboard an NVIDIA Jetson TX2 embedded computer. Using the Robot Operating System (ROS) [26] for message passing, each component can be designed, implemented, and tested independently of each other. This allows for rapid experimentation and hardware validation of the Safe2Ditch concept.

In addition to other standard software engineering concepts used in real-time applications, the visual processing stage is implemented using CUDA routines to exploit the parallelism inherit in image processing. The NVIDIA Jetson TX2 contain 256 low-power CUDA cores that enables the use of this type of processing.

The Safe2Ditch system is able to communicate with the Pixhawk autopilot that is used to control the multirotor and to estimate its current pose and velocity. This allows the geolocation algorithm (see Section 4.4.4) and the Ditch Site Selector (see Section 4.3.1.2) to access the needed inertial information about the vehicle.

4.5.2 Software-in-the-Loop Simulation

To perform successful hardware flight tests, it is important to simulate the integrated system as best as possible. Using the ROS ecosystem with the Gazebo physics simulator, it is possible to create a software-in-the-loop (SIL) simulation by executing the Pixhawk/APM software on a desktop computer. This allows the same Safe2Ditch software that would run on

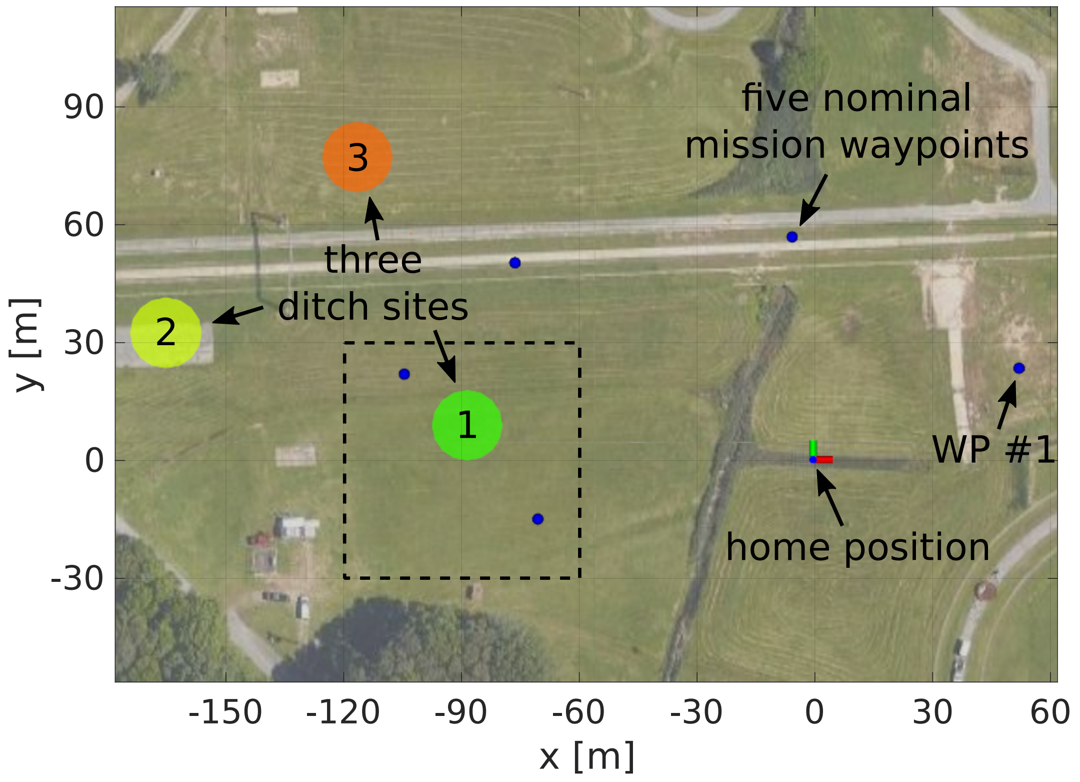


Figure 4.8: Simulated environment at the NASA Langley Research Center, expressed in an ENU coordinate frame. The five nominal mission waypoints are shown as blue dots, with the first waypoint near (50 m, 20 m). The sUAS takes off from the home position and begins to follow the waypoints in a counter-clockwise fashion. At some random time t_{engage}, Safe2Ditch is engaged and attempts to select ditch site 1, which has the highest priority in the DitchDB. Safe2Ditch must manage landing safely and quickly in the presence of N_t randomly moving targets in the area bounded by dashed lines. If a target is detected in ditch site 1, Safe2Ditch reroutes to the next best ditch site, shown in yellow.

the flight computer to be run in simulation. Using satellite imagery, the SIL is additionally able to produce synthetic images of flight test locations with simulated people walking in various ditch sites. Within the SIL environment, end-to-end emergency landing scenarios are tested that exercise the Safe2Ditch components and increase confidence in effective flight demonstrations.

The geographical location used in SIL simulations is the sUAS Testing Site at the NASA Langley Research Center. The nominal mission starts with the sUAS taking off at the home position and consists of flying counter-clockwise through five waypoints at an altitude of 60 meters, as shown in Figure 4.8. Three prioritized ditch sites are loaded into the DitchDB and are also shown in Figure 4.8, with ditch site 1 being the first choice. Using this test environment, a series of Monte Carlo simulations were run parameterized by the number of moving targets, $N_t \in \{1, 2, \ldots, 10\}$. The i^{th} target is modeled as a simple unicycle with kinematics

$$\begin{aligned}\dot{x}_i &= v_i \cos(\theta_i)\\ \dot{y}_i &= v_i \sin(\theta_i)\\ \dot{\theta}_i &= \omega_i\end{aligned}$$

where $v_i(t)$ is linear velocity and $\omega_i(t)$ is angular velocity. A waypoint-based path planner is used to command each target to a sequence of desired positions (x_i^d, y_i^d) using $\omega_i(t)$ as the control and letting $v_i(t) \equiv v_i(0)$.

