Real-Time Visual Multi-Target Tracking in Realistic Tracking Environments

Jacob Harley White
Brigham Young University

Follow this and additional works at: https://scholarsarchive.byu.edu/etd

BYU ScholarsArchive Citation
https://scholarsarchive.byu.edu/etd/7486

This Thesis is brought to you for free and open access by BYU ScholarsArchive. It has been accepted for inclusion in Theses and Dissertations by an authorized administrator of BYU ScholarsArchive. For more information, please contact ellen_amatangelo@byu.edu.
Real-Time Visual Multi-Target Tracking
in Realistic Tracking Environments

Jacob Harley White

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

Randal W. Beard, Chair
Timothy W. McLain
Dah Jye Lee

Department of Electrical and Computer Engineering
Brigham Young University

Copyright © 2019 Jacob Harley White
All Rights Reserved
ABSTRACT

Real-Time Visual Multi-Target Tracking in Realistic Tracking Environments

Jacob Harley White
Department of Electrical and Computer Engineering, BYU
Master of Science

This thesis focuses on visual multiple-target tracking (MTT) from a UAV. Typical state-of-the-art multiple-target trackers rely on an object detector as the primary detection source. However, object detectors usually require a GPU to process images in real-time, which may not be feasible to carry on-board a UAV. Additionally, they often do not produce consistent detections for small objects typical of UAV imagery.

In our method, we instead detect motion to identify objects of interest in the scene. We detect motion at corners in the image using optical flow. We also track points long-term to continue tracking stopped objects. Since our motion detection algorithm generates multiple detections at each time-step, we use a hybrid probabilistic data association filter combined with a single iteration of expectation maximization to improve tracking accuracy.

We also present a motion detection algorithm that accounts for parallax in non-planar UAV imagery. We use the essential matrix to distinguish between true object motion and apparent object motion due to parallax. Instead of calculating the essential matrix directly, which can be time-consuming, we design a new algorithm that optimizes the rotation and translation between frames. This new algorithm requires only 4 ms instead of 47 ms per frame of the video sequence.

We demonstrate the performance of these algorithms on video data. These algorithms are shown to improve tracking accuracy, reliability, and speed. All these contributions are capable of running in real-time without a GPU.

Keywords: unmanned aerial vehicle, multiple target tracking, motion detection, stationary object tracking, homography, probabilistic data association, relative pose estimation, essential matrix, parallax
ACKNOWLEDGMENTS

I have learned much during my experience as a graduate student in the MAGICC Lab at Brigham Young University. None of this learning would have been possible without the support and help from many others around me. I am grateful to my professors who have sacrificed time to teach and mentor me and who have worked diligently to provide me with meaningful classroom experiences and one-on-one instruction. I’d especially like to thank Dr. Beard for his time and the valuable insight he provided during our regular research meetings, during the many classes I took from him, and as he reviewed papers I was preparing to publish.

I’d like to thank my colleagues in the MAGICC lab, specifically Jeff Millard, Parker Lusk, and Jerel Nielsen, for their discussions and advice on target tracking, filtering, and computer vision algorithms. I’d also like to thank Karl Salva, my summer internship mentor from the Air Force Research Laboratory, who mentored me on target tracking methodologies during the early stages of my research.

I especially appreciate the support and influence from my family. I’d like to thank my father who mentored me in math, writing, and programming, and helped me develop a love for computers and engineering early in my youth. I’d also like to thank my wife, Kylie, for her support and encouragement throughout my graduate school experience.

This work has been funded by the Center for Unmanned Aircraft Systems (C-UAS), a National Science Foundation Industry/University Cooperative Research Center (I/UCRC) under NSF award No. IIP-1650547 along with significant contributions from C-UAS industry members.
TABLE OF CONTENTS

List of Tables ................................................................................... vi

List of Figures ................................................................................... vii

Chapter 1 Introduction ........................................................................ 1
  1.1 Motivation .................................................................................. 1
  1.2 Previous Work .......................................................................... 1
  1.3 Contributions ......................................................................... 3
  1.4 Thesis Organization .................................................................. 4

Chapter 2 Extending Motion Detection to Track Stopped Objects in Visual Multi-Target Tracking ................................................................................... 5
  2.1 Introduction ................................................................................ 5
  2.2 KLT-Based Motion Detection Using Homography Outliers .......... 7
  2.3 Recursive-RANSAC .................................................................. 7
  2.4 Persistent Point Tracking Using Optical Flow and Feature Matching ... 8
  2.5 Grid-Based Feature Prioritization (GFP) ....................................... 9
  2.6 Reducing Ghost Tracks ............................................................. 12
  2.7 Complete Tracker Pipeline ....................................................... 14
  2.8 Results .................................................................................... 14
  2.9 Conclusion ............................................................................. 17

Chapter 3 The Membership Probabilistic Data Association (MPDA) Filter ......................................................... 20
  3.1 Introduction ............................................................................. 20
  3.2 KLT-Based Motion Detection with Stopped Object Tracking as a Measurement Source ......................................................... 22
  3.3 Recursive RANSAC ................................................................. 23
  3.4 Probabilistic Data Association Filter .......................................... 23
  3.5 Expectation Maximization .......................................................... 25
  3.6 Incorporating Membership Probabilities into the Probabilistic Data Association Filter ......................................................... 28
  3.7 Nearly Constant Velocity Target Model ....................................... 29
  3.8 Results .................................................................................... 30
     3.8.1 Video Result Analysis with Hand-Tuned Parameters .......... 31
     3.8.2 Parameter Sweeps ............................................................... 32
  3.9 Conclusion ............................................................................. 37

Chapter 4 The Homography as a State Transformation Between Frames in Visual Multi-Target Tracking ......................................................... 40
  4.1 Introduction ............................................................................. 40
  4.2 The Homography as a Non-Linear Transformation ...................... 43
  4.3 Transforming the Target’s Velocity .............................................. 43
  4.4 Transforming the Target’s Covariance ........................................ 46
LIST OF TABLES

2.1 Tracker timing results .................................................. 18
3.1 Tracker MOTA and MOTP scores ....................................... 31
3.2 Computation times ...................................................... 32
5.1 Relative pose cost functions ........................................ 88
5.2 Gauss-Newton and Levenberg-Marquardt error comparison ......... 88
5.3 Pose disambiguation comparison ..................................... 90
5.4 ReSORtSAC relative pose refinement ................................ 91
5.5 Computation time ...................................................... 92
5.6 Motion detection and tracking computation times .................. 94
### LIST OF FIGURES

2.1 Motion detection results .............................................. 8
2.2 Feature matches over time ............................................ 9
2.3 Persistent point tracking “explosion” ................................. 10
2.4 GFP search optimization .............................................. 11
2.5 GFP results ............................................................. 12
2.6 Examples of non-existent ghost tracks ................................. 13
2.7 Block diagram of the complete visual multi-target tracker .......... 15
2.8 Challenges in video sequence ......................................... 16
2.9 Tracking accuracy for one of the targets in the video sequence .... 17
2.10 Comparison with other state-of-the-art trackers ...................... 18

3.1 Example of receiving multiple detections from a target at a single time step 21
3.2 PDA and MPDA weights .................................................. 29
3.3 Example of crossing targets ............................................ 33
3.4 Example of tracks coalescing .......................................... 34
3.5 Example of parallel traveling targets .................................. 35
3.6 Example of lack of convergence ....................................... 36
3.7 Position sensor noise parameter sweep ................................ 37
3.8 Velocity sensor noise parameter sweep ................................. 38
3.9 Process noise parameter sweep ........................................ 39

4.1 The geometry for the derivation of the homography matrix between two camera poses 41
4.2 The target’s position, velocity, and position covariance in the image plane 46
4.3 Results of covariance transformation .................................. 48
4.4 Comparison of three methods of transforming points, velocities, and covariances 50
4.5 Matlab UAV simulation .................................................... 52
4.6 UAV and target world position over time .............................. 53
4.7 Target states estimated using a Kalman filter .......................... 54

5.1 The geometry for the derivation of the essential matrix ................ 58
5.2 Block diagram of ReSORtSAC .......................................... 77
5.3 Screenshot of the holodeck video sequence ............................ 84
5.4 Error over entire video sequence ...................................... 85
5.5 ReSORtSAC Levenberg-Marquardt seeding methods .................. 86
5.6 ReSORtSAC and OpenCV error over time comparison ............... 87
5.7 Gauss-Newton and Levenberg-Marquardt convergence ............... 89
5.8 Gauss-Newton and Levenberg-Marquardt average step size ........... 90
5.9 Average rotation and translation errors for RANSAC and LMedS .... 91
5.10 Time required to generate each hypothesis set ......................... 92
5.11 Video motion detection results ........................................ 93
5.12 Recursive RANSAC tracks .............................................. 94

6.1 Occlusion search region ............................................... 99
B.1 Two solutions for the rotation matrix basis vectors . . . . . . . . . . . . . . . . . . . . 113
B.2 Two solutions for the rotation matrix after enforcing a right-hand coordinate system . . 114
CHAPTER 1. INTRODUCTION

1.1 Motivation

In recent years, the popularity and availability of small unmanned aerial vehicles (UAVs) have increased. Some of the many applications of UAVs include the military, law enforcement, photography, search and rescue, infrastructure monitoring, wide-area surveillance, and package delivery. Over time UAVs have become more reliable and affordable; however, they are not yet able to operate autonomously. Operation usually requires constant attention from a trained pilot to remotely control the vehicle. To achieve the full benefit of drones in these applications, autonomous operation with limited human intervention is required.

To ensure safe operation, the UAV must avoid contact with humans, ground vehicles, and other aircraft. In many cases, the UAV may also be required to follow one of these objects at a distance. In order to perform these tasks autonomously, the UAV must be able to detect nearby objects of interest and estimate their position and velocity. This task is called multiple target tracking (MTT) or multiple object tracking (MOT) in the literature. This document focuses on multiple target tracking from a UAV camera.

1.2 Previous Work

There are a plethora of multiple target trackers in the literature. However, many of these trackers are unable to run in real-time or require hardware that is infeasible to carry on-board a UAV.

Current state-of-the-art multiple object trackers are moving towards using object detectors trained using deep learning as the primary detection source [1]–[5]. The advantage of this approach is that the object detector outputs a single bounding box for each target of interest and only needs a single frame to detect the objects. However, the object detection process is the bottleneck of these
trackers, making real-time implementation challenging. Until recently most of the object detectors used a two-stage detection process, where the first stage generates candidate bounding boxes, which are then classified in the second stage. Faster R-CNN [3] is one example of a two-stage object detector. Since then object detectors have been improved to use a single-stage detection pipeline, reducing computation and allowing these object detectors to run in real-time. YOLO [4] and SSD [5] are two examples of single-stage object detectors. Even so, these object detectors still require a GPU for real-time performance, which may not be feasible to carry on-board a UAV.

Other state-of-the-art tracking algorithms focus on tracking only one object [6]–[9]. Instead of using an object detector, these trackers rely on a template initialized once at the beginning of the video sequence. This enables real-time performance without requiring a GPU. However, these trackers require initializing the target of interest manually at the beginning of the video sequence which may not be feasible for autonomous operation, particularly if there are multiple objects of interest in the scene.

Another approach that can enable real-time performance is to detect motion in the image instead of relying on object detectors. Motion detection algorithms, often referred to as background subtraction algorithms in the literature, are typically faster to run than neural network object detectors, thus enabling real-time performance without a GPU.

The most common motion detection algorithms are image-based, meaning they generate a foreground mask the same size as the original image. The foreground blobs can then be detected and fed to the tracker. On a stationary camera, this foreground mask can be generated by analyzing the pixels in the image over time, using techniques such a mixture of Gaussians [10]. On a moving camera, if the scene is approximately planar or the camera motion mostly rotational, a homography can be used to align the images. Image differencing can then be used to generate a foreground mask [11], [12]. However, if the scene is not planar and the camera is translating, the resulting parallax can cause significant artifacts in the foreground mask.

Motion detection algorithms that handle parallax in the scene are typically feature-based instead of image-based [13]–[15]. Rather than analyzing each pixel over time, these algorithms track the locations of features in the image over time in order to determine which are from moving objects. The locations of features in the image can be tracked in real-time without requiring a GPU using algorithms such as Lucas-Kanade optical flow [16].
To design a tracker capable of running in real-time, on-board the UAV, and without requiring a GPU, we have chosen to focus on tracking using feature-based motion detection algorithms as the primary detection source. However, there are still obstacles to making a tracking algorithm which uses feature-based motion detection work well for tracking from a UAV camera. One challenge is that motion detection algorithms struggle to detect objects once they have stopped. This challenge is addressed in Chapter 2. Another challenge is that feature-based approaches often return multiple detections per object. This can reduce tracking quality if the tracking algorithm assumes each object only returns one detection per time step. This challenge is addressed in Chapter 3. Additionally, tracking objects in the image plane requires transforming the coordinates of targets between frames at each time step. The derivation of these transformations is presented in Chapter 4. Lastly, the motion detection algorithm must account for parallax while running in real-time. This challenge is discussed in Chapter 5.

1.3 Contributions

The contributions of this thesis are outlined below.

- This thesis extends the MAGICC lab visual multi-target tracker to be able to track stopped objects from a moving camera.
- It improves the probabilistic data association component of the tracker to track clusters of detections with greater accuracy.
- It derives planar transformations of the velocity, covariance, and higher order terms for tracking targets in the image plane.
- It introduces a simple parallax-compensated motion detection algorithm to be used with the tracker.
- It presents a light-weight algorithm for estimating the rotation and translation between consecutive frames of a video sequence.
1.4 Thesis Organization

The remainder of the thesis is organized as follows. Chapter 2 introduces a novel method for increasing the robustness of the visual multi-target tracker developed in the MAGICC lab [17], by adding the capability to track stopped objects. This algorithm uses persistent point tracking and a feature prioritization scheme to ensure the number of points being tracked is tractable, while still obtaining information from each area of the image. A method of reducing non-existent ghost tracks is also presented. Results are shown on a moving camera video sequence and compared with state-of-the-art trackers.

Chapter 3 presents a novel membership probabilistic data association (MPDA) filter. This filter accounts for more than one measurement per target at each time step and uses clustering methodologies from the statistics community to produce more accurate target estimates. The filter is benchmarked on a video sequence and is shown to reduce track coalescence and help target estimates converge more quickly to the center of targets. The filter is compared to the original probabilistic data association filter and the joint probabilistic data association filter.

Chapter 4 describes an algorithm for using the homography matrix to transform the image plane projection of the position, velocity, and covariance of targets from one image frame to the next image frame. The derivation of this transformation is shown using calculus and probability principles. Potential pitfalls of the derivation of the transformation are also discussed. The effectiveness of the transformation is shown on simulated tracking data and video sequences.

Chapter 5 describes an algorithm for parallax-compensated motion detection. This motion detection algorithm uses the rotation and translation between frames to distinguish between true object motion and apparent object motion due to parallax. A novel iterative five-point algorithm is introduced that optimizes the rotation and translation between consecutive frames of the video sequence. The accuracy of the algorithm is shown to be comparable to the OpenCV essential matrix solver, but an order of magnitude faster, thus enabling real-time performance.

Conclusions and future work are given in Chapter 6. The contributions of this thesis are summarized. Limitations of and potential improvements to the tracking algorithms are also discussed.
CHAPTER 2. EXTENDING MOTION DETECTION TO TRACK STOPPED OBJECTS IN VISUAL MULTI-TARGET TRACKING

2.1 Introduction

Visual multi-target tracking from an unmanned aerial vehicle (UAV) has many applications including wide-area surveillance, search and rescue, photography, law enforcement, and military operations. These applications, however, have different requirements than those of traditional state-of-the-art trackers. A tracker in these scenarios must be able to run in real time from a UAV, track from a moving camera, and track multiple objects simultaneously.

Perhaps the most critical distinction between our work and state-of-the-art trackers is the requirement that the algorithm not only run in real-time but also run on-board the UAV. On-board computation avoids many of the challenges inherent in point-to-point video links including data corruption and compression artifacts, which can degrade the performance of the tracker. Tracking on-board the UAV often limits the available computational power.

Many state-of-the-art algorithms commonly used on stationary cameras, such as image-based background subtraction, cannot be used on a moving camera, because the pixel locations of stationary objects are not constant across time.

There are many single object trackers [6]–[9], some of which are capable of running in real time. However, it is difficult to use these trackers for multiple objects because they lack automatic track initialization and, assuming the required computation scales linearly, may no longer run in real time when tracking multiple objects.

Multiple object trackers [19], on the other hand, can automatically initialize new tracks. These trackers rely on a multiple-class object detector trained using deep learning. There are two main obstacles to using object detectors for tracking from a UAV. First, many of these object detectors do not run in real time. For example, DPM [2] and Faster R-CNN [3], two commonly

1 This paper was written by Jacob H. White, Karl T. Salva, and Randal W. Beard, and published in the American Controls Conference [18]
used object detectors in the multiple object tracking challenge [1], run at 0.5 and 5 frames per second respectively. Recently several real-time object detection frameworks have been proposed, including YOLO [20] and SSD [5]. These may enable object detectors to be used for real time tracking. Even so, real-time performance is only achieved with a GPU, which may not be feasible to carry on-board a small UAV. Second, object detectors struggle to detect far away objects typical in aerial imagery, likely because they are not usually trained on aerial imagery. Retraining the object detectors on aerial imagery can improve performance [21].

Wide-area motion imagery (WAMI) trackers [11], [12] must use a different set of algorithms because of the enormous camera resolution and limited frame rate of 1 frame every 1 or 2 seconds. WAMI trackers are one of the few trackers that rely on motion detection rather than a multi-class object detector or a user-specified bounding box to initialize tracks. However, the large image resolution makes it very challenging to process in real time. As a result, we focus in this paper on more standard resolution imagery taken at lower altitudes.

The contribution of this paper is a multi-target tracker capable of tracking moving and stopped objects in real time from a moving camera. Our work uses recursive-RANSAC [22] for track estimation and track management. Our work extends the tracker presented in [17] which successfully tracks multiple objects from a moving camera but fails to track these objects once they have stopped. To track stopped objects, we combine the motion detector used in [17] with BRIEF feature matching [23] and Lucas-Kanade optical flow [16] for persistent point tracking. We use grid-based feature prioritization, denoted as GFP, to keep the number of tracked points small enough to run in real time. We use a track failure detection method to reduce non-existent ghost tracks that sometimes arise during track occlusions.

The remainder of the paper is outlined as follows. Sections 2 and 3 describe motion detection and recursive-RANSAC, the previous methods we extend in this paper. Section 4 and 5 describe our method of persistent point tracking and reducing duplicate points using grid-based feature prioritization. Section 6 describes a method for detecting tracking failure. Section 7 summarizes the complete visual multi-target tracker. The results on a moving camera video sequence are discussed in Section 8.
2.2 KLT-Based Motion Detection Using Homography Outliers

A simple motion detection technique that works on a moving camera is using homography outliers as described in [17]. Good features to track [24] are first detected in the current frame and then matched to the previous frame using optical flow. A homography is then fit to these points to describe the motion between the previous and current frames. Outliers to the homography are classified as moving points (see Algorithm 1). The results of this algorithm on a moving camera sequence are shown in Figure 2.1.

The homography works well to align the frames if the motion is mostly rotational or if the UAV altitude is large when compared to the non-planar depth variations. This is often true for UAVs. When these assumptions do not hold, stationary objects can appear to be moving. This effect is known as parallax. There are more complicated methods for motion detection that account for parallax [13]–[15].

Algorithm 1 Motion Detection

1: Detect features in the current frame
2: Calculate velocity of features using LK optical flow
3: Align previous and current frame using a homography
4: Transform points in previous frame to current frame
5: Subtract camera motion to obtain net pixel velocity
6: Threshold net pixel velocity to find moving points

2.3 Recursive-RANSAC

Moving points found using motion detection are then fed into recursive-RANSAC, a newly proposed algorithm for multi-target tracking presented in [22]. At each time step recursive-RANSAC searches for new models using RANSAC. When a sufficient number of inliers are detected, a new track is initialized. Existing tracks are propagated forward using a Kalman filter. Probabilistic data association [25] is used to account for measurement association uncertainty. Each track is given an inlier score based on the percentage of time steps in which the track is detected. recursive-RANSAC also has a track management system that merges similar tracks and removes tracks that have an inlier score lower than the minimum threshold.
2.4 Persistent Point Tracking Using Optical Flow and Feature Matching

Using motion detection as an input to recursive-RANSAC allows any moving object to be tracked. However, moving objects that subsequently stop can no longer be tracked using this approach. For this reason, we propose extending motion detection to track stopped objects by using long-term point tracking. Long-term point tracking can be done using optical flow or feature matching. Only one method is required, but combining both has the potential to improve overall tracking accuracy. In this paper we will use Lucas-Kanade optical flow [16] and BRIEF feature matching [23]. Though there are more sophisticated feature descriptors, BRIEF has the advantage of being easily run in real time, while still being accurate.

Each point being tracked will be denoted as the tuple $\phi_{t,i}^s = \{s, p, v, a, f\}$, where $t$ is the time step, $i$ is the point index, $s$ is the source index, $p$ is the point position, $v$ is the point net velocity, $a$ is the point track association, and $f$ is the feature descriptor. The net velocity for persistent points is calculated in the same manner as described in the motion detection algorithm. The track association is determined by R-RANSAC when a new track is initialized or when points fall within the gate of an existing track. The feature descriptor is stored the first time the tracked
point is initialized. This collection of information is sufficient to allow long-term point tracking using optical flow and feature matching.

Optical flow calculations are made between consecutive frames (in other words the template is updated each frame). This method of point tracking is potentially more susceptible to gradual drift, but is more accurate between consecutive frames because it avoids jerky estimates due to non-translational image warps and non-rigid object motion. Feature matching, on the other hand, is calculated without updating the template at each time step. This allows lost points to be recovered and has the potential to help distinguish between crossing and interacting targets (see Figure 2.2). Features are only matched if the feature hamming distance falls below a set threshold. The location of the associated track (assigned by recursive-RANSAC) defines a search region which reduces unnecessary feature comparisons.

![Figure 2.2: Feature matches over time. Features are detected in the current frame (bottom right, shown in green) and successfully matched to previous frames (matches shown in blue). Few (in this case none) of the features on the original target are matched to the crossing target. Note that the number of matched features in each frame grows and shrinks over time as the appearance of the target changes.](image)

2.5 Grid-Based Feature Prioritization (GFP)

If every moving feature were stored in memory and tracked using both optical flow and feature matching, the large number of accumulating features would quickly become computationally intractable (see Figure 2.3). As a result, features associated with a particular track must be prioritized and less-important features thrown out. To prioritize these features we use
something we will call grid-based feature prioritization (GFP), based on the feature prioritization algorithm in OpenCV’s GoodFeaturesToTrack function [26].

Figure 2.3: Each new detected point is tracked using both optical flow (green) and feature matching (blue). Without a method that filters out old or redundant points, the resulting “explosion” of points becomes computationally intractable.

GFP adds incoming features to a temporary grid data structure, while ensuring that none of the features are closer than the minimum allowed distance between features. Before being added to the grid the features are sorted by priority. Initially the grid is empty and so almost all of the high-priority features are added. But as more and more features are added, eventually the lower-priority features no longer meet the minimum distance requirement and so they are discarded (Algorithm 2). The grid data structure is used for efficiency so that whenever a new feature is added only the adjacent cells need to be searched rather than searching the entire grid (see Figure 2.4).

The main advantage of this strategy is that it maximizes the amount of information gained from each area of the image while reducing computational complexity. For example, when optical flow is used to track non-rigid objects such as pedestrians, the features quickly clump together because of crossing limbs. After some time the points are basically duplicates of one another and the extra computation no longer provides any additional information to the tracker. The
Algorithm 2 GFP Algorithm

1: Score features using type and/or quality of feature
2: Sort features with highest priority first
3: for Each feature \(i\) do
4:   if feature meets minimum distance requirement then
5:     Add feature
6:   else
7:     Discard feature
8:   end if
9: end for

Figure 2.4: The GFP algorithm can be optimized by storing the points in a grid data structure. As a result of this modification, only the adjacent cells (9 cells total) must be searched each time a new candidate point is added, as opposed to searching the entire collection of points.
minimum distance requirement ensures that each point is unique and contributes a valuable piece of information to the tracker.

In our setup, we have chosen the moving object detection to be the highest priority because it is relied on for track initialization and gives original information about the location of new and existing targets. Feature matching and optical flow are not original sources since they can only be used to compare the current frame to earlier frames in which the moving objects were detected. Feature matching and optical flow are used whenever the corresponding region of the image has no moving object detections. The result of the feature prioritization algorithm is shown in Figure 2.5.

![Figure 2.5: Moving object detections (teal), feature matches (blue), and optical flow (green) are combined using grid-based feature prioritization (GFP).](image)

### 2.6 Reducing Ghost Tracks

Often non-existent ghost tracks are a side-effect of attempting to track stationary objects. Ghost tracks usually occur when an object being tracked disappears behind an occlusion. If the tracking methods do not detect that the object is no longer visible they will begin to track the occlusion instead of the original object. Examples of ghost tracks are shown in Figure 2.6.

Automatic detection of tracking failure is arguably just as important as automatic track initialization. In this paper we detect track failure by comparing the current frame to a background image to determine whether the current frame still contains the object. If the two images are different, the difference implies that something is present in the current frame. If the current frame is identical to the background image, the object being tracked must no longer be visible. Optical flow can be used to determine similarity between the background image and the current frame. If
the calculated motion is greater than a set threshold, the point is kept, otherwise it is discarded to avoid creating ghost tracks. In our tests a threshold of 1 pixel gave good results.

Instead of maintaining a background model, which on a moving camera is not trivial, we look for a recent previous image that contains background near the object of interest. The main challenge is to decide which frame contains background. A simple method to determine whether an area of the image contains foreground or background is to keep a history of recent measurements and check for measurements in the area of interest at the given time step. If there are no nearby measurements at that time step, the frame is likely to contain only background. Since not all movement is observed in the image, there is still some uncertainty in whether these frames contain background. To improve the ghost track reduction accuracy, multiple candidate background frames are compared to the current frame. After a few time steps the tracked point is flagged as a verified stationary point and no further comparisons are performed.

On a stationary camera, optical flow can be calculated directly on the two images without any image registration. However, on a moving camera the images must first be aligned using a homography or other transformation. In our approach we do not calculate this homography from scratch but instead concatenate previously-calculated homographies to determine the transformation. This is more susceptible to drift over time, but since most of the time we are only going back a couple of frames the registration is quite accurate.

Other methods for detecting track failure include forward-backward error [27] and affine motion dissimilarity [24]. Analyzing and comparing these track failure detection methods is beyond the scope of this paper.
2.7 Complete Tracker Pipeline

The complete visual multi-target tracker pipeline is shown in Figure 2.7. The diagram shows the flow of tracked points through the system. Three measurement sources are shown on the left of the figure: KLT-based motion detection, KLT point tracking, and BRIEF feature matching. These sources are combined using grid feature prioritization (GFP) to eliminate redundant measurements. The points are then fed into recursive-RANSAC to remove outlier measurements. The remaining points persist and are tracked at the next time step.

Each collection of points is denoted as \( \Phi_t = \{ \phi_{t,1}, \phi_{t,2}, \ldots, \phi_{t,n} \} \), where \( t \) indicates the current time step in frames. The superscript indicates the source of the collection of points. The output of the three measurement sources are denoted as \( \Phi^1_t \), \( \Phi^2_t \), \( \Phi^3_t \). The output of GFP and recursive-RANSAC are denoted as \( \Phi^F_t \) and \( \Phi^R_t \) respectively.

The combination and filtering of measurement sources can be shown using set notation. The collection of points resulting from GFP is

\[
\Phi^F_t = GFP \left( \Phi^1_t \cup \Phi^2_t \cup \Phi^3_t \right).
\]

The collection of points resulting from recursive-RANSAC is

\[
\Phi^R_t = RR \left( \Phi^F_t \right),
\]

where RR denotes recursive-RANSAC. An important characteristic of GFP and R-RANSAC is that neither perform any kind of averaging; they only accept or reject points. Specifically, the collections of points are subsets of each other, so that

\[
\Phi^R_t \subseteq \Phi^F_t \subseteq \left( \Phi^1_t \cup \Phi^2_t \cup \Phi^3_t \right).
\]

The output of the multi-target tracker is track estimates with position and velocity.

2.8 Results

The complete tracker was tested on multiple video sequences. However, for brevity only results on one of the video sequences are included in this paper. A video of the results is available.
at https://youtu.be/6bXjKb_6qY. The video sequence is from a hand-held moving camera with significant rotation but almost no translation. This motion is characteristic of UAV motion at high altitudes since the translational component is small in comparison to the altitude of the UAV. It is also similar to UAV motion at low altitudes in planar scenes, from the standpoint of the accuracy of the homography transformation. Figure 2.8 shows the challenges in the video sequence.

The results of the tracker are shown in Figure 2.9. Tracking using only motion detection is not able to track targets once they stop. As a result the target is lost and later a new track is initialized when the person resumes walking.