For each $N_t \in \{1, 2, \ldots, 10\}$, $M = 200$ trials were run to test the sensitivity of Safe2Ditch with respect to the number of moving obstacles near the ditch site, N_t, the initial position $(x_i(0), y_i(0))$ and initial velocity $v_i(0)$ of a target, and the time that the Safe2Ditch system receives an emergency event, t_{engage}. The initial position of the i^{th} target is sampled from a uniform distribution bounded by the area around the first ditch site as shown in Figure 4.8.

Subsequent waypoints are sampled from this same distribution, $(x_i^d, y_i^d) \sim \mathcal{U}(-120, -60) \times \mathcal{U}(-30, 30)$. The velocity with which the target moves is chosen as $v_i(0) \sim \mathcal{U}(0.5, 2.5)$ and $t_{\text{engage}} \sim \mathcal{U}(30, 70)$.

4.5.3 Monte Carlo Simulation Results

The following three metrics are used to evaluate simulation results:

1. **Time to action**, t_{action}. This metric captures how many seconds it took the system to detect an obstacle in the currently selected ditch site and then reroute to the next best ditch site. Because the multirotor descends with a constant velocity of 2 m/s in both the lateral and down directions, t_{action} best describes the responsiveness of the system. Responsiveness is critical for emergency landing systems because each passing second can cause greater loss in controllability, energy, or altitude.

2. **Rate of failure**. A failure occurs if the multirotor lands in a ditch site with one or more targets present. For the purposes of simulation, we consider landing to occur once the multirotor descends to 5 meters. Additionally, we assume that agents would not knowingly walk under a landing multirotor and in our results we ignore any agent that is in the ditch site but not in the field of view of the camera. In the future, a camera with a wider field of view or complementary sensor modalities could be used to increase the detection region of Safe2Ditch.

3. **Rate of false reroutes**. As the sUAS descends toward the ditch site, it is possible that the visual tracking system outputs false positives that cause the Ditch Site Selector to unnecessarily reroute to another ditch site. Additionally, estimation error in the geolocation algorithm could cause Safe2Ditch to misclassify tracks as being inside the ditch site when the true target position is not.

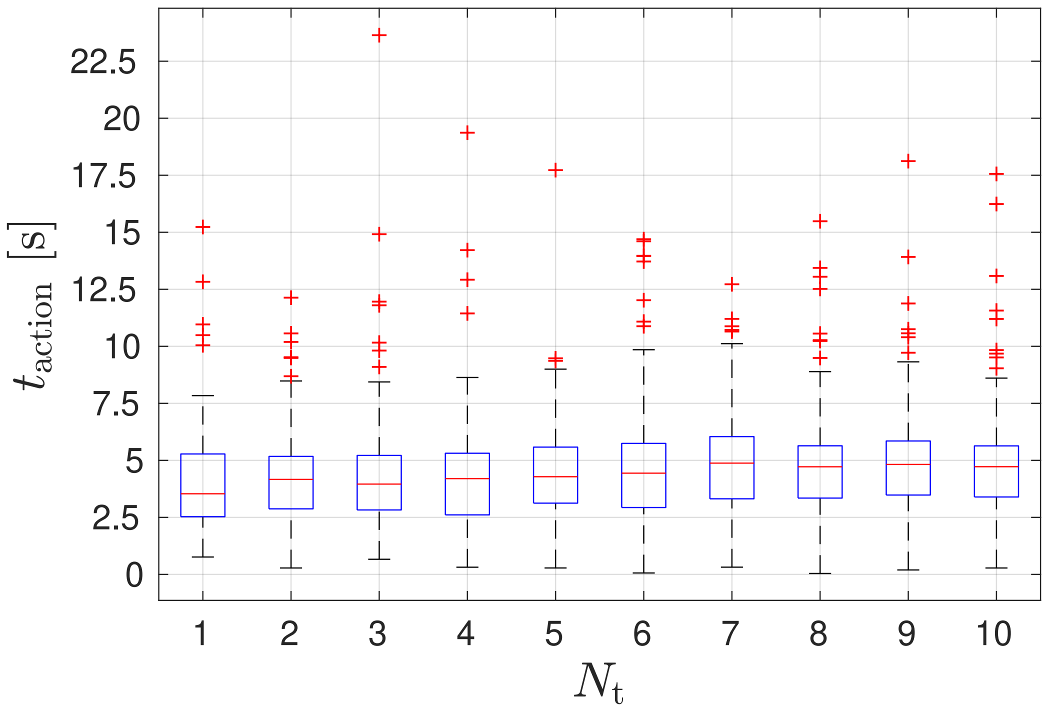


Figure 4.9: Distribution of t_{action}. The inclusion of visual data allows Safe2Ditch to quickly respond to non-cooperative obstacles. Once a moving obstacle is in the camera's field of view and in the ditch site, the median amount of time it takes to reroute to another ditch site is less than 5 seconds.

These metrics are shown in Figures 4.9–4.11. From Figures 4.9 and 4.10, we note that the performance of Safe2Ditch is robust to a varying number of obstacles moving in and near the ditch site. As expected, the higher the number of moving objects in and near a ditch site, the more likely a reroute will occur (see Figure 4.11).

4.5.4 Hardware Flight Test

Sixteen flight demonstrations are performed to exercise and validate the vision-aided Ditch Site Selection component of the Safe2Ditch emergency landing system. Three of the flights are performed near Provo, UT with a mission altitude of 100 ft (30 m) and ditch site radius of 13 ft (4 m). All other flights are performed at the NASA Langley sUAS Testing Site, with six tests operating at 200 ft (60 m) and seven tests operating at 400 ft (120 m). The ditch site radius of the tests performed at NASA Langley is 30 ft (9 m). Of the sixteen

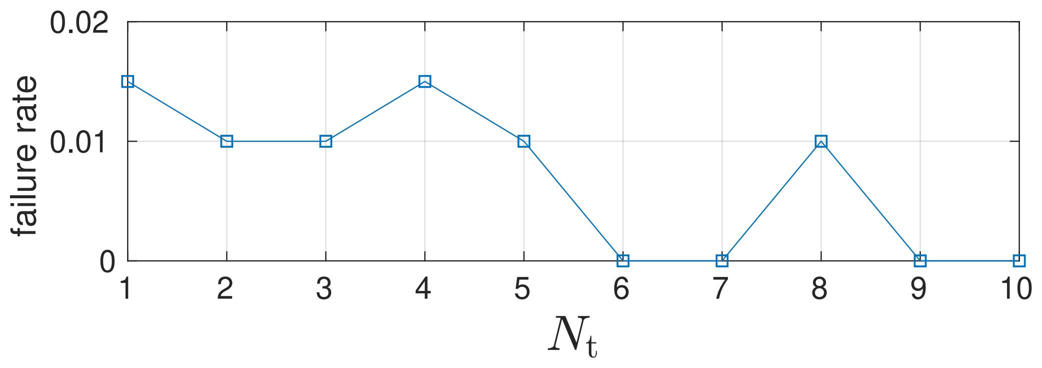


Figure 4.10: The rate of failed emergency landings given N_t targets.