Tracking using motion detection and optical flow is able to track stopped targets. However, without a ghost track reduction method, the track is lost when the pedestrian walks behind a tree (near frame 1300). Occasional tracking errors throughout the video sequence also cause the estimate to lag behind the true trajectory.
Tracking using motion detection, optical flow, and ghost track reduction successfully tracks the target when stopped and does not lose the target during the short occlusion.

Adding feature matching in addition to motion detection, optical flow, and ghost track reduction does not noticeably improve the tracking accuracy. This is likely because motion detection and optical flow combined already do a good job distinguishing between foreground and background points. Feature matching is most helpful when distinguishing between interacting targets. However, the probabilistic data association filter in our implementation does not use the results of feature matching information to determine track association. This is beyond the scope of this paper and will be implemented in future work.

Figure 2.10 compares the stopped object tracker with other state-of-the-art trackers. The single object tracker (ECO) performs well, but cannot track multiple objects or automatically initialize tracks. The trackers that rely on object detectors (Deep SORT with Faster R-CNN and YOLO) have poor track continuity because the object detectors struggle to detect small objects in the image.

The computation speed in frames per second of each tracker is shown in Table 2.1. The video size used in the comparison is 1280×720 pixels. Computation speeds marked with a star (*) are taken from the original paper describing each method. In our implementation every 3rd frame is used in the video sequence, which means the algorithm is capable of running in real time from...
Figure 2.9: Tracking accuracy for one of the targets in the video sequence. Each track id is plotted in a separate color. Tracking using only motion detection fails when the object stops. Persistent point tracking without ghost track reduction lags behind and eventually fails at the occlusion. Persistent point tracking using optical flow with ghost track reduction succeeds in tracking the target for the entire video sequence. Adding feature matching to these methods does not noticeably improve the tracking estimate.

a 30 fps camera. The ECO and YOLO trackers are also capable of running in real time. However, our tracker is the only tracker listed capable of tracking multiple objects in real time without a GPU.

2.9 Conclusion

In this work, we have presented a tracker capable of tracking multiple objects in real time from a moving platform. Our tracker combines motion detection, optical flow, and feature
Figure 2.10: Comparison with other state-of-the-art trackers for one of the targets in the video sequence. Each track id is plotted in a separate color. Our tracker and the ECO tracker successfully track the target for the entire sequence. Note that the ECO tracker only tracks one target and requires manual track initialization. Tracking using Deep SORT with Faster R-CNN detections gives poor track continuity. Using Deep SORT with YOLO instead gives slightly better results, though the track is still quite fragmented.

Table 2.1: Tracker timing results

<table>
<thead>
<tr>
<th>Tracker</th>
<th>FPS</th>
<th>Hardware</th>
</tr>
</thead>
<tbody>
<tr>
<td>Motion detection</td>
<td>18.5</td>
<td>i7 CPU</td>
</tr>
<tr>
<td>No Ghost</td>
<td>15.6</td>
<td>i7 CPU</td>
</tr>
<tr>
<td>Optical flow (Ours)</td>
<td>12.2</td>
<td>i7 CPU</td>
</tr>
<tr>
<td>Feature matching (Ours)</td>
<td>10.6</td>
<td>i7 CPU</td>
</tr>
<tr>
<td>ECO-HC [8]</td>
<td>60*</td>
<td>K40 GPU</td>
</tr>
</tbody>
</table>
matching using a grid-based feature prioritization algorithm that reduces computational complexity and spatial redundancy. Recursive-RANSAC is used to automatically initialize new tracks and reject outlier measurements. Ghost track reduction helps avoid tracking non-existent objects when true objects are no longer visible.

In the future we plan to extend this work to accurately track objects in the presence of parallax. We also plan to use the results of feature matching in the probabilistic data association filter to better distinguish between interacting targets.
CHAPTER 3. THE MEMBERSHIP PROBABILISTIC DATA ASSOCIATION (MPDA) FILTER

3.1 Introduction

There are many applications where an unmanned aerial vehicle (UAV) is required to track multiple targets using an on-board camera. In these applications, a computer vision algorithm is often used to produce detections of targets of interest which are then fed into a tracking algorithm. It is not usually known which detection corresponds to which target. One of the essential tasks of the tracking algorithm is to associate detections to target estimates.

One well-known data association algorithm is the probabilistic data association (PDA) filter [25]. This filter calculates a weighted average with the weights determined by evaluating a Gaussian probability density function (PDF) for each measurement. The larger the error between the estimated target location and the received measurement, the smaller the weight for that measurement will be. This weighted average is then fed into a Kalman filter. The PDA filter also accounts for the null hypothesis, in other words, the possibility that none of the measurements at the given time step originated from the target. The PDA filter performs well in the presence of clutter, intermittent measurements, and crossing targets.

However, one of the weaknesses of the PDA filter is that it is only designed to track a single target at a time. When tracking multiple targets, the PDA filter assumes that the measurement probabilities for each target are independent. It analyzes each target separately, allowing measurements to be double counted which can often cause tracks to coalesce.

The joint probabilistic data association (JPDA) filter [28] overcomes this weakness by calculating the joint PDF of the association probabilities. The JPDA filter enumerates all possible joint target-to-measurement association events, but discards any events where two targets are associated with the same measurement. The probability for each of these joint events is calculated as the product of the probability of each of the single target-to-measurement association events.
The marginal probabilities are then summed over all the joint events and are used as weights when calculating the weighted average before it is fed to the Kalman filter.

The JPDA filter is an improvement over the PDA filter and handles interacting targets better. However, both the JPDA and PDA filters make the assumption that each target only produces one detection. This assumption is likely true for RADAR tracking, but when tracking objects from a camera, computer vision algorithms often return multiple blob centroids or point detections. An example of receiving multiple detections from a single target is shown in Figure 3.1. When the PDA filter receives more than one detection, the additional detections are treated as clutter. The closest detection to the target estimate receives a higher weight than the other detections, biasing the result. This can cause the target estimate to converge slower to the center of the target, and in some cases, only converge to the edge of the target.

Figure 3.1: Example of receiving multiple detections from a target at a single time step

The multiple-detection probabilistic data association (MD-PDA) filter [29] overcomes this weakness by enumerating all possible measurement-to-target association events instead of enumerating the target-to-measurement association events. This allows multiple detections to be associated with the same target and treated as valid measurements instead of clutter. It is only designed to track a single target, however, there is also an improvement to this filter called the multiple-detection joint probabilistic data association (MD-JPDA) filter [30] which is designed to track multiple targets.
Though the JPDA, MD-PDA, and MD-JPDA filters perform better than the PDA filter for challenging tracking problems, they are all combinatoric in nature. As a result, the computation times of these algorithms scale exponentially with the number of targets and measurements.

An alternative formulation of the problem is to treat each group of measurements surrounding a target as a cluster, borrowing methodologies from the statistics community. The objective is then to track the center of the cluster within a filtering framework. Expectation maximization (EM) is a well-known algorithm often used to find the centers of multiple Gaussian-probability clusters and can be adapted to work well within a filtering framework. This accounts for multiple targets and multiple detections per target.

The probabilistic multiple hypothesis tracker (PMHT) [31] is a well-constructed formulation of the expectation maximization algorithm within a tracking framework. The algorithm is a batch optimization algorithm which treats each cluster of detections as a measurement input to a fixed-interval Kalman smoother [32]. The algorithm begins with an initial guess of each target location and goes back and forth between optimizing the cluster centers and the target states until convergence.

However, the probabilistic multiple hypothesis tracker (PMHT) is a batch algorithm, requiring the complete sequence of detections. Our algorithm is similar to the PMHT, but does not require Kalman smoothing and only performs a single iteration of expectation maximization to estimate the center of each measurement cluster. This simplification allows the algorithm to track multiple objects in real-time on-board a UAV.

3.2 KLT-Based Motion Detection with Stopped Object Tracking as a Measurement Source

The measurement source used in this paper for video tracking is KLT-based motion detection with stopped object tracking [18]. Good features to track [24] are first detected in the current frame and then matched to the previous frame using optical flow. A homography is then fit to these points to describe the motion between the previous and current frames. Outliers to the homography are classified as moving points.

However, using motion detection alone does not allow stopped objects to be tracked. Thus moving points from previous frames are also tracked to subsequent frames using optical flow.
When an object’s velocity and the number of moving object detections on the target decreases, these long-term tracked points fill in the missing detections using a spatial prioritization algorithm.

3.3 Recursive RANSAC

Many target tracking algorithms, such as the PDA and JPDA filters, assume that the number of targets is known before-hand. This may be true in simulation, but is not true when the tracker is used on a UAV or when the tracker is used on recorded videos that have not been annotated. Thus an automatic track initialization algorithm is necessary.

Recursive-RANSAC is a multiple-target tracking algorithm with automatic track initialization. At each time step new models are searched for using RANSAC. When a sufficient number of inliers are detected a new track is created. Since cardinality errors are corrected by recursive-RANSAC, existing tracks can be propagated forward using the probabilistic data association filter [25], resulting in high quality tracking.

3.4 Probabilistic Data Association Filter

The probabilistic data association filter [25] calculates a weighted average from the probabilistic density function of the Gaussian measurement random variable. The covariance of the state uncertainty combined with the measurement uncertainty is

$$S = R + CPC^\top. \quad (3.1)$$

The corresponding Gaussian PDF is

$$f(z_i(k); \tilde{z}(k|k-1), S(k)) = \frac{1}{\sqrt{(2\pi)^d \det(S)}} e^{-\frac{1}{2}(z_i - \tilde{z})^\top S^{-1}(z_i - \tilde{z})}. \quad (3.2)$$

From these are calculated likelihoods that each measurement is correct, giving

$$\mathcal{L}_i(k) \triangleq \mathcal{N}(z_i(k); \tilde{z}(k|k-1), S(k)) \frac{P_D}{\lambda}, \quad (3.3)$$
where $P_D$ is the target detection probability, $P_G$ is the gate probability, and $\lambda$ is the clutter density.

Equation (3.3) essentially compares the Gaussian density function for the correct measurement to the clutter density. These likelihoods, along with the null hypothesis, are then normalized so they sum to 1. This gives

$$
\beta_i(k) = \begin{cases} 
\frac{\mathcal{L}_i(k)}{1 - P_D P_G + \sum_{j=1}^{m(k)} \mathcal{L}_i(k)}, & i = 1, \ldots, m(k) \\
\frac{1 - P_D P_G}{1 - P_D P_G + \sum_{j=1}^{m(k)} \mathcal{L}_i(k)}, & i = 0,
\end{cases}
$$

where $i = 0$ corresponds to the possibility that none of the measurements are correct.

The state update equation for the PDA filter is

$$
\hat{x}(k|k) = \hat{x}(k|k-1) + W(k) v(k),
$$

where the combined innovation is

$$
v(k) = \sum_{i=1}^{m(k)} \beta_i(k) v_i(k),
$$

and the gain $W(k)$ is the same as the Kalman filter. The covariance update equation for the PDA filter is

$$
P(k|k) = \beta_0(k) P(k|k-1) + [1 - \beta_0(k)] P^c(k|k) + \tilde{P}(k),
$$

where the covariance of the state updated with the correct measurement is

$$
P^c(k|k) = P(k|k-1) - W(k) S(k) W(k)^T,
$$

and the spread of innovation terms is

$$
\tilde{P}(k) \triangleq W(k) \left[ \sum_{i=1}^{m(k)} \beta_0(k) v_i(k) v_i(k)^T - v_i(k) v_i(k)^T \right] W(k)^T.
$$
3.5 Expectation Maximization

Expectation maximization is a technique often used to find the mean, covariance, and weights of multiple Gaussian clusters. Let $z_i$ be the $i$th measurement in the set of $n$ received measurements, and let $\mu_j$, $\Sigma_j$, and $\tau_j$ be the mean, covariance, and weight of the $j$th Gaussian cluster. Additionally, let the sum of the cluster weights $\sum_{j=1}^{N} \tau_j = 1$.

The likelihood of receiving the given measurements $z_1, z_2, \cdots, z_n$, given the cluster parameters $\theta \equiv \{ \mu_j, \Sigma_j, \tau_j \}$ for $j = 1, 2, \cdots, n$ is

$$L(\theta ; z) = \prod_{i=1}^{n} \sum_{j=1}^{m} \tau_j f_j(z_i),$$

where $f_j(x_i)$ is the probability density function of a multivariate Gaussian given by

$$f_j(x_i) = \frac{1}{\sqrt{(2\pi)^d |\Sigma|}} e^{-\frac{1}{2} (z_i - \mu_j)^\top \Sigma_j^{-1} (z_i - \mu_j)}.$$

Taking the log of the likelihood function reduces the product to a sum

$$\log (L(\theta ; z)) = \log \left( \prod_{i=1}^{n} \sum_{j=1}^{m} \tau_j f_j(z_i) \right)$$

$$= \sum_{i=1}^{n} \log \left( \sum_{j=1}^{m} \tau_j f_j(z_i) \right). \quad (3.6)$$

However, this does not simplify the inner sum. If the inner sum were a product it could be easily simplified using the same procedure. But we can manipulate the equation a bit to pretend that the sum is outside of the logarithm by adding a variable $T_{i,j}$ which we will call the membership probability, in other words, the probability that measurement $i$ belongs to cluster $j$. Calculating these membership probabilities is known as the E step in expectation maximization. This gives

$$T_{i,j} = \frac{\tau_j f_j(z_i)}{\sum_{k=1}^{m} \tau_k f_k(z_i)}, \quad (3.7)$$
which can also be rewritten as 
\[
\sum_{k=1}^{m} \tau_k f_k(z_i) = \frac{\tau_j f_j(z_i)}{T_{i,j}}. \tag{3.8}
\]
Substituting (3.8) into (3.6) gives
\[
\log(L(\theta; z)) = \sum_{i=1}^{n} \log \left( \frac{\tau_j f_j(z_i)}{T_{i,j}} \right).
\]
The summation is now hidden within the membership probability \( T_{i,j} \). Notice that the variable \( j \) is no longer inside of a summation and so now it could be any cluster and the equation would still hold.