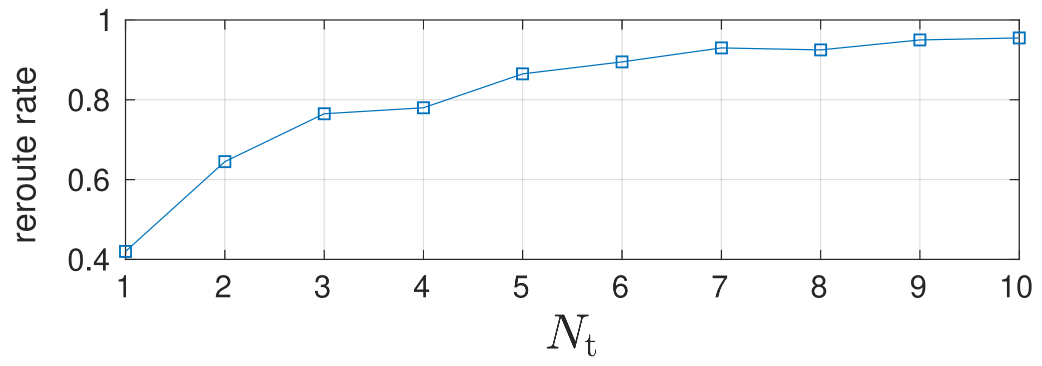


Figure 4.11: The rate at which Safe2Ditch rerouted to another ditch site because of visually perceived motion in the original site. In all of the trials there no false reroutes occurred.

flights, nine are performed with motion induced in or around the ditch site by either an operator-driven RC car or a person wearing personal protective equipment. The other seven flights have no motion induced in the ditch site to test for false positives.

Each flight scenario starts with a safety pilot controlling the takeoff phase and then transferring control to the autopilot to fly a predetermined mission with a number of waypoints. Each flight test has slight variations in its predetermined mission, with altitudes differing as noted above. At some time t_{engage} during the execution of each mission, an emergency event is generated and the Safe2Ditch system disrupts the mission operation and begins to safely land the vehicle. Throughout the duration of the test flights where motion is induced, targets are commanded to move with arbitrary trajectories and speeds inside of the ditch site.

4.5.5 Hardware Results

In addition to the metrics used for simulation results (see Section 4.5.3), we analyze the hardware data using the following metrics:

1. **Time to track**, t_{track}. This measurements captures how quickly the visual system detects and tracks moving obstacles. The difference between t_{action} and t_{track} measures how long it took the Ditch Site Selector to consider the track as actionable and reroute to another ditch site.

2. **Altitude of reroute**, h_{reroute}. For hardware tests where a reroute is expected, h_{reroute} gives the altitude that the reroute occurred.

We present the data in Figure 4.12 from the 16 flight tests organized into three sets, broken up by the nominal mission altitude: (i) 3 flights at 30 m, (ii) 6 flights at 60 m, and (iii) 7 flights at 120 m.

As shown by t_{track} in Figure 4.12, the visual multiple target tracker can quickly detect and track targets at 120 meters and below. However, we can see from t_{action} that the Safe2Ditch system took more time to react to moving ground obstacles for the 120 meter set. The greatest contributor to this latency in response time is the track geolocation algorithm (see Section 4.4.4), which often produced estimates with an error between 5 and 10 meters. Because of this error, ground obstacles near the edge of the ditch site would often be misclassified as being inside or outside. This misclassification caused a single false positive during a flight from the 30 m test set. Six of the sixteen tests landed in the first selected ditch site, with no failures.

We show the trajectories of two of the 120 meter mission altitude tests flown at NASA Langley in Figures 4.13 and 4.14. Both tests start on the ground at the origin of a local ENU inertial coordinate frame. The trajectory of the multirotor is shown in black, as it begins to travel counter-clockwise about the nominal mission waypoints. When Safe2Ditch is engaged, the color of the plotted trajectory matches one of the two ditch sites available to be selected. In Figure 4.13, there is no motion in the primary ditch site and the vehicle successfuly lands. In Figure 4.14, an agent moves around the primary ditch site, which is detected and plotted as red. Once the track of the moving agent becomes actionable, Safe2Ditch reroutes to the secondary ditch site, and the color of the plotted trajectory changes.

The red plotted points in both Figures 4.13 and 4.14 represent all tracks created by the visual tracker, throughout all time. Note that in both figures, the visual tracker detects and tracks false positives at various locations of the mission. These false positives are a result of the parallax of caused by non-planar objects in the scene (such as trees, poles, and buildings) and not the Recursive-RANSAC algorithm. Parallax is an effect where positions of objects in a 3D scene differ when viewed from two different angles. Because cameras do not convey depth information, the planar homography transformation that is discussed in Section 4.4.2 cannot compensate for non-planar motion. While this shows a weakness in the ability of the

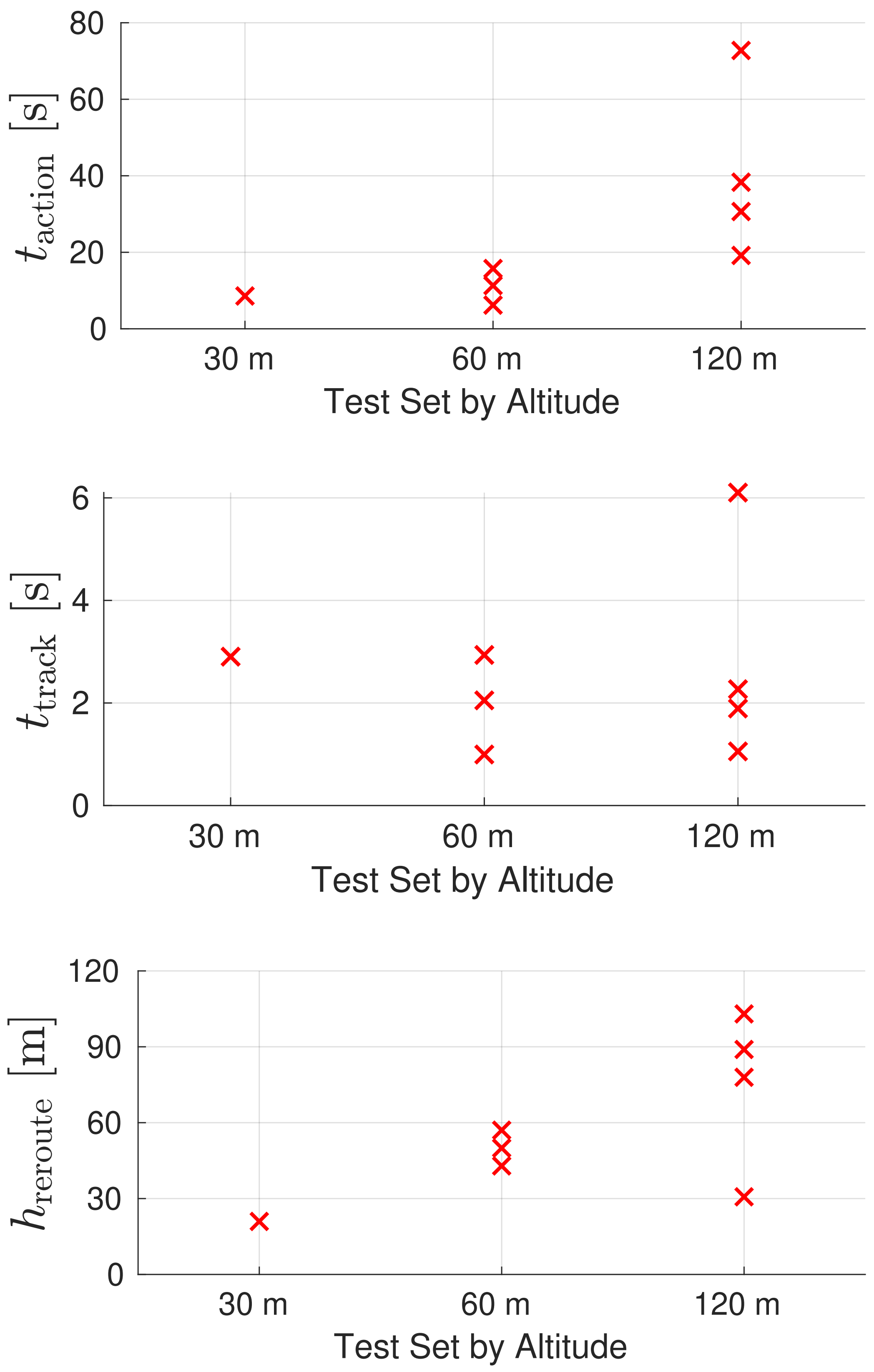


Figure 4.12: Results for 16 hardware flight tests broken up into test sets by nominal mission altitude.

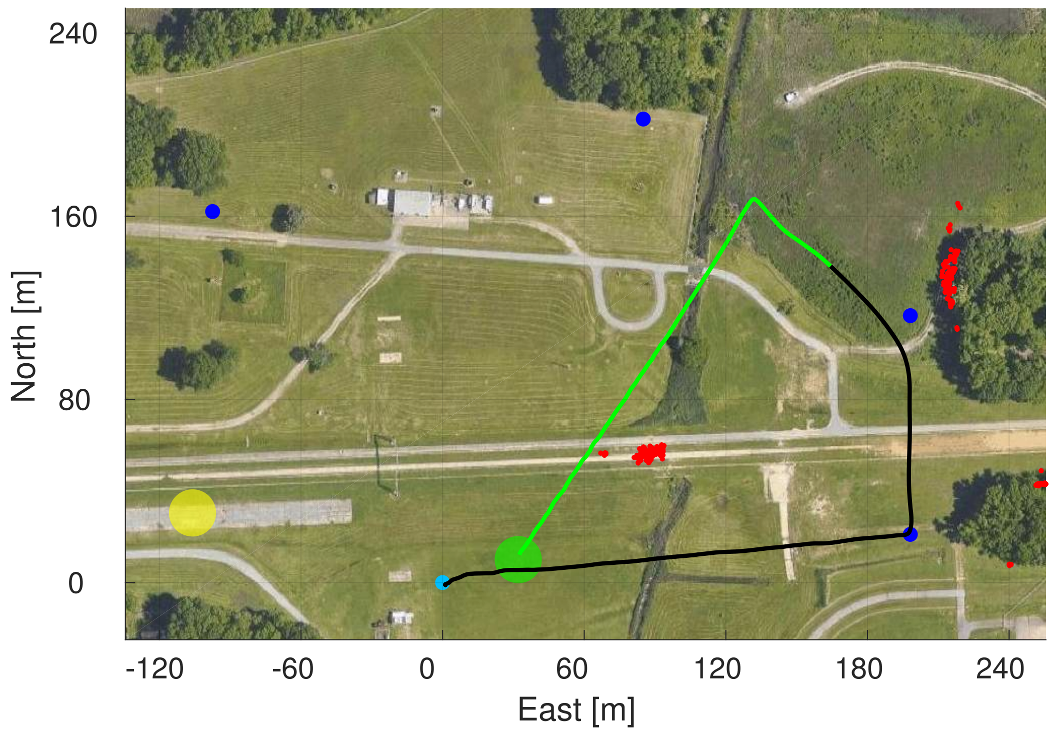


Figure 4.13: Trajectory of flight test #12. No obstacles are present in the primary (green) ditch site. Once Safe2Ditch is engaged, the vehicle successfully lands in the ditch site.

visual tracker to differentiate between moving objects and 3D structure, it does not hinder the objective of Safe2Ditch, which is to avoid obstacles as the vehicle lands. Obstacles in the ditch site that are detected based on motion or parallax will cause the vehicle to reroute to a safer area.