Now we’ll add another summation outside the logarithm using the fact that \( \sum_{j=1}^{m} T_{i,j} = 1 \). This gives
\[
\log(L(\theta; x)) = \sum_{i=1}^{n} \sum_{j=1}^{m} T_{i,j} \log \left( \frac{\tau_j f_j(x_i)}{T_{i,j}} \right). \tag{3.9}
\]
Careful comparison shows that equation (3.9) and (3.6) are equivalent. However, the equation is still not linear because \( f_j(z_i) \) is directly correlated to \( T_{i,j} \). We can’t necessarily optimize \( f_j(z_i) \) independently because we would also be changing \( T_{i,j} \). However, we can pretend that \( T_{i,j} \) is constant and optimize the rest of the equation. To avoid confusion we will denote this constant as \( T_{i,j}^{(k)} \), where \( k \) represents the \( k \)th iteration. It seems that there is a danger that optimizing \( f_j(z_i) \) could improve the estimate temporarily, but then after calculating \( T_{i,j}^{(k+1)} \) the estimate could get worse, thus resulting in an inefficient algorithm. However, we can use Jensen’s inequality to prove that the real cost function will never be worse than our approximation of it.

Jensen’s inequality states
\[
\text{if } \sum_{i} \lambda_i = 1,
\]
then
\[
\log \sum_{i} \lambda_i Q_i \geq \sum_{i} \lambda_i \log Q_i.
\]
This can be used to show that

\[ \sum_{i=1}^{n} \log \left( \sum_{j=1}^{m} T_{i,j}^{(k)} \frac{T_{j}(z_{i})}{T_{i,j}^{(k)}} \right) \geq \sum_{i=1}^{n} \sum_{j=1}^{m} T_{i,j}^{(k)} \log \left( \frac{T_{j}(z_{i})}{T_{i,j}^{(k)}} \right). \]

We can maximize equation (3.9), replacing \( T_{i,j} \) with \( T_{i,j}^{(k)} \), giving

\[
\log(L(\theta;x)) \geq \sum_{i=1}^{n} \sum_{j=1}^{m} T_{i,j}^{(k)} \log \left( \frac{T_{j}(z_{i})}{T_{i,j}^{(k)}} \right) \\
= \sum_{i,j} T_{i,j}^{(k)} \left( \log \tau_{j} + \log \left( f_{j}(z_{i}) \right) - \log T_{i,j}^{(k)} \right) \\
= \sum_{i,j} T_{i,j}^{(k)} \left( \log \tau_{j} - \frac{d}{2} \log(2\pi) - \log |\Sigma| - \frac{1}{2} (z_{i} - \mu_{j})^{\top} \Sigma^{-1} (z_{i} - \mu_{j}) - \log T_{i,j}^{(k)} \right).
\]

Each section of the equation can now be optimized independently. Optimizing over \( \tau_{j} \) gives

\[
\tau_{j}^{(k+1)} = \frac{1}{n} \sum_{i=1}^{n} T_{i,j}^{(k)}.
\]

Optimizing over the mean \( \mu_{j} \) and covariance \( \Sigma_{j} \) gives

\[
\mu_{j}^{(k+1)} = \frac{\sum_{i=1}^{n} T_{i,j}^{(k)} z_{i}}{\sum_{i=1}^{n} T_{i,j}^{(k)}},
\]

and

\[
\Sigma_{j}^{(k+1)} = \frac{\sum_{i=1}^{n} T_{i,j}^{(k)} (z_{i} - \mu_{j}^{(k)}) (z_{i} - \mu_{j}^{(k)})^{\top}}{\sum_{i=1}^{n} T_{i,j}^{(k)}}.
\]

Calculating the mean and covariance of each of the clusters is known as the M step in expectation maximization.
3.6 Incorporating Membership Probabilities into the Probabilistic Data Association Filter

The membership probabilities given in Equation (3.7) can be used as weights instead of using the raw Gaussian probabilities in the probabilistic data association filter. The same combined state and measurement uncertainty given in (3.1) is used. The corresponding Gaussian PDF in measurement space is also identical to the PDA filter and is given in (3.2). From these are calculated membership probabilities. Equation (3.7) is adapted to include a uniform random variable to represent the clutter. For simplicity it is assumed that each weights $t_j$ of each Gaussian cluster are identical and the clutter density $\lambda$ is known before-hand. This gives

$$T_{i,j} = \begin{cases} \frac{f_j(z_i)}{\lambda + \sum_{k=1}^{m} f_k(z_i)} & i = 1, \ldots, n \\ \frac{\lambda}{\lambda + \sum_{k=1}^{m} f_k(z_i)} & i = 0. \end{cases}$$

The case $i = 0$ corresponds to the possibility that the measurement is clutter. This equation compares the Gaussian density function to the clutter density, but focuses on each measurement instead of each target. The membership probabilities always sum to one for each measurement, but to be used in the probabilistic data association filter, they and the null hypothesis must be normalized so they sum to 1 for each target. This gives

$$\beta_{i,j}(k) = \begin{cases} \frac{T_{i,j}}{1 - P_D P_G + \sum_{k=1}^{m} T_{k,j}} : & i = 1, \ldots, n \\ \frac{1 - P_D P_G}{1 - P_D P_G + \sum_{k=1}^{m} T_{k,j}} : & i = 0. \end{cases}$$

The state and covariance update equations are the same as the PDA filter and are given in equations (3.4) and (3.5).

Figure 3.2 graphically summarizes the difference between the weights used in the PDA and MPDA filters. The PDA filter considers which detection is most likely to have originated from the target. It uses the Gaussian probabilities as weights for each of the measurements. On the other hand, the MPDA filter considers which target is mostly to have produced each detection, by comparing the relative PDFs of each Gaussian distribution. It uses the membership probabilities as weights, which are the Gaussian probabilities normalized so they sum to one.
3.7 Nearly Constant Velocity Target Model

For the results in this paper, the targets are modeled using a nearly constant velocity (NCV) Kalman filter, where the measurement input is generated using the membership probabilistic data association algorithm described above. The states of this Kalman filter are

\[
x = \begin{bmatrix} p_x \\ p_y \\ v_x \\ v_y \end{bmatrix},
\]

where \( p_x \) and \( p_y \) are the target’s \( x \) and \( y \) position measured in pixels, and \( v_x \) and \( v_y \) are the target’s \( x \) and \( y \) velocity measured in pixels per frame. The target propagates according to the equation

\[
x_{k+1} = \begin{bmatrix} 1 & 0 & \delta_k & 0 \\ 0 & 1 & 0 & \delta_k \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} x_k + w,
\]

where \( w \) is the process noise and \( \delta_k \) is the duration of the \( k \)th discrete time step.
At each time step $k$, position and velocity measurements are received giving the measurement vector

$$y_k = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} x_k + v,$$

where $v$ is the measurement noise.

The process noise is distributed according to the process noise matrix

$$Q = q \begin{bmatrix} \frac{\delta_k^3}{3} & 0 & \frac{\delta_k^2}{2} & 0 \\ 0 & \frac{\delta_k^3}{3} & 0 & \frac{\delta_k^2}{2} \\ \frac{\delta_k^2}{2} & 0 & \delta_k & 0 \\ 0 & \frac{\delta_k^2}{2} & 0 & \delta_k \end{bmatrix},$$

where $q$ is a tuning parameter. The measurement noise is distributed according to the sensor noise matrix

$$R = \begin{bmatrix} \sigma_p^2 & 0 & 0 & 0 \\ 0 & \sigma_p^2 & 0 & 0 \\ 0 & 0 & \sigma_v^2 & 0 \\ 0 & 0 & 0 & \sigma_v^2 \end{bmatrix},$$

where $\sigma_p$ is the standard deviation of the position measurements and $\sigma_v$ is the standard deviation of the velocity measurements. The same sensor noise matrix is used for both the Kalman filter and data association algorithms.

### 3.8 Results

The PDA, JPDA, and MPDA filters were tested on a video sequence taken from the top of a building on Brigham Young University campus. The 5-minute video sequence includes a large number of targets crossing paths and complex target interactions. The same hand-tuned parameters were used for all three filters, with $\delta_k = 1$, $\sigma_p = 10$, $\sigma_v = 5$, and $q = 0.11$, as described
Table 3.1: Tracker MOTA and MOTP scores

<table>
<thead>
<tr>
<th>Version</th>
<th>Rcll ↑</th>
<th>Prcn ↑</th>
<th>FP ↓</th>
<th>FN ↓</th>
<th>MOTA ↑</th>
<th>MOTP ↓</th>
</tr>
</thead>
<tbody>
<tr>
<td>PDA</td>
<td>55.7%</td>
<td>81.6%</td>
<td>17049</td>
<td>60015</td>
<td>43.1%</td>
<td>11.06</td>
</tr>
<tr>
<td>JPDA</td>
<td>57.0%</td>
<td>88.5%</td>
<td>10069</td>
<td>58237</td>
<td>49.5%</td>
<td>10.87</td>
</tr>
<tr>
<td>MPDA (Ours)</td>
<td>54.4%</td>
<td>95.2%</td>
<td>3756</td>
<td>61744</td>
<td>51.6%</td>
<td>8.33</td>
</tr>
</tbody>
</table>

in Section 3.7. First, the filters were benchmarked using these hand-tuned parameters and the video results analyzed. The video results are available at https://youtu.be/E1HeI6gxJ5Q. Second, parameter sweeps were also performed by sweeping one parameter at a time, while keeping the other parameters constant.

3.8.1 Video Result Analysis with Hand-Tuned Parameters

The PDA, JPDA, and MPDA filters were benchmarked using the CLEAR MOT scores [33], using py-motmetrics [34]. The CLEAR multiple object tracking accuracy (MOTA) is a measure of track continuity. In this metric, track switches, false positives, and false negatives are penalized. This metric is computed by finding the optimal target-to-track assignment. The better the track continuity, the higher the tracking accuracy will be.

On the other hand, the multiple object tracking precision (MOTP) metric shows the ability of the tracker to estimate precise object locations, regardless of whether the trajectories are consistent. The closer the target estimates are to the true location, the lower the MOTP score will be. Note that in this paper, we use the original MOTP definition, which defines the MOTP score as an error metric where a lower error is better, while the MOT challenge has inverted the metric so that a higher MOTP score is better.

The CLEAR MOT metric results are shown in Table 3.1. In comparison to the original PDA filter, the JPDA filter gives a significant increase in accuracy. However, it only gives a small improvement in precision. On the other hand, the MPDA filter gives significant improvements in both the accuracy and precision of the tracker.

The computation times of the filtering algorithms are shown in Table 3.2. The algorithms were run on a Linux laptop with a 2.1 GHz Intel i7 CPU and 8 GB of memory. Both the PDA and MPDA filters have comparable computation times. However, the JPDA algorithm requires two
Table 3.2: Computation times

<table>
<thead>
<tr>
<th>Filtering Algorithm</th>
<th>Average time per frame</th>
<th>Maximum time per frame</th>
</tr>
</thead>
<tbody>
<tr>
<td>PDA</td>
<td>1.3 ms</td>
<td>6 ms</td>
</tr>
<tr>
<td>JPDA</td>
<td>126 ms</td>
<td>15 sec</td>
</tr>
<tr>
<td>MPDA (Ours)</td>
<td>1.1 ms</td>
<td>7.3 ms</td>
</tr>
</tbody>
</table>

orders of magnitude more computation time and sometimes takes an entire 15 seconds to process a single frame. This is because the JPDA filter enumerates all possible association events, which is time consuming. Note that in our JPDA implementation, to reduce memory and CPU usage we only enumerate combinations for which the target gating regions intersect. Even so, on some frames of the video we had to truncate the number of possible association events to ten million to avoid consuming all available memory on the laptop.

Figure 3.3 shows an example of two targets crossing paths. For target crossings, all three filters perform well. Figure 3.4 shows an example of two target paths that join together. The small distance between the two targets causes the PDA and JPDA filter estimates to coalesce, while the MPDA filter estimates remain separated. Figure 3.5 shows an example of two targets traveling parallel at different speeds. This scenario causes a track switch for the PDA and JPDA filters, but the MPDA filter tracks both targets correctly. Lastly, Figure 3.6 shows an example where the MPDA filter converges quickly to the center of a target, but the PDA and JPDA filter estimates fail to converge.

3.8.2 Parameter Sweeps

Three parameter sweeps were also performed using the same video sequence. For the parameter sweeps, an idealistic set of target detections from the video sequence was generated. These same detections were used for each filter run in the parameter sweeps. In each parameter sweep, the MOTA and MOTP scores were calculated for both the MPDA and PDA filters. Due to the large time requirement of the JPDA filter, the JPDA filter was not included in the parameter sweeps.

The same hand-tuned parameters described in the previous section were used for the parameter sweeps, with \( \delta_k = 1, \sigma_p = 10, \sigma_v = 5, \) and \( q = 0.11, \) with one of the values modified.
Figure 3.3: Example of two crossing targets. The PDA (blue), JPDA (green), and MPDA (red) filters all perform well in this case.

at a time. Note that for the sensor noise parameter sweeps, only the data association sensor noise was modified, while the Kalman filter sensor noise was kept constant. This allows the effects on the data association to be isolated.

Figure 3.7 shows the effect of changing the position component of the data association sensor noise matrix. The position is important when distinguishing between interacting targets, as shown by the sharp drop in tracking accuracy if $\sigma_p$ is too large. The MPDA filter has significantly better tracking accuracy and precision for all values of $\sigma_p$. 

33
Figure 3.4: Example of tracks coalescing. In this part of the video a couple meet and walk together. The PDA filter estimates (blue) coalesce. The JPDA filter (green) performs slightly better. The MPDA filter (red) is the only filter that avoids coalescence completely.
Figure 3.5: Example of parallel traveling targets. In this part of the video two pedestrians are traveling in the same direction, but with different velocities, so that one of the pedestrians passes the other. The PDA filter estimates (blue) and the JPDA filter estimates (green) are pulled away from the target center by the other target, which causes a track switch. The MPDA filter avoids this issue and successfully tracks both targets without a track switch.
Figure 3.6: Example of lack of convergence. All three filters are initialized with error. The MPDA filter (red) converges to the center of the target, while the PDA filter (blue) and the JPDA filter (green) do not.

Figure 3.8 shows the effect of changing the velocity component of the data association sensor noise matrix. Notice that the PDA and MPDA filters are not as sensitive to the value of $\sigma_v$ as they are to the value of $\sigma_p$. There is a small decrease in accuracy for large values of $\sigma_v$, suggesting that the velocity is still important for distinguishing between interacting targets. The MPDA filter has better MOTA and MOTP scores for all values of $\sigma_v$.

Figure 3.9 shows the effect of changing the process noise of the Kalman filter. Notice how the PDA filter has better tracking accuracy but worse tracking precision for large values of $q$. On the other hand, this trade-off does not seem to affect the MPDA filter. The MPDA filter tracking
accuracy is less sensitive to the value of $q$ which allows better accuracy and precision objectives to be met at the same time.