4.6 Conclusion

In this paper, we presented a crash management system for sUAS. This system was verified in 2000 Monte Carlo simulations with a constant nominal mission altitude and a varying number of targets. The simulation results were then supplemented with 16 hardware flight tests with either zero or one targets and varying the nominal mission altitude.

Using a pre-compiled database of potential ditch sites along the route of the nominal mission gives the sUAS potential landing options in the event of an emergency. The contribution

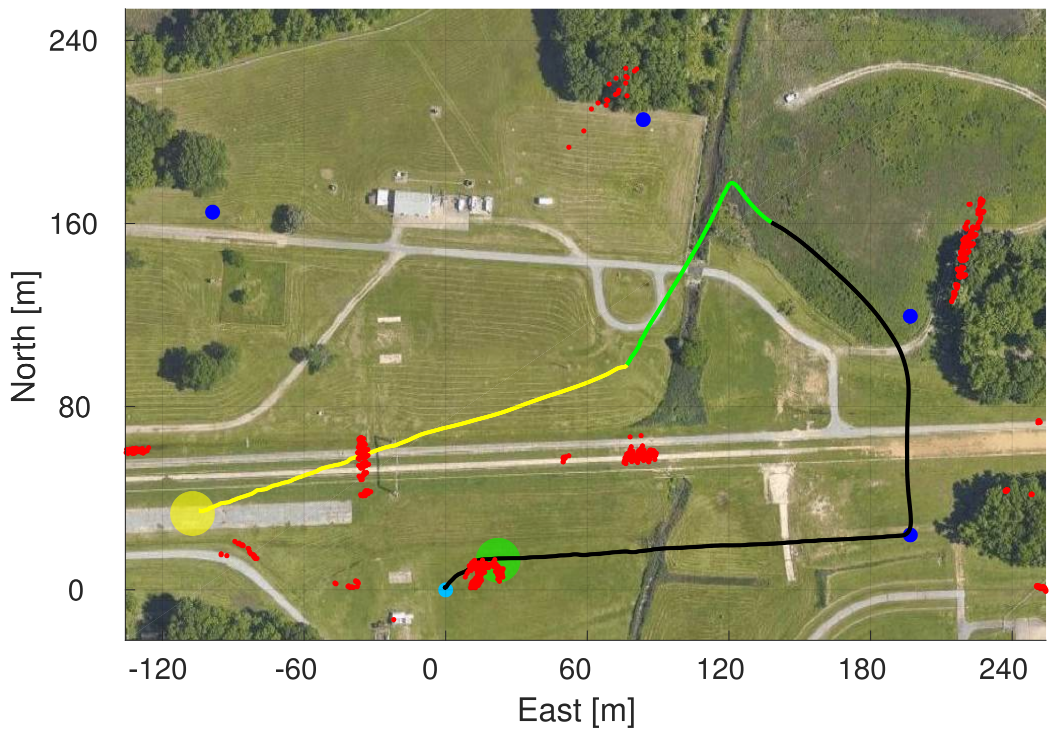


Figure 4.14: Trajectory of flight test #13. Once Safe2Ditch is engaged, the vehicle begins its descent toward the primary ditch site. Once motion is detected in the ditch site (plotted as red), Safe2Ditch reroutes the vehicle to the secondary (yellow) ditch site, where the vehicle successfully lands. The tracks shown in red, except for near the primary ditch site, are the result of parallax due to the 3D structure of the scene. As the vehicle descends, parallax becomes more pronounced and a greater number of false positives are received.

of this paper is the augmentation of a pre-compiled database with on-board vision capabilities, resulting in the Safe2Ditch system. The use of on-board visual tracking in conjunction with a pre-compiled database for initial landing site selection reduces the processing requirements of the vision system, but allows the sUAS to react to non-cooperative obstacles in populated environments. It is an important design criteria that the computational resources needed for managing emergency landing be low so that the normal mission operation is not hindered.

Future work will address the accuracy of the geolocation algorithm needed to determine if obstacles are located within the selected ditch site. To increase the accuracy of the flat-earth model geolocation estimate, the attitude uncertainty should be projected onto the ground plane as well. Then, instead of checking if a single point is within a ditch site, the intersection of the uncertainty and the ditch site would indicate how likely a target is in the ditch site. Alternatively, instead of tracking targets in the image plane, a visual-inertial odometry or visual SLAM approach could be taken to resolve the missing depth information from camera data and track targets directly in the inertial frame. This type of approach would additionally address the effects of parallax when flying low over non-planar environments.

Chapter 5

Conclusion and Future Work

Emergency landing is a critical component of flying vehicles. To safely integrate unmanned aircraft into the NAS, sUAS need greater autonomy so that they can safely land when needed. In this thesis, we presented Safe2Ditch, a crash management system for sUAS that utilizes a pre-compiled database of ditch sites and visual multiple target tracking. Using Safe2Ditch, sUAS can successfully land in environments with non-cooperative moving obstacles. A number of simulation and hardware demonstrations verified the current operational envelope and helped identify areas of future work. The following sections discuss these topics of future research for both Safe2Ditch specifically and multiple target tracking generally.

5.1 Safe2Ditch

As discussed in Chapter 4, the current version of Safe2Ditch assumes that there is always an alternate ditch site with no motion, and that the vehicle has enough energy to get there. Clearly, this may not always be the case and the vehicle may be forced to land in an area that is still semi-populated. Using the information from the on-board camera and the visual tracker, we would like to steer the vehicle to a obstacle-free sub-area within the selected ditch site. Future work should address this need using a control method such as image-based visual servoing (IBVS), which is the idea of using computer vision in a feedback loop to control robot motion [65] Using IBVS to align the optical axis with a line-of-sight vector that points at the safest sub-area in the ditch site, the vehicle could tactically maneuver around non-cooperative obstacles, thereby addressing the “last 50 feet” problem [5].