![Figure 3.7: Position sensor noise parameter sweep. The PDA and MPDA filters have good tracking accuracy and precision near $\sigma_p = 10$, indicating that the measurement position is important for data association. The MPDA filter has better tracking accuracy and precision for all values of $\sigma_p$.](image)

3.9 Conclusion

In this paper we have presented a membership probabilistic data association (MPDA) filter. We have shown that this filter outperforms the PDA and JPDA filters for complex target interactions. In comparison to the PDA and JPDA filters, the MPDA filter is more likely to avoid track switches, reduces track coalescence, and converges faster to the center of targets. It also is less sensitive to parameter tuning and is not affected by the trade-off between accuracy
Figure 3.8: Velocity sensor noise parameter sweep. The PDA and MPDA filters are not as sensitive to the value of $\sigma_v$ as they are to the value of $\sigma_p$. The MPDA filter has better tracking accuracy and precision for all values of $\sigma_v$.

and precision. Future work includes using methodologies from the statistics community such as Dirichlet processes to estimate the number of measurement clusters in order to improve model merging and splitting.
Figure 3.9: Process noise parameter sweep. The PDA filter has better accuracy for large values of $q$, but lower precision. The tracking accuracy of the MPDA filter is not sensitive to $q$ and performs better for all values of $q$. 
4.1 Introduction

There are many applications where an unmanned aerial vehicle (UAV) is required to visually track a ground-based target using an on-board camera. In these applications, successful target tracking often requires that past estimates of the target be transformed into the current image frame so that moving targets can be correctly identified [17]. If the target is moving on an approximately planar surface, then the coordinate transformation can be accomplished using the homography matrix.

Figure 4.1 shows the relevant geometry. Suppose that $P$ is the target of interest on the ground and that $P$ lies on a planar surface with unit normal vector $\hat{n}$, where $\hat{n}$ points into the earth. Furthermore, suppose that the UAV captures two images of $P$ at time indices $k - 1$ and $k$. The coordinate frame of the camera at time index $k - 1$ is denoted $F_{k-1}$ and the coordinate frame of the camera at time index $k$ is denoted $F_k$. Let $P_{k-1}$ be the target’s position at expressed in frame $F_{k-1}$, and let $P_k$ be the target’s position expressed in frame $F_k$. Let $d_{k-1}$ be the perpendicular distance from the camera to the ground plane at time $k - 1$, and let $d_k$ be the distance from the camera to the ground plane at time $k$. The relative pose between the camera frames $F_{k-1}$ and $F_k$ is given by the vector $t_{k-1/k}$ from $F_{k-1}$ to $F_k$ and the rotation matrix $R_{k-1/k}$ that transforms vectors expressed in $F_{k-1}$ to vectors expressed in $F_k$. We will use a superscript to denote the frame in which a vector is expressed. Therefore $\hat{n}^{k-1} \in \mathbb{R}^3$ is $\hat{n}$ expressed relative to the basis vectors of $F_{k-1}$.

With reference to Figure 4.1, we have

$$P_k = R_{k-1/k} P_{k-1} + t_{k-1/k}.$$
Figure 4.1: The geometry for the derivation of the homography matrix between two camera poses

Observing that

\[ d_{k-1} = (\hat{n}^{k-1})^\top p^{k-1}, \]

we get that

\[
p^k = R^k_{k-1} p^{k-1} + t^k_{k-1/k} \left( \frac{(\hat{n}^{k-1})^\top p^{k-1}}{d_{k-1}} \right) \\
= \left( R^k_{k-1} + \frac{t^k_{k-1/k}}{d_{k-1}} (\hat{n}^{k-1})^\top \right) p^{k-1}. \tag{4.1} \]

Let \( p^k \) represent projection of \( P^k \) onto the normalized image plane of camera \( k \). In other words \( p^k \) is the location of \( P \) in the image plane of camera frame \( \mathcal{F}^k \). If \( P^k = \begin{bmatrix} x^k & y^k & z^k \end{bmatrix}^\top \), then

\[
p^k = \begin{bmatrix} \frac{x^k}{2^k} \\
\frac{y^k}{2^k} \\
1 \end{bmatrix}.\]
Therefore, Equation (4.1) can be written as

\[
\begin{pmatrix}
Z^k \\
Z^{k-1}
\end{pmatrix} p_k = 
\left( \begin{pmatrix}
R_{k-1}^k + \frac{t_{k-1}^k (\hat{n}^{k-1})^\top}{d_{k-1}}
\end{pmatrix}
\right) p_{k-1}.
\]

Defining the scalar \( \gamma_k \triangleq \frac{Z^k}{Z^{k-1}} \) we obtain

\[
\gamma_k p_k = \left( \begin{pmatrix}
R_{k-1}^k + \frac{t_{k-1}^k (\hat{n}^{k-1})^\top}{d_{k-1}}
\end{pmatrix}
\right) p_{k-1}.
\] (4.2)

The matrix

\[
H_k \triangleq \left( \begin{pmatrix}
R_{k-1}^k + \frac{t_{k-1}^k (\hat{n}^{k-1})^\top}{d_{k-1}}
\end{pmatrix}
\right)
\] (4.3)
is called the Euclidean homography matrix at time \( t_k \) [35]. Equation (4.2) demonstrates that the homography matrix \( H_k \) transforms the pixel location of \( P \) at time index \( k-1 \) into the pixel location of \( P \) at time index \( k \). The homography is defined up to a scale factor \( \gamma_k \); any scaling of the matrix results in an equivalent matrix. In practice, the scale factor \( \gamma_k \) is determined as the inverse of the third element of \( H_k p_{k-1} \), so as to ensure that the third element of \( p_k \) is one.

The homography between frames can be estimated by first detecting features in one image and tracking them using optical flow or matching them to features in the next image. A homography is calculated from these feature pairs using random sample consensus (RANSAC) [36], or least median of squares (LMedS) [37]. These algorithms randomly pick 4-point subsets, calculate a homography hypothesis from each subset, score the hypotheses, and pick the best one. The hypothesis is then refined using an iterative method on only the inlier points [38]. There are readily-available open-source implementations of these algorithms, such as the `findHomography` function in OpenCV [26].

Video tracking algorithms from a UAV typically represent the target’s position \( p \), velocity \( v \), and position covariance \( \Sigma \) in the image plane. These states are updated using a Kalman filter [39], or in the case of multiple targets or uncertain measurement association, a probabilistic data association filter (PDAF) [25]. The Kalman filter prediction step advances the target state forward in time. However, after the Kalman filter prediction step, the target’s state is still represented in the previous camera frame. This state must also be transformed into the current camera frame. The
purpose of this technical note is to show how to correctly transform the image plane projection of the target’s position, velocity, and covariance from one frame to the next frame.

4.2 The Homography as a Non-Linear Transformation

While the homography is a matrix, and matrices define linear transformations, the homography is not a linear transformation due to the scale factor $\gamma$. To see this, represent the homography as a block matrix,

$$H = \begin{bmatrix} H_1 & h_2 \\ 2 \times 2 & 2 \times 1 \\ h_3 & h_4 \\ 1 \times 2 & 1 \times 1 \end{bmatrix}.$$ 

Solving for $\gamma$ gives

$$\gamma \begin{bmatrix} p^2 \\ 1 \end{bmatrix} = \begin{bmatrix} H_1 & h_2 \\ h_3 & h_4 \end{bmatrix} \begin{bmatrix} p^1 \\ 1 \end{bmatrix},$$

and

$$p_2 = \frac{H_1 p^1 + h_2}{\gamma} = \frac{H_1 p^1 + h_2}{h_3 p^1 + h_4}.$$  

To simplify notation, let us represent this transformation as the function

$$g(p_1) = \frac{H_1 p^1 + h_2}{h_3 p^1 + h_4}.$$  

This equation is equivalent to (4.2), but more clearly shows the non-linear elements of the transformation. This will be useful in subsequent sections when deriving the target velocity and covariance transformations in the image plane.

4.3 Transforming the Target’s Velocity

We will now derive a method for transforming the target’s velocity between frames. Define the target’s velocity in the image plane as the net velocity of the target, after the ego-motion of the
UAV has been subtracted. Thus according to this definition, if an object is stationary it will also have zero velocity in the image plane, regardless of the motion of the UAV.

The equation to transform the target’s position in the image plane is given in (4.5). However, transforming the target’s velocity in the image plane using this same equation would add a non-zero offset to the velocity of \( \frac{h_2}{h_3 p^1 + h_4} \) whenever the camera translates or rotates. Thus homogeneous coordinates should not be used when transforming the target’s velocity. The desired transformation for the velocity should instead be a \( 2 \times 2 \) matrix that only stretches and rotates the velocity, rather than a \( 3 \times 3 \) matrix. Similarly, the position covariance of the target is a \( 2 \times 2 \) matrix and also requires a \( 2 \times 2 \) matrix to stretch or rotate it into the next frame.

The temptation is to ignore the non-linear component of the transformation, and simply take the derivative of \( p^2 \) in (4.4), which results \( H_1 \), the upper-left \( 2 \times 2 \) sub-matrix of the homography matrix. This can produce correct results under certain special cases, as will be discussed later in the paper. However, the problem with this approach is that it ignores the scale factor \( \gamma \). For example, multiplying the homography matrix by two gives an equivalent homography, but since the submatrix is also multiplied by two, it will produce a different velocity!

An improved technique is to represent the velocity with two points and subtract the difference of the transformed points to obtain the transformed velocity. Suppose the position of the target in the first image is given by vector \( p^1 \) and its velocity by \( v^1 \). In \( \delta t \) seconds, the target will be at position \( q^1 = p^1 + \delta t v^1 \). This point can be transformed into the second frame using (4.5) to give \( q^2 \). The difference between these two points in the second frame is a good approximation of the target’s velocity in the second frame. This gives

\[
v^2 \approx \frac{g(p^1 + \delta t v^1) - g(p^1)}{\delta t}.
\]

(4.6)

The smaller the time difference used in calculating (4.6), the more accurate the approximation will be. If we take the limit as \( \delta t \) approaches 0, we obtain

\[
v^2 = \lim_{\delta t \to 0} \frac{g(p^1 + \delta t v^1) - g(p^1)}{\delta t}.
\]

(4.7)

Observe that (4.7) is the definition of a calculus derivative, in the direction \( v^2 \). Thus the correct approach to obtain an exact answer is to calculate the derivative of the homography transformation.
By differentiating (4.5) using the quotient rule we obtain

\[
G(p) = \frac{\partial}{\partial p} g(p). \\
= \frac{\partial}{\partial p} \frac{H_1 p + h_2}{h_3^\top p + h_4} \\
= \frac{(h_3^\top p + h_4) H_1 - (H_1 p + h_2) h_3^\top}{(h_3^\top p + h_4)^2}. \tag{4.8}
\]

This transforms the image plane velocity of the target to the next frame as

\[
v^2 = G(p)v^1. \tag{4.9}
\]

We can now return to the simple, but mathematically incorrect submatrix method mentioned above, to determine under what conditions it will produce the same answer as (4.9). Suppose the velocity is transformed to the next frame according to the equation

\[
v^2 = H_1 v^1. \tag{4.10}
\]

This equation will be equivalent to (4.9) if the matrices \(G\) and \(H_1\) are equal. Using the definition of \(G\) in (4.8) we find that there are two conditions that must hold for the matrices \(G\) and \(H_1\) to be equal. First, the vector \(h_3\) must be zero, and second, the scalar \(h_4\) must be one.

In order for the vector \(h_3\) to be zero, the camera’s motion must meet certain conditions. The homography matrix is the sum of a rotation matrix and a outer product from the translation vector and plane normal, as shown in (4.3). In order for the rotation-contributed term of \(h_3\) to be zero, the rotation matrix must be a rotation about the optical axis, or z axis, of the camera. Any other rotation will cause a large difference in the result. In order for the translation-contributed term of \(h_3\) to be zero, either the translation in the z direction must be zero, or the plane normal vector x and y components must be zero. In other words, translating the camera in the x and y directions is allowed, but translating the camera in the z direction is only allowed if the camera is pointed directly at the plane.

45
The second condition for (4.9) and (4.10) to be equivalent is that $h_4$ must equal 1. Since the homography is defined up to a scale factor, $h_4$ will only be 1 if the scale factor happens to be exactly the right number to make $h_4$ be 1. This can occur by coincidence, or if the library used to calculate the homography intentionally scales the homography so that $h_4$ is 1. As it turns out, OpenCV’s `findHomography` function happens to scale the homography so that $h_4$ is 1. This enables (4.9) and (4.10) to be equivalent if the conditions described above are also met.

### 4.4 Transforming the Target’s Covariance

![Diagram of target's position, velocity, and position covariance in the image plane](image)

Figure 4.2: The target’s position, velocity, and position covariance in the image plane

Suppose that the position of the target in frame $k - 1$ is represented by the random vector $X^{k-1}$, with mean $\mu^{k-1}$ and covariance $\Sigma^{k-1}$ (see Figure 4.2). Using the definition of $g(\cdot)$ given in (4.5), any realizations of the random vector $X$ can be transformed into the current frame $k$ by

$$X^k = g(X^{k-1}).$$
Using a first-order Taylor series approximation of $g$ evaluated at the mean $\mu^{k-1}$ of the random vector $X^{k-1}$ gives

$$
g(X^{k-1}) \approx g(\mu^{k-1}) + \frac{\partial}{\partial z} g(z) \bigg|_{z=\mu^{k-1}} (X^k - \mu^k)
$$

$$
\approx g(\mu^{k-1}) + G(\mu^{k-1}) (X^{k-1} - \mu^k),
$$

where $G$ is defined in (4.8). With this linear approximation of the homography transformation, we can calculate the covariance $\Sigma^k$ in the current frame, giving

$$
\Sigma^k = E \left[ (X_k - \mu_k) (X_k - \mu_k)^\top \right]
$$

$$
= E \left[ (g(X^{k-1}) - \mu_k) (g(X^{k-1}) - \mu_k)^\top \right]
$$

$$
\approx E \left[ (g(\mu^{k-1}) + G(\mu^{k-1}) (X^{k-1} - \mu^k) - \mu_k) (g(\mu^{k-1}) + G(\mu^{k-1}) (X^{k-1} - \mu^k) - \mu_k)^\top \right]
$$

$$
\approx E \left[ (G(\mu^{k-1}) (X^{k-1} - \mu^k)) (G(\mu^{k-1}) (X^{k-1} - \mu^k))^\top \right]
$$

$$
= E \left[ G(\mu^{k-1}) (X^{k-1} - \mu^k) (X^{k-1} - \mu^k)^\top G^\top (\mu^{k-1}) \right]
$$

$$
= G(\mu^{k-1}) E \left[ (X^{k-1} - \mu^k) (X^{k-1} - \mu^k)^\top \right] G^\top (\mu^{k-1})
$$

$$
= G(\mu^{k-1}) \Sigma^{k-1} G^\top (\mu^{k-1}).
$$

(4.11)

This produces a mathematically-correct approximation to the true covariance transformation. However, as mentioned before it can be tempting to use the upper-left $2 \times 2$ sub-matrix of the homography matrix

$$
\Sigma^k = H_1 \Sigma^{k-1} H_1^\top.
$$

(4.12)

This is not mathematically based and can cause the problems described in the previous section.