A point of weakness in the current system is the naive similar-triangles geolocation algorithm used. This algorithm is problematic for two primary reasons: the flat-earth assumption and attitude uncertainty. Using similar triangles to calculate the inertial position of ground targets does not allow the uncertainty of vehicle attitude to be used. A stochastic filtering approach could use the uncertainty information to better estimate where the targets are on the ground. Alternatively, if the attitude uncertainty could be projected through the geolocation equations, while respecting the manifold structure, the attitude uncertainty projected onto the ground could be used to detect obstacles in the ditch site using an intersection-over-union measure. This would alleviate the need for a single point to be inside in the ditch site before a reroute is triggered by Safe2Ditch.

The problem with a flat-earth assumption is especially highlighted in Figure 4.14. Notice the cloud of red markers indicating a tracked object at $(-30, 60)$. As the vehicle flies low (about 40 m) toward the secondary (yellow) ditch site, the tracking system begins tracking a 5 m structure due to parallax. Because this structure is not flat, the geolocation algorithm incorrectly estimates that its position is further west. However, the tracking due to parallax is a fault of its own and is something that could better be addressed in the visual tracking system, as discussed in the next section.

Currently, the visual system only tracks moving objects using a KLT-based feature tracker. Using semantic segmentation could enhance the awareness of Safe2Ditch so that it knows the type of environment (i.e., street, buildings, lake, parking lot) the vehicle is flying in. Further, machine learning approaches such as support vector machines and convolutional neural networks could be used as another measurement source to R-RANSAC to give the ability to classify the types of objects it is tracking. This would give Safe2Ditch additional information that it could use while selecting ditch sites and during tactical maneuvering.

5.2 Visual Tracking

The visual tracker discussed in this thesis is a simple measurement generator for R-RANSAC that can only detect and track moving objects. However, if the activation layers of a convolutional neural network were used correlate historically tracked objects with similar objects, R-RANSAC could receive measurements from targets that never began moving and it would be able to track them.

As mentioned in the previous section, parallax often creates false positives while tracking because it is difficult to separate the 3D structure of the scene from the motion of the camera. While Safe2Ditch uses the false positives caused by parallax to correctly reroute to a flatter, more suitable landing site, it would be ideal if false positives caused by parallax could be separated from tracks from true targets. By decomposing the essential matrix into a rotation and translation between two camera frames, the 3D structure of the scene could better be accounted for.

References

[1] L. M. Pearson, *Developing the Flying Bomb.* Naval Air Systems Command, 1969, pp. 70–73, alternate URL: https://www.scribd.com/document/283873876/Developing-the-Flying-Bomb. [Online]. Available: http://www.history.navy.mil/download/ww1-10.pdf

[2] FAA, "FAA Aerospace Forecast, 2017-2037," Federal Aviation Administration, Tech. Rep., 2017. [Online]. Available: https://www.faa.gov/data_research/aviation/aerospace_forecasts/media/FY2017-37_FAA_Aerospace_Forecast.pdf

[3] R. Allen, M. Pavone, and M. Schwager, "Flying Smartphones: When Portable Computing Sprouts Wings," *IEEE Pervasive Computing*, vol. 15, no. 3, pp. 83–88, 2016.

[4] FAA, "Air Traffic by the Numbers," Federal Aviation Administration, Tech. Rep., 2017. [Online]. Available: https://www.faa.gov/air_traffic/by_the_numbers/media/Air_Traffic_by_the_Numbers_2017_Final.pdf

[5] P. Kopardekar, J. Rios, T. Prevot, M. Johnson, J. Jung, and J. E. Robinson III, "UAS Traffic Management (UTM) Concept of Operations to Safely Enable Low Altitude Flight Operations," in *AIAA Aviation Technology, Integration, and Operations Conference*, June 2016, pp. 1–16. [Online]. Available: http://arc.aiaa.org/doi/10.2514/6.2016-3292

[6] M. R. Endsley, "Toward a Theory of Situation Awareness in Dynamic Systems," *Human Factors: The Journal of the Human Factors and Ergonomics Society*, vol. 37, no. 1, pp. 32–64, 1995. [Online]. Available: http://journals.sagepub.com/doi/10.1518/001872095779049543

[7] ——, "Automation and Situation Awareness," *Automation and Human Performance: Theory and Applications*, pp. 163–181, 1996.

[8] J. A. Adams, "Unmanned Vehicle Situation Awareness: A Path Forward," *Proceedings of the 2007 Human Systems Integration Symposium*, no. 615, 2007.

[9] O. McAree and W. H. Chen, "Artificial situation awareness for increased autonomy of unmanned aerial systems in the terminal area," *Journal of Intelligent and Robotic Systems: Theory and Applications*, vol. 70, no. 1-4, pp. 545–555, 2013.

[10] P. C. Niedfeldt, "Recursive-RANSAC: A Novel Algorithm for Tracking Multiple Targets in Clutter," Ph.D. dissertation, Brigham Young University, 2014. [Online]. Available: http://scholarsarchive.byu.edu/etd/4195

[11] P. C. Niedfeldt and R. W. Beard, "Multiple target tracking using recursive RANSAC," in *American Control Conference*, Portland, Oregon, USA, 2014, pp. 3393–3398. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6859273

[12] K. Ingersoll, P. C. Niedfeldt, and R. W. Beard, "Multiple target tracking and stationary object detection in video with Recursive-RANSAC and tracker-sensor feedback," in *International Conference on Unmanned Aircraft Systems (ICUAS)*, Denver, CO, USA, June 2015, pp. 1320–1329.

[13] J. K. Ingersoll, "Vision-Based Multiple Target Tracking Using Recursive-RANSAC," Master's thesis, Brigham Young University, 2015.

[14] P. C. Defranco, "Detecting and Tracking Moving Objects from a Small Unmanned Air Vehicle," Master's thesis, Brigham Young University, 2015.

[15] E. B. Quist, P. C. Niedfeldt, and R. W. Beard, "Radar odometry with recursive-RANSAC," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 52, no. 4, pp. 1618–1630, 2016.