A simple Matlab script was used to illustrate the difference between using the covariance transforms given in Equations (4.11) and (4.12). For this plot, 20,000 points were sampled from a covariance distribution. Each of these points were then transformed to the next frame using a homography matrix generated from an out-of-the-plane y-axis rotation of $\frac{\pi}{8}$ radians (22.5 degrees).
The results are shown in Figure 4.3. The sample covariance of the points in the first frame is shown in dashed red and is nearly identical to the true distribution covariance. The sample covariance of the points in the second frame is shown in dashed blue. The expected covariance calculated using (4.11) is shown in green. This transformed covariance is very close to the sample covariance. On the other hand, the covariance calculated using (4.12) is very different from the sample covariance.

The small difference between expected distribution and the measured covariance is because the partial derivative is not constant over the entire distribution. Thus the covariance propagation is not exact. However, if the covariance is small enough, the effects of the non-linearities are reduced and the covariance after the transformation becomes very close to the true covariance. If certain applications expect large covariances, better results may be obtained with the transformations used in the unscented Kalman filter (UKF) and the particle filter (PF). These filters are much better at dealing with non-linearities. However, for our use case, the Kalman filter (KF) transformation is accurate enough.

Figure 4.3: The true covariance $\Sigma^1$ (solid red) with almost identical measured covariance (dashed red) are transformed by a y-axis rotation of $\frac{\pi}{8}$ radians (22.5 degrees). Calculating the expected covariance of the transformed points using the sub-matrix method (cyan) is much less accurate than using the derivative method (green).
4.5 Video Results

The derivative transformations given in (4.9) and (4.11), the submatrix transformations given in (4.10) and (4.12), and the two-point transformation given in (4.6) were also tested on six video sequences captured by a hand-held camera. Each video sequence shows a target which starts near the center of the field of view. Three of the video sequences show translations in each of the three unit vector directions. The other three videos show camera rotations about each of the three axes. The homography is calculated using OpenCV’s `findHomography` function. The homography is used to update the target’s position, velocity, and position covariance in the image plane. Note that because OpenCV scales the homography so that the bottom-right entry of the homography matrix is always one, it is possible to obtain identical results for certain transformations listed at the end of section 4.3. To make the results easier to see, the Kalman filter time and measurement updates were omitted. In other words, the target in the video is actually stationary, but the velocity calculations done on the video sequence are still valid.

The results are shown in Figure 4.4 and at https://youtu.be/ixyq1qa9m70. The position and velocity are shown by arrows and the covariance is shown by an ellipse representing the 1-sigma bound. The first frame of each video sequence is shown on the left and the last frame is shown on the right. The derivative, submatrix, and two-point methods are shown in green, cyan, and red respectively. Notice that all three methods give similar results when the translation is in the x, y, and z directions. As mentioned earlier, the z direction translation can give similar results if the camera is pointed at the plane. All three methods also give similar results for rotations about the camera’s z axis. However, the three methods give different results for rotations about the x and y axes. The derivative and two-point give good results, but the submatrix method has large unexpected stretching and shrinking of the velocity and covariance estimates as the camera rotates.

4.6 Matlab UAV Simulation Results

A Matlab UAV simulation was used to test the position, velocity, and covariance transformations defined above, within a Kalman filter target tracker. In this simulation, a target is traveling at a constant velocity in the world frame, while being followed by a UAV. A gimbaled camera is mounted on the UAV and rotates to keep the target in the center of the field of view. In
addition to keeping the target in the center of the field of view, the camera is also rotating slowly about its optical axis at an approximately constant angular rate. Both the UAV and gimbal are controlled using simple PID controllers. However, the UAV gains are not well tuned, resulting in significant oscillations. Figure 4.5 shows the UAV represented with a red asterisk and the target’s position represented with a blue asterisk. Figure 4.6 shows the world position of the UAV and the target.

At each time step, the camera returns the $x$ and $y$ pixel locations of the target in the image plane. The position and velocity of the target are updated using a Kalman filter. However, at each time step after the Kalman filter time update, the position, velocity, and covariance of the target must be transformed into the next camera frame. All three methods transform the target’s
position into the next frame using (4.5), but apply different transformations to the target’s velocity and covariance. The resulting estimates and their error are shown in Figure 4.7. For a baseline comparison, no transformation is performed on the velocity and covariance (blue). This results in significant error as the camera rotates about its optical axis. The second comparison is the submatrix method, which transforms the velocity and covariance using (4.10) and (4.12). This transformation is fairly accurate when the UAV is at a constant attitude. However, when the UAV tips forward or backward, there are large spikes in the error.

The third comparison uses the derivative of the homography transformation. This method transforms the velocity and covariance using (4.9) and (4.11). This method is the most accurate. However, there is still error because the target’s state is represented as a constant-velocity target in the image plane. Constant velocity in the world plane is not constant velocity in the image plane. For example, imagine a camera pointed at the horizon as a motorcycle drives past. The motorcycle may be moving at a constant velocity, but its velocity in the image plane gets smaller and smaller as its distance from the camera increases.

The last comparison also uses the derivative of homography transformation to transform the velocity and covariance as shown in (4.9) and (4.11). However, instead of performing the Kalman filter time update in the image plane, the states are temporarily transformed into the world ground-plane to perform the Kalman filter update. This transformation is also accomplished using a homography. This method has no error for the entire simulation because the correct homography state transformation is used and the target model is exact.

### 4.7 Conclusion

In this work we have derived a method of transforming target states represented in the image plane between consecutive frames of a video sequence. We have demonstrated the accuracy of this state transformation on both a hand-held video sequence and in a Matlab target tracking simulation. We have also shown the potential pitfalls when transforming states between frames using the homography matrix and how these pitfalls can affect tracking performance. Future work includes developing improved models to estimate target depth from a monocular camera in order to track in 3D without the flat-earth assumption.
Figure 4.5: Matlab UAV simulation
Figure 4.6: UAV and target world position over time
Figure 4.7: The target’s position and velocity in the camera frame in pixels, estimated using a Kalman filter
CHAPTER 5.  AN ITERATIVE FIVE-POINT ALGORITHM WITH APPLICATION TO MULTI-TARGET TRACKING

5.1 Introduction

Motion estimation from a video sequence has many applications in robotics including target tracking, visual odometry, and 3D scene reconstruction. These applications often require on-board processing of the video sequence in real-time and have size, weight, and power (SWAP) constraints. Furthermore, the motion estimation algorithms often must share computational resources with other computer vision algorithms on the robot.

One method of estimating motion from a video sequence is by calculating the essential matrix between consecutive frames. The essential matrix relates the normalized image plane coordinates in one frame to the next frame using the epipolar constraint. The essential matrix can be decomposed into a rotation and a normalized translation to determine the motion of the camera. The essential matrix is typically calculated by generating a large number of hypotheses from five-point minimum subsets of matching features in order to be robust to noise and outliers. This is often done using random sample consensus (RANSAC) [36] or least median of squares (LMedS) [37]. When using RANSAC, these hypotheses are scored by counting the number of inlier points from the entire set. When using LMedS, these hypotheses are scored by calculating the median error.

Previous methods calculate essential matrix hypotheses directly from each five-point minimum subset. One of the most well-known of these algorithms is Nister’s algorithm [40]. Nister showed that for five matching points, the essential matrix can have up to ten solutions, all of which can be found by solving a tenth-order polynomial, where some of the roots are complex and can be discarded. Since the number of complex roots come in pairs, there always remain an even number of real roots. On average there are typically four valid solutions. There are a number of open-source implementation of Nister’s five-point algorithm. One of the most popular implementations
is OpenCV’s `findEssentialMat` function [26]. However, constructing, solving, and extracting the essential matrix from this tenth-order polynomial is complex and can be computationally expensive. Furthermore, since each minimum subset produces up to ten hypotheses, it can be time consuming to score them.

As an alternative to directly calculating essential matrix solutions, some authors [41]–[45] propose solving for the essential matrix using non-linear optimization algorithms such as Gauss-Newton (GN) and Levenberg-Marquardt (LM). Since the essential matrix has nine entries but only five degrees of freedom, the optimization is done on the essential matrix manifold to ensure that the result is valid. There are a number of ways to define the essential matrix manifold. Some authors define the manifold using a rotation and translation unit vector, which are elements of $SO(3)$ and $S^2$ respectively [41], [42]. Others define the manifold using two elements of $SO(3)$, in an SVD-like product [44], [45].

During each iteration of the optimization algorithm, the optimizer step is solved for in terms of the five degrees of freedom along the manifold. Since $SO(3)$ and $S^2$ are or can be represented using Lie groups, the matrix exponential can then be used to map the optimizer step back to the manifold in order to perform the update. Each iteration of the optimization is quite fast because the cost function and its derivatives are mathematically simple. The most time-consuming part of the iteration is inverting a $5 \times 5$ matrix, which is faster than solving a tenth-order polynomial. For this reason, these optimization-based solvers have the potential to be faster than direct essential matrix solvers.

However, one weakness of these optimization-based solvers is that they only find one of the ten possible solutions to the essential matrix at a time. Finding all solutions requires additional optimization runs with different initialization points. The optimization method is also sensitive to initial conditions, which can cause the optimizer to fail to produce a valid solution. For example, GN may diverge if the initial guess is too far from the true solution. LM can be used to prevent increases in the cost function, but may occasionally still fail to converge. Because of the need to run the optimizer more times to produce the same number of hypotheses, these existing optimization-based solvers might not necessarily be faster than the direct essential matrix solvers if the same level of accuracy is desired.
However, not all of the ten possible solutions are needed in order achieve comparable accuracy to direct essential matrix solvers, if the best solution can be found the first time. The main contribution of this work is a novel optimization-based relative pose solver that leverages prior information in order to find the desired solution without requiring additional optimization runs. Building on previous work, our algorithm utilizes LMedS to produce robust estimates in the presence of noise and outliers. We use the LM optimizer to solve for the relative pose from each minimum subset.

The main difference between our work and previous work is that during each iteration of LMedS, we seed the optimizer with an informed initial guess instead of a random seed. In video sequences, the rotation and translation between consecutive frames is similar to nearby frames, so we can use the relative pose estimate from the previous time step as a good initial guess of the relative pose at the current time step. Then in subsequent LMedS iterations, we update this initial guess whenever a better relative pose hypothesis is found. We show that this change significantly reduces the number of hypotheses that must be generated and scored to achieve the same level of accuracy. This allows the algorithm to run in real-time while sharing resources with other computer vision algorithms.

After the best LMedS hypothesis is found, it is refined using only inlier points. The same optimizer described above can easily be extended to accept more than five points. In contrast, direct essential matrix solvers are unable to refine the best hypothesis and thus require a separate algorithm to do so.

The remainder of the paper is outlined as follows. Section 2 finalizes the problem description. Section 3 develops our iterative five-point algorithm. Section 4 applies the algorithm to target tracking in the presence of parallax. Section 5 presents results of both the iterative five-point algorithm and the target tracking algorithm. Conclusions are given in section 6.

5.2 Problem Definition

Given two consecutive video frames with point correspondences detected in each frame, the goal is to find the rotation $R^2_1$ and translation $t^2_{1/2}$ between the two frames. The relevant geometry is shown below in Figure 5.1.
Figure 5.1: The geometry for the derivation of the essential matrix

Let $P^1_i = [X^1_i \ Y^1_i \ Z^1_i]$ and $P^2_i = [X^2_i \ Y^2_i \ Z^2_i]$ be the 3D position of point $i$ with respect to camera frames 1 and 2. The equation describing the relationship between the 3D coordinates of each point in frame 1 and 2 is

$$P^2_i = R^2_1 P^1_i + t^2_{1/2},$$

where $R^2_1 \in SO(3)$ is the rotation matrix from frame 1 to 2 and $t^2_{1/2}$ is the translation vector from frame 2 to frame 1, resolved in frame 2.

Left multiplying each side of the equation by $\left( t^2_{1/2} \right)_x$, where

$$\begin{bmatrix}
  x \\
  y \\
  z
\end{bmatrix} = \begin{bmatrix}
  0 & -z & y \\
  z & 0 & -x \\
  -y & x & 0
\end{bmatrix},$$

58
gives
\[
\left( t_{1/2}^2 \right)_\times P_i^2 = \left( t_{1/2}^2 \right)_\times R_1^2 P_i^1 + \left( t_{1/2}^2 \right)_\times t_{1/2}^2.
\]

The last term on the right is zero, thus
\[
\left( t_{1/2}^2 \right)_\times P_i^2 = \left( t_{1/2}^2 \right)_\times R_1^2 P_i^1.
\]

We can then left-multiply each side of the equation by \((P_i^2)^\top\) to get
\[
(P_i^2)^\top \left( t_{1/2}^2 \right)_\times P_i^2 = (P_i^2)^\top \left( t_{1/2}^2 \right)_\times R_1^2 P_i^1.
\]

However, the left side of the equation is always zero because the cross product \(\left( t_{1/2}^2 \right)_\times P_i^2\) gives a vector perpendicular to \(P_i^2\), and therefore
\[
(P_i^2)^\top \left( t_{1/2}^2 \right)_\times R_1^2 P_i^1 = 0.
\]

Since the right side of the equation is zero, any scalar multiple of \(P_i^1\) and \(P_i^2\) will also satisfy the equation. This removes the depth dependency and allows us to use normalized image plane coordinates in the equation. Let \(p_i^1\) and \(p_i^2\) be the normalized homogeneous image coordinates of point \(i\) in camera frame 1 and 2 respectively, i.e.
\[
p_i = \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix} = \begin{bmatrix} x_i' \\ y_i' \\ Z_i \end{bmatrix} = \begin{bmatrix} p_i^1 \\ p_i^2 \end{bmatrix}.
\]

This gives
\[
(P_i^2)^\top \left( t_{1/2}^2 \right)_\times R_1^2 p_i^1 = 0.
\] (5.1)

The essential matrix is defined as
\[
E \triangleq \left( t_{1/2}^2 \right)_\times R_1^2.
\] (5.2)
In our method, we optimize the rotation and translation on the manifold to satisfy Equation (5.1). The essential matrix is only produced from the rotation and translation when it is needed for cost functions and derivatives. In contrast, many approaches in the literature solve for the essential matrix directly, which requires decomposing the essential matrix later to obtain the desired rotation and translation.

5.3 New Iterative Five-Point Algorithm

In this section, we drop the frame notation and simply write the rotation as $R$ and the translation as $t$, where $R \in SO(3)$ and $t \in S^2$. To estimate the rotation and translation between frames, we generate a large number of relative pose hypotheses from five-point minimum subsets and score them using LMedS. However, instead of calculating the essential matrix in closed-form, we optimize the rotation and translation components of the essential matrix to find the solution.

The optimization is done on the $SO(3) \times S^2$ manifold. We will call this the essential matrix manifold. This manifold can be represented using Lie groups, as discussed in subsections 5.3.1-5.3.3.

At each iteration, the optimizer requires derivatives with respect to each of the five degrees of freedom. Additionally, since the optimizer step is represented in these five degrees of freedom, it must be mapped back to the Lie group using the matrix exponential. These derivatives and the exponential map are also discussed in subsections 5.3.1-5.3.3.

The optimization algorithm minimizes the Sampson error using the Levenberg-Marquardt algorithm. This error and its derivatives are described in subsection 5.3.6. The Levenberg-Marquardt optimization algorithm used is described in subsection 5.3.7.

At each iteration of the LMedS algorithm, the optimization produces a relative pose hypothesis from a random five-point minimum subset. The best hypothesis is always used to seed the optimizer. The LMedS algorithm is described in subsection 5.3.8 and the optimizer seeding method is described in subsection 5.3.9.

The best relative pose hypothesis is refined using only inlier points. This refinement is performed using the same optimization algorithm used to generate the rotation and translation hypotheses. The refinement is described in subsection 5.3.10.
Lastly, since there are two rotations and two translations for which the Sampson error produces the same cost value, the correct rotation-translation pair must determined. Since the cheirality check often gives the wrong rotation matrix, we instead pick the rotation matrix with the smallest angle. We only use the cheirality check to determine the correct translation. This is described in subsection 5.3.11.

5.3.1 The \( SO(3) \) Manifold

The \( SO(3) \) manifold is the three-dimensional Lie group consisting of all \( 3 \times 3 \) rotation matrices. To be able to use elements of \( SO(3) \) in an iterative algorithm, these three degrees of freedom must be defined and the derivatives in each of these directions calculated. These will later be used to form the Jacobian in the LM optimization. The LM algorithm returns updates to the Lie group represented in the Lie algebra. Thus we also need a method to map between the Lie group and the Lie algebra. This can be done using the matrix exponential and the matrix logarithm. Thus to update elements of the Lie group by an incremental amount \( \delta R = \begin{bmatrix} \delta R_{1,1} & \delta R_{1,2} & \delta R_{1,3} \end{bmatrix}^\top \in so(3) \), we will use

\[
R_{k+1} = e^{(\delta R)} \cdot R_k.
\]

To simplify notation, we will define this update using the boxplus operator, as is done in [46],

\[
\boxplus : SO(3) \times \mathbb{R}^3 \rightarrow SO(3),
\]

\[
R_k \boxplus \delta R \triangleq e^{(\delta R)} \cdot R_k.
\]

For skew symmetric \( 3 \times 3 \) matrices the matrix exponential can be calculated efficiently using the Rodrigues formula

\[
e^{\omega \times} = I + \sin(\theta) \cdot \hat{\omega} \times + (1 - \cos \theta) \cdot \hat{\omega}^2 \times,
\]

(5.3)
where
\[ \hat{\omega} = \frac{\omega}{\|\omega\|} \]
and
\[ \theta = \|\omega\| . \]

To avoid division by zero this can be implemented using sinc functions, by substituting the trigonometric identity \(1 - \cos \theta = 2 \sin^2 \left( \frac{\theta}{2} \right)\). This gives
\[
e^{\omega_x} = I + \text{sinc} (\theta) \omega_x + \frac{1}{2} \text{sinc}^2 \left( \frac{\theta}{2} \right) \omega_x^2,
\]
where
\[
\text{sinc} (\theta) \equiv \begin{cases} 
\frac{\sin(\theta)}{\theta} & \text{if } \theta \neq 0 \\
1 & \text{otherwise.}
\end{cases}
\]

The derivative of the rotation matrix \( R_{k+1} \) along the direction \( \hat{\omega} \), evaluated at \( \theta = 0 \) is given by
\[
\frac{\partial}{\partial \theta} R_{k+1} = \left. \frac{\partial}{\partial \theta} e^{\theta \hat{\omega}_x} R_k \right|_{\theta=0} = \hat{\omega}_x e^{\theta \hat{\omega}_x} R_k \bigg|_{\theta=0} = \hat{\omega}_x R_k.
\]

Thus the derivatives of \( R \in SO(3) \) along each of the three degrees of freedom are
\[
\frac{\partial}{\partial \omega_{R,1}} R_{k+1} = (e_1)_x R_k, \\
\frac{\partial}{\partial \omega_{R,2}} R_{k+1} = (e_2)_x R_k,
\]

62
and

\[
\frac{\partial}{\partial \theta_{R,3}} R_{k+1} = (e_3) \times R_k,
\]

where

\[
e_1 = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix},
\]

\[
e_2 = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix},
\]

and

\[
e_3 = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}.
\]

### 5.3.2 The \( S^2 \) Manifold

The \( S^2 \) manifold is the set of all \( 3 \times 1 \) unit vectors and has two degrees of freedom. However, \( S^2 \) is not a Lie group, so instead we will represent it internally using elements of \( SO(3) \). To obtain an \( S^2 \) unit vector, we will multiply this internal rotation matrix by a unit vector in the \( \hat{z} \) direction to give

\[
t = Q^\top e_3.
\]
We will define an incremental update on the internal rotation matrix $Q$ by $\delta_t \in \mathbb{R}^2$ as

$$Q_{k+1} = e^{[\delta_{t,1} \ \delta_{t,2} ]} \times Q_k.$$ 

Once again, we will define a boxplus operator to condense notation,

$$\boxplus : S^2 \times \mathbb{R}^2 \rightarrow S^2,$$

so

$$Q_{k+1} = Q_k \boxplus \delta_t \triangleq e^{[\delta_{t,1} \ \delta_{t,2} ]} \times Q_k.$$ 

Taking derivatives in both degrees of freedom gives

$$\frac{\partial}{\partial \delta_{t,1}} Q_{k+1} = \left( (e_1 ) \times Q_k \right)^\top e_3$$

$$= \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & -1 & 0 \end{bmatrix} e_3$$

$$= Q_k^\top e_2$$
and

\[
\frac{\partial}{\partial \delta_{k,2}} Q_{k+1} = ((e_2) \times Q_k) \top e_3
\]

\[
= Q_k \top \begin{bmatrix}
0 & 0 & -1 \\
0 & 0 & 0 \\
1 & 0 & 0
\end{bmatrix} e_5
\]

\[
= -Q_k \top e_1.
\]

### 5.3.3 The Essential Matrix Manifold

The essential matrix is the product of the skew symmetric matrix of a translation unit vector and a rotation matrix:

\[
E = t \times R.
\]

Therefore, define the essential matrix manifold as the Cartesian product of the rotation matrix and the unit vector translation

\[
E \in \{ SO(3) \times S^2 \}.
\]

As before, we will define a boxplus operator to update elements of the essential matrix manifold by an incremental update \( \delta \in \mathbb{R}^5 \),

\[
\boxplus: \{ SO(3) \times S^2 \} \times \mathbb{R}^5 \to \{ SO(3) \times S^2 \},
\]

\[
E \boxplus \delta \triangleq \left\{ R \boxplus \begin{bmatrix} \delta_1 \\ \delta_2 \\ \delta_3 \end{bmatrix}, Q \boxplus \begin{bmatrix} \delta_4 \\ \delta_5 \end{bmatrix} \right\}.
\]   (5.4)
The derivatives of the essential matrix in each of the five degrees of freedom are

\[
\frac{\partial E_{k+1}}{\partial \delta_1} = t \times (e_1) \times R_k, \quad (5.5)
\]

\[
\frac{\partial E_{k+1}}{\partial \delta_2} = t \times (e_2) \times R_k, \quad (5.6)
\]

\[
\frac{\partial E_{k+1}}{\partial \delta_3} = t \times (e_3) \times R_k, \quad (5.7)
\]

\[
\frac{\partial E_{k+1}}{\partial \delta_4} = ((e_1) \times Q)^\top e_3 R_k, \quad (5.8)
\]

and

\[
\frac{\partial E_{k+1}}{\partial \delta_5} = ((e_2) \times Q)^\top e_3 R_k. \quad (5.9)
\]

### 5.3.4 Algebraic Error

In order to perform Gauss-Newton or Levenberg-Marquardt optimization and solve for the essential matrix, we need an error function. This is often called the residual and will be denoted \( r_i(p_i^1, p_i^2, E) \). The residual can be positive or negative. The cost is the square of the residual.

A simple residual or error function that is based on the original essential matrix equation is the algebraic error

\[
r_{i, \text{alg}} = (p_i^2)^\top E p_i^1.
\]

The partial derivative of this residual function is

\[
\frac{\partial r_{i, \text{alg}}}{\partial \delta_j} = (p_i^2)^\top \frac{\partial E}{\partial \delta_j} p_i^1.
\]
5.3.5 Single-Image Error

The algebraic error has no geometric interpretation and penalizes points that are further away from the center of the image more than those that are near the center of the image. An error function that has a geometric meaning can give better results. For example, for a point in the first image we can find the epipolar line in the second image and use the distance to this line as an error function. The epipolar line in the second image corresponding to the \( i \)th point can be described using the coefficients

\[
l_i^2 = \begin{bmatrix} a \\ b \\ \gamma \end{bmatrix} = E p_i^1.
\]

This results in the equation

\[
\begin{bmatrix} x_i^2 & y_i^2 & 1 \end{bmatrix} \begin{bmatrix} a \\ b \\ \gamma \end{bmatrix} = ax_i^2 + by_i^2 + \gamma = 0.
\]

This equation is a line in standard form. It can be shown that the vector \( v = \begin{bmatrix} a & b \end{bmatrix}^\top \) is perpendicular to this epipolar line. This vector can be normalized to be a unit vector,

\[
\hat{v} = \frac{\begin{bmatrix} a & b \end{bmatrix}^\top}{\sqrt{a^2 + b^2}}.
\]
If \( \begin{bmatrix} \bar{x}_i^2 & \bar{y}_i^2 \end{bmatrix}^\top \) is some point on the line such that \( \begin{bmatrix} \bar{x}_i^2 & \bar{y}_i^2 \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix}^\top = -\gamma \), then the distance from the point \( \begin{bmatrix} x_i^2 & y_i^2 \end{bmatrix}^\top \) to the line is the inner product with the perpendicular vector \( \hat{v} \), giving

\[
 r_{i,\text{single}} = \left( \begin{bmatrix} x_i^2 & y_i^2 \end{bmatrix} - [\bar{x} \ \bar{y}] \right) \frac{\begin{bmatrix} a \\ b \end{bmatrix}^\top}{\sqrt{a^2 + b^2}}
\]

\[
 = \begin{bmatrix} x_i^2 & y_i^2 \end{bmatrix} \frac{\begin{bmatrix} a & b \ \gamma \end{bmatrix}^\top}{\sqrt{a^2 + b^2}}
\]

\[
 = \frac{(p_i^2)^\top \ E p_i^1}{\sqrt{a^2 + b^2}}. \tag{5.10}
\]

We will call this the single-image error. Note that this distance can be positive or negative.

To write the derivative more succinctly, let

\[
 s = a^2 + b^2.
\]

Then the single-image error can be written as

\[
 r_{i,\text{single}} = \frac{(p_i^2)^\top \ E p_i^1}{\sqrt{s}}.
\]

Observe that the partial derivatives of \( a, b, \) and \( s \) are

\[
 \frac{\partial a}{\partial \delta_j} = \left( \frac{\partial E}{\partial \delta_j} p_i^1 \right)_1,
\]

\[
 \frac{\partial b}{\partial \delta_j} = \left( \frac{\partial E}{\partial \delta_j} p_i^1 \right)_2,
\]

and

\[
 \frac{\partial s}{\partial \delta_j} = 2a \frac{\partial a}{\partial \delta_j} + 2b \frac{\partial b}{\partial \delta_j}.
\]

68
Using the chain rule and the quotient rule, the derivative of the single-image error (5.10) can be written as

\[
\frac{\partial r_i}{\partial \delta_j} = \frac{\sqrt{s} \left( (p_i^2)^\top p_i^1 \right) - (p_i^2)^\top p_i^1 \frac{\partial s}{\partial \delta_j}}{s}.
\]

### 5.3.6 Sampson Error

However, using the single-image error assumes that the point in the first image has no error and forces all error to exist in the second image. In reality, error exists on points in both images. The Sampson error is a well-known approximation of the reprojection error in both images [47]. This error will be used as the residual function in the Levenberg-Marquardt optimization. The Sampson error for the \( i \)th point is

\[
r_{i,\text{sampson}} = \frac{(p_i^2)^\top E p_i^1}{\sqrt{(E p_i^1)_1^2 + (E p_i^1)_2^2 + (E^\top p_i^2)_1^2 + (E^\top p_i^2)_2^2}}.
\] (5.11)

To write the derivative more succinctly, let

\[
a = (E p_i^1)_1,
\]

\[
b = (E p_i^1)_2,
\]

\[
c = (E^\top p_i^2)_1,
\]

\[
d = (E^\top p_i^2)_2,
\]

and

\[
s = a^2 + b^2 + c^2 + d^2.
\]
Then we can express (5.11) as

\[ r_{i,\text{sampson}} = \frac{(p_i^2)^\top E p_i^1}{\sqrt{s}}. \]  

(5.12)

The partial derivatives of the short-hand expressions defined above are

\[ \frac{\partial a}{\partial \delta_j} = \left( \frac{\partial E}{\partial \delta_j} p_i^1 \right)_1, \]
\[ \frac{\partial b}{\partial \delta_j} = \left( \frac{\partial E}{\partial \delta_j} p_i^1 \right)_2, \]
\[ \frac{\partial c}{\partial \delta_j} = \left( \frac{\partial E}{\partial \delta_j} p_i^2 \right)_1, \]
\[ \frac{\partial d}{\partial \delta_j} = \left( \frac{\partial E}{\partial \delta_j} p_i^2 \right)_2, \]

and

\[ \frac{\partial s}{\partial \delta_j} = 2a \frac{\partial a}{\partial \delta_j} + 2b \frac{\partial b}{\partial \delta_j} + 2c \frac{\partial c}{\partial \delta_j} + 2d \frac{\partial d}{\partial \delta_j}. \]