[16] J. K. Wikle, "Integration of a Complete Detect and Avoid System for Small Unmanned Aircraft Systems," Master's thesis, Brigham Young University, 2017.

[17] P. C. Niedfeldt and R. W. Beard, "Recursive RANSAC: Multiple signal estimation with outliers," *IFAC Proceedings Volumes (IFAC-PapersOnline)*, vol. 9, no. PART 1, pp. 430–435, 2013.

[18] M. A. Fischler and R. C. Bolles, "Random Sample Consensus: A Paradigm for Model Fitting with Applicatlons to Image Analysis and Automated Cartography," *Communications of the ACM*, vol. 24, no. 6, pp. 381 – 395, 1981. [Online]. Available: http://dx.doi.org/10.1145/358669.358692

[19] T. K. Moon and W. C. Stirling, *Mathematical methods and algorithms for signal processing*. Upper Saddle River, NJ 07458, USA: Prentice-Hall, 2000.

[20] X. Li and V. Jilkov, "Survey of maneuvering target tracking: dynamic models," in *Proceedings of SPIE Conference on Signal and Data Processing of Small Targets 2000*, no. April, 2000, pp. 212–235. [Online]. Available: http://proceedings.spiedigitallibrary.org/proceeding.aspx?articleid=905730

[21] K. Mehrotra and P. R. Mahapatra, "A jerk model for tracking highly maneuvering targets," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 33, no. 4, pp. 1094–1105, 1997.

[22] D. Nister, O. Naroditsky, and J. Bergen, "Visual odometry," in *Computer Vision and Pattern Recognition*, vol. 1. Washington, DC, USA: IEEE, 2004, pp. 652–659.

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

[24] G. Guennebaud, B. Jacob *et al.*, "Eigen v3," http://eigen.tuxfamily.org, 2010.

[25] J. D. Millard, “Multiple Target Tracking in Realistic Environments Using Recursive-RANSAC in a Data Fusion Framework,” Master’s thesis, Brigham Young University, 2017.

[26] M. Quigley, B. Gerkey, K. Conley, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler, and A. Ng, “ROS: an open-source robot operating system,” in *Proc. of the IEEE Intl. Conf. on Robotics and Automation (ICRA) Workshop on Open Source Robotics*, Kobe, Japan, May 2009.

[27] P. C. Lusk and R. W. Beard, “Visual multiple target tracking from a descending aerial platform,” in *American Control Conference (ACC)*, Milwaukee, WI, USA, June 2018, pp. 5088–5093.

[28] Amazon.com Inc., “Determining Safe Access with a Best-Equipped, Best-Served Model for Small Unmanned Aircraft Systems,” *NASA UTM: The Next Era of Aviation*, Jul. 2015.

[29] S. Scherer, L. Chamberlain, and S. Singh, “Autonomous landing at unprepared sites by a full-scale helicopter,” *Robotics and Autonomous Systems*, vol. 60, no. 12, pp. 1545–1562, 2012. [Online]. Available: http://dx.doi.org/10.1016/j.robot.2012.09.004

[30] L. Mejias and D. Fitzgerald, “A Multi-layered Approach for Site Detection in UAS Emergency Landing Scenarios using Geometry-Based Image Segmentation,” 2013, pp. 366–372.

[31] W. Luo, J. Xing, A. Milan, X. Zhang, W. Liu, X. Zhao, and T. K. Kim, “Multiple Object Tracking: A Literature Review,” pp. 1–18, 2014. [Online]. Available: http://arxiv.org/abs/1409.7618

[32] A. W. Smeulders, D. M. Chu, R. Cucchiara, S. Calderara, A. Dehghan, and M. Shah, “Visual tracking: An experimental survey,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 36, no. 7, pp. 1442–1468, 2014.

[33] L. Leal-Taixé, A. Milan, K. Schindler, D. Cremers, I. Reid, and S. Roth, "Tracking the Trackers: An Analysis of the State of the Art in Multiple Object Tracking," no. March, 2017. [Online]. Available: http://arxiv.org/abs/1704.02781

[34] M. Andriluka, S. Roth, and B. Schiele, "People-tracking-by-detection and people-detection-by-tracking," *26th IEEE Conference on Computer Vision and Pattern Recognition, CVPR*, 2008.

[35] A. Pieropan, M. Björkman, N. Bergström, and D. Kragic, "Feature Descriptors for Tracking by Detection: a Benchmark," 2016. [Online]. Available: http://arxiv.org/abs/1607.06178

[36] J. Shi and C. Tomasi, "Good Features to Track," in *IEEE Conference on Computer Vision and Pattern Recognition*, Seattle, June 1994, pp. 593–600.

[37] Z. Kalal, K. Mikolajczyk, and J. Matas, "Tracking-learning-detection," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 6, no. 1, pp. 1409–1422, 2010.

[38] 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, July 2017. [Online]. Available: http://ieeexplore.ieee.org/document/7921549/

[39] C. Teulière, L. Eck, and E. Marchand, "Chasing a moving target from a flying UAV," *IEEE International Conference on Intelligent Robots and Systems*, pp. 4929–4934, 2011.

[40] J. Pestana, J. L. Sanchez-Lopez, P. Campoy, and S. Saripalli, "Vision based GPS-denied Object Tracking and following for unmanned aerial vehicles," *2013 IEEE International Symposium on Safety, Security, and Rescue Robotics, SSRR 2013*, 2013.

[41] G. R. Rodríguez-Canosa, S. Thomas, J. del Cerro, A. Barrientos, and B. MacDonald, "A Real-Time Method to Detect and Track Moving Objects (DATMO) from Unmanned Aerial Vehicles (UAVs) Using a Single Camera," *Remote Sensing*, vol. 4, no. 4, pp. 1090–1111, 2012.

[42] G. Klein and D. Murray, "Parallel tracking and mapping for small AR workspaces," *2007 6th IEEE and ACM International Symposium on Mixed and Augmented Reality, ISMAR*, 2007.

[43] X. Jiang and X. Cao, "Surveillance from above: A detection-and-prediction based multiple target tracking method on aerial videos," in *Integrated Communications Navigation and Surveillance (ICNS)*. IEEE, apr 2016, pp. 1–13. [Online]. Available: http://ieeexplore.ieee.org/document/7486348/

[44] J. Li, D. H. Ye, T. Chung, M. Kolsch, J. Wachs, and C. Bouman, "Multi-target detection and tracking from a single camera in Unmanned Aerial Vehicles (UAVs)," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2016, pp. 4992–4997.

[45] M. Teutsch and W. Krüger, "Detection, segmentation, and tracking of moving objects in UAV videos," in *Advanced Video and Signal-Based Surveillance (AVSS)*, 2012, pp. 313–318.

[46] P. C. Niedfeldt, K. Ingersoll, and R. W. Beard, "Comparison and Analysis of Recursive-RANSAC for Multiple Target Tracking," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 53, no. 1, pp. 461–476, feb 2017. [Online]. Available: http://ieeexplore.ieee.org/document/7812590/

[47] K. Bernardin and R. Stiefelhagen, "Evaluating Multiple Object Tracking Performance: The CLEAR MOT Metrics," *EURASIP Journal on Image and Video Processing*, vol.

2008, pp. 1–10, 2008. [Online]. Available: http://jivp.eurasipjournals.com/content/2008/1/246309

[48] C. Vondrick, D. Patterson, and D. Ramanan, “Efficiently scaling up crowdsourced video annotation,” *International Journal of Computer Vision*, vol. 101, no. 1, pp. 184–204, 2012. [Online]. Available: http://dx.doi.org/10.1007/s11263-012-0564-1

[49] M. Michał, A. Wiśniewski, and J. McMillan, “Clarity from above,” *PwC Drone Powered Solutions*, p. 38, May 2016. [Online]. Available: https://www.pwc.pl/pl/pdf/clarity-from-above-pwc.pdf

[50] Amazon.com Inc., “Revising the Airspace Model for the Safe Integration of Small Unmanned Aircraft Systems,” *NASA UTM: The Next Era of Aviation*, no. 1, pp. 2–5, Jul. 2015.

[51] P. Narayan, P. Wu, D. Campbell, and R. Walker, “An Intelligent Control Architecture for Unmanned Aerial Systems (UAS) in the National Airspace System (NAS),” in *2nd Australasian Unmanned Air Vehicle Systems Conference*, March 2007, pp. 20–31.

[52] P. Kopardekar and S. Bradford, “UAS Traffic Management (UTM): Research Transition Team (RTT) Plan,” 2017, accessed: 2017-08-29. [Online]. Available: https://www.faa.gov/uas/research/utm/media/FAA_NASA_UAS_Traffic_Management_Research_Plan.pdf

[53] M. Warren, L. Mejias, X. Yang, B. Arain, F. Gonzalez, and B. Upcroft, “Field and Service Robotics,” in *Springer Tracts in Advanced Robotics*, ser. Springer Tracts in Advanced Robotics, L. Mejias, P. Corke, and J. Roberts, Eds. Springer International Publishing, 2015, vol. 105, ch. Enabling Aircraft Emergency Landings Using Active Visual Site Detection, pp. 167–181. [Online]. Available: http://link.springer.com/10.1007/978-3-319-07488-7

[54] Y. F. Shen, Z. U. Rahman, D. Krusienski, and J. Li, “A vision-based automatic safe landing-site detection system,” *IEEE Transactions on Aerospace and Electronic Systems*, vol. 49, no. 1, pp. 294–311, 2013.

[55] P. Eendebak, A. Van Eekeren, and R. Den Hollander, “Landing spot selection for UAV emergency landing,” *Proceedings of SPIE - The International Society for Optical Engineering*, vol. 8741, no. 5, May 2013.

[56] J. Mackay, G. Ellingson, and T. W. McLain, “Landing Zone Determination for Autonomous Rotorcraft in Surveillance Applications,” in *AIAA Guidance, Navigation, and Control Conference*, January 2016, pp. 1–9. [Online]. Available: http://arc.aiaa.org/doi/10.2514/6.2016-1137

[57] ParaZero, “Parazero - drone safety systems,” https://parazero.com, 2017, accessed: 2018-07-17. [Online]. Available: https://parazero.com

[58] DJI, *Mavic Pro Disclaimer and Safety Guidelines v1.0*, May 2017. [Online]. Available: https://dl.djicdn.com/downloads/mavic/20170809/Mavic+Pro+Disclaimer+and+Safety+Guidelines-EN.pdf

[59] P. F. A. Di Donato and E. M. Atkins, “Evaluating Risk to People and Property for Aircraft Emergency Landing Planning,” *Journal of Aerospace Information Systems*, vol. 14, no. 5, pp. 4–8, 2016.

[60] M. Coombes, W. H. Chen, and P. Render, “Reachability Analysis of Landing Sites for Forced Landing of a UAS,” *Journal of Intelligent & Robotic Systems*, vol. 73, no. 1-4, pp. 635–653, 2014. [Online]. Available: http://link.springer.com/10.1007/s10846-013-9920-9

[61] M. W. Mueller and R. D’Andrea, “Stability and control of a quadrocopter despite the complete loss of one, two, or three propellers,” *Proceedings - IEEE International Conference on Robotics and Automation*, pp. 45–52, 2014.

[62] Z. Pan, S. Liu, and W. Fu, "A review of visual moving target tracking," *Multimedia Tools and Applications*, vol. 76, no. 16, pp. 16 989–17 018, aug 2017. [Online]. Available: http://link.springer.com/10.1007/s11042-016-3647-0

[63] C. Kanellakis and G. Nikolakopoulos, "Survey on Computer Vision for UAVs: Current Developments and Trends," *Journal of Intelligent and Robotic Systems*, vol. 87, no. 1, pp. 141–168, 2017.

[64] R. W. Beard and T. W. McLain, *Small Unmanned Aircraft: Theory and Practice.* Princeton University Press, 2012.

[65] F. Chaumette and S. Hutchinson, "Visual Servo Control I: Basic Approaches," *IEEE Robotics & Automation Magazine*, vol. 13, no. 4, pp. 82–90, dec 2006. [Online]. Available: http://ieeexplore.ieee.org/document/4015997/