Using the chain rule and the quotient rule, we can write the derivative of (5.12) as

\[ \frac{\partial r_{i,\text{sampson}}}{\partial \delta_j} = \frac{\sqrt{s} (p_i^2)^\top}{s} \frac{\partial E}{\partial \delta_j} p_i^1 - \frac{(p_i^2)^\top}{\sqrt{s}} E p_i^1 \frac{1}{\sqrt{s}} \frac{\partial s}{\partial \delta_j}. \]  

(5.13)

### 5.3.7 Levenberg-Marquardt Optimization

Both the Gauss-Newton and Levenberg-Marquardt optimizers begin with an initial guess of the rotation matrix and the translation unit vector. The Sampson error (5.11) can be evaluated at
this initial guess for each point to construct the residual vector

\[ r = \begin{bmatrix} r_1 \\ r_2 \\ \vdots \\ r_m \end{bmatrix}. \]

The Jacobian of the residual vector is constructed from the residual derivatives (5.13) in each of the five degrees of freedom along the essential matrix manifold described in section 5.3.3. This gives

\[ J = \begin{bmatrix} \frac{\partial r_1}{\partial \delta_1} & \frac{\partial r_1}{\partial \delta_2} & \frac{\partial r_1}{\partial \delta_3} & \frac{\partial r_1}{\partial \delta_4} & \frac{\partial r_1}{\partial \delta_5} \\ \frac{\partial r_2}{\partial \delta_1} & \frac{\partial r_2}{\partial \delta_2} & \frac{\partial r_2}{\partial \delta_3} & \frac{\partial r_2}{\partial \delta_4} & \frac{\partial r_2}{\partial \delta_5} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \frac{\partial r_m}{\partial \delta_1} & \frac{\partial r_m}{\partial \delta_2} & \frac{\partial r_m}{\partial \delta_3} & \frac{\partial r_m}{\partial \delta_4} & \frac{\partial r_m}{\partial \delta_5} \end{bmatrix}. \]

This Jacobian is a first-order Taylor-series approximation of the error function. In other words, the residual can be approximated by

\[ r \approx r_0 + J_0 \delta. \]

The goal is to bring the residual to zero. Based on this linear approximation of the error, we can solve for the Gauss-Newton incremental update \( \delta_{GN} \) using least squares, so that

\[ \delta_{GN} = - \left( J_k^\top J_k \right)^{-1} J_k^\top r_k. \]

The essential matrix manifold element is then updated using the boxplus operator defined in Equation (5.4).

Gauss-Newton iterations are repeated until the norm of the residual is lower than a threshold, or the maximum number of iterations is reached. In our implementation we use a maximum of ten iterations.

However, if the initial guess is too far from the true solution, the Gauss-Newton algorithm can take longer to converge or become unstable. The Levenberg-Marquardt algorithm is an
improvement on the Gauss-Newton algorithm that performs better in these conditions. It is essentially a hybrid between the Gauss-Newton algorithm and gradient descent, with a parameter $\lambda$ to mix the two algorithms, so that

$$
\delta_{LM} = - \left( J_k^T J_k + \lambda \left( J_k^T J_k \right)_{\text{diag}} \right)^{-1} J_k^T r_k,
$$

(5.14)

where $(\cdot)_{\text{diag}}$ indicates a matrix formed by only using the diagonal elements of the matrix. When $\lambda$ is small, the algorithm behaves like Gauss-Newton. When $\lambda$ is large, the algorithm behaves like gradient descent. The larger $\lambda$ is, the smaller the step size.

In applying the Levenberg-Marquardt algorithm, we begin with $\lambda = 10^{-4}$. At each iteration of the algorithm, we attempt to decrease the error by taking a step using Equation (5.14). If the new error is larger, we reject the step and double $\lambda$. We continue increasing $\lambda$ until we successfully decrease the error. We can then accept the step and move on to the next iteration of the algorithm. On the other hand, if the new error is smaller, we can immediately accept the step and cut $\lambda$ in half. This ensures the error monotonically decreases, while allowing the algorithm to converge faster if the residual function is well-behaved.

### 5.3.8 Consensus Algorithms

The two consensus algorithms used in this paper are random sample consensus (RANSAC) [36] and least median of squares (LMedS) [37]. At each iteration of RANSAC or LMedS, a minimum subset of size $n$ is chosen from which to generate a model. In the case of solving for the relative pose between frames, the subset is chosen from the $m$ point correspondences at the current time step. The subset is called a minimum subset because it is the smallest number of points that provide enough constrains to generate a solution, or model. In the case of the solving for the relative pose, $n = 5$ point correspondences are used, since there are five degrees of freedom and each point correspondence gives one constraint. For each minimum subset, one or more models that fit the points are solved for.

After generating $N$ minimum subsets, each with their corresponding solutions, all of these models are scored to find the best one. The difference between RANSAC and LMedS is how the models are scored. For RANSAC, the score is the total number of inliers with a residual lower...
than the RANSAC threshold $\tau$. This can be written as a cost function if the number of outliers are counted instead of the number of inliers, so that

$$cost = \sum_{i=0}^{m} \begin{cases} 
1 & |r_i| > \tau \\
0 & \text{otherwise}.
\end{cases}$$

For LMedS, the cost is instead calculated by finding the median of the square of the residuals, so that

$$cost = \text{median}_{i=0}^{m} r_i^2.$$ 

The advantage to using LMedS is that the threshold does not need to be known before-hand. However, it only works if less than half of the points are outliers.

The number of iterations required when using RANSAC or LMedS is typically solved for by treating each matched pair of points as a binomial random variable representing whether the point is an inlier or an outlier [48]. For example, suppose the ratio of outlier point correspondences is $\epsilon$, and the number of points used to generate the model is $n$. Then the probability that all point correspondences used to generate the model are inliers is

$$p_m = (1 - \epsilon)^n.$$ 

If $N$ models are generated, then the probability that at least one of models is generated using only inliers is

$$p = 1 - (1 - (1 - \epsilon)^n)^N.$$ 

We can now solve for the number of subsets needed to achieve a desired confidence level $p$. This gives

$$N = \frac{\log (1 - p)}{\log (1 - (1 - \epsilon)^n)}.$$ 

For example, achieving a 99% confidence ratio when the outlier ratio is 50% requires 145 RANSAC or LMedS iterations.
5.3.9 Seeding the Optimizer

The method in which the optimizer is seeded is essential to achieving good performance. It is important that the optimizer be seeded with a good initial guess. When no prior information is known about the camera motion, the algorithm can be seeded with the identity rotation and a random translation unit vector. This method we will call the random initialization method.

Prior information can significantly increase performance. One way to incorporate prior information is to seed the optimizer at each LMedS iteration with the best hypothesis from the previous time step. We will denote this method as the “prior” initialization method.

Another approach is to seed the optimizer with the best hypothesis so far from the current time step. For the first iteration of LMedS, there is no best hypothesis and so the optimizer must be seeded randomly. However, for each subsequent iteration, the best hypothesis can be used to seed the optimizer. Since each hypothesis depends on the previous best hypothesis, we will denote this method as the “random recursive” initialization method.

These two approaches can also be combined. This results in one long continuous stream of hypotheses which depend on the previous best hypothesis. We will denote this method as the “prior recursive” initialization method.

5.3.10 Refining Relative Pose Hypotheses

Even the best hypothesis from RANSAC and LMedS usually has some error due to error on the individual points in the minimum subset used to create the hypothesis. The estimate can be improved by using least-squares optimization on the manifold over inlier points.

Since our algorithm already uses LM to optimize relative pose hypotheses, it can easily be extended to accept more than five points to refine the best relative pose hypothesis. In contrast, the OpenCV five-point algorithm does not have any built-in optimization method or separate function to perform the optimization.

Naive optimization using least-squares will often make the estimate worse if the data contains outliers. Robust least-squares optimization methods either explicitly find the inliers [49], or use a robust cost function that essentially gives higher weights to measurements with lower residuals [50].
For robust refinement we will find the inliers by first estimating the standard deviation $\hat{\sigma}$ [49]. For an explanation of these cryptic numbers the reader is referred to [51], page 202. The robust standard deviation is given by

$$\hat{\sigma} = 1.4826 \left[ 1 + \frac{5}{n-p} \right] \sqrt{m},$$

where $p = 5$ is the number of points used to create the model, $n$ is the total number of points, and $m$ is the median squared from LMedS. Inliers are defined to be any points within two and a half standard deviations. Thus the weights or inlier mask is

$$w_i = \begin{cases} 
1 & r_i^2 < 2.5\hat{\sigma} \\
0 & \text{otherwise} 
\end{cases}$$

Even after attempting to remove outliers, however, the refined estimate can sometimes be worse than the original. To prevent this, we score the refined hypothesis using the same LMedS Sampson error metric to see if it is better or worse than the original. If the new hypothesis has a lower LMedS error it is kept, otherwise it is discarded.

### 5.3.11 Determining the Correct Rotation and Translation

For any relative pose hypothesis, there are two possible rotations $R_1$ and $R_2$, as well as two possible translations $t$ and $-t$, that will produce the same Sampson cost. This is because the Sampson cost is based on the essential matrix. Each of these possible rotation and translation pairs produce equivalent essential matrices, and thus our hypothesis generation and scoring algorithms cannot distinguish between them.

Direct essential matrix solvers usually find these rotation matrices and the translation vector by extracting them from the essential matrix using the singular-value decomposition (SVD), as shown in Appendix A. However, since our optimization already produces one possible rotation matrix and translation vector, we can determine the other possible rotation by rotating the first rotation 180 degrees about the translation vector, as shown in Appendix B, Equation (B.3). This is simpler than taking the SVD of the essential matrix.
After finding the two rotation matrices and the translation vector, the cheirality check is often used to determine which of rotation matrices and which translation sign is correct. The cheirality check involves triangulating each point to determine its 3D position. In our implementation of the cheirality check we count the number of triangulated points with positive depth and then add the totals from both cameras. The rotation and translation pair with the largest sum is chosen. However, with four possible choices, the cheirality check often gives spurious results.

Alternatively, since the two possible rotations are always 180 degrees apart, we can pick the rotation with the smallest angle and use the cheirality check to find the correct translation. In a continuous video sequence the true rotation between frames is usually very small. Thus one possible rotation angle will be close to zero and the other possible rotation angle will be close to 180 degrees. As long as the true rotation is never more than 90 degrees, the correct rotation will always be the one with the smaller angle.

Taking the trace of the matrix is a computationally efficient method of picking the rotation with the smaller angle. Notice that the first and third terms of the Rodrigues formula (5.3) are the only terms with diagonal components. Thus the trace of the rotation matrix is

\[
tr(R) = 3 - (1 - \cos(\theta))(2\hat{\omega}_1^2 + 2\hat{\omega}_2^2 + 2\hat{\omega}_3^2)
\]

\[
= 3 - 2(1 - \cos(\theta))
\]

\[
= 1 + 2\cos(\theta).
\]  

(5.15)

Since \(\cos(\theta)\) is monotonically decreasing on \(0 < \theta < \pi\), taking the trace of the matrix \(tr(R(\theta))\) will also be monotonically decreasing. Thus the trace can be used to determine which rotation matrix has the smallest angle, giving

\[
R = \begin{cases} 
R_1 & \text{if } tr(R_1) > tr(R_2) \\
R_2 & \text{otherwise.}
\end{cases}
\]
5.3.12 Complete System

The complete system is shown in Figure 5.2. At each time step, features are detected in one frame using good features to track [24] and then matched to the next frame using LK optical flow [16]. These point pairs are the inputs to ReSORtSAC.

At each iteration of LMedS, the LM optimizer is seeded with the best initial guess, using the prior recursive method described in section 5.3.9. Ten LM iterations are used, with iterations being repeated if the error is not successfully decreased (Section 5.3.7). After each iteration of LMedS, the new hypothesis is scored. If the median squared error is lower than the median squared error of the best hypothesis so far, the new hypothesis is kept (Section 5.3.8).

The best hypothesis is then refined using inliers (Section 5.3.10) and its rotation and translation disambiguated (Section 5.3.11). The result is the final essential matrix estimate.

Figure 5.2: Block diagram of ReSORtSAC
5.3.13 Computational Considerations

There are a couple of computational considerations in implementing the ReSORtSAC relative pose estimator. The most important of these considerations is to reduce the number of matrices allocated on the heap, as heap allocations are computationally expensive. One reason the OpenCV polynomial essential matrix solver is slow is because it allocates a large number of dynamic matrices on the heap.\footnote{Based on our own observation of the publicly available OpenCV source code [26]} Our implementation allocates fixed-size matrices on the stack and initializes dynamic matrices from raw buffers created once at the beginning of the program. These initialization options are available in both Eigen and OpenCV, but we have chosen to implement our algorithm using Eigen.

5.4 Motion Detection and Tracking in the Presence of Parallax

One application of relative pose estimation is motion detection and tracking in the presence of parallax. Motion detection is a valuable source of information in tracking. It can be used to track objects without any prior knowledge about their appearance, in contrast to many trackers that are designed to track specific classes of objects.

There are many successful image-based background subtraction techniques in the literature that work on stationary cameras. In order for image differencing techniques to work on a moving camera, the images must first be aligned using a homography. This works well for planar scenes. But if there is parallax, artifacts can appear in the difference image. If the parallax is small enough in comparison to the movement of objects in the scene, the effects of parallax can be reduced using simple morphological operations and gradient suppression, as is done in [11].

In the presence of strong parallax, however, a better motion model that accounts for depth variation must be used. There are several methods in the literature that use a geometric model to describe the motion of tracked points in the scene over time. A KLT tracker or other tracker can be used to generate sparse point trajectories. These can then be used in the motion model.

With these tracked points, one approach is to approximate the camera as orthographic instead of perspective [14]. With this approximation, the motion of the camera and the points in the scene over time can be solved using a system of linear equations. Points that are not consistent with
this motion must be moving points. This approach works well if the camera is far from the scene or only observes a small field of view, but does not work well for perspective transformations, because the linear orthographic assumption is violated.

Another approach is to maintain an affinity matrix over time. Each cell of this matrix stores the dissimilarity between two points and is calculated from the maximum difference of position and velocity between these two points up until the current frame. This affinity matrix can be mapped into an embedding space. The points can then be clustered in the embedding space to distinguish between foreground and background points and to segment moving objects [15]. Another approach is to use matrices to describe multiple-frame geometric constraints [13], [52].

These approaches are well developed, but the papers describing these methods do not include computation times. In this paper, we use a simple method for motion detection using only the two-frame essential matrix constraint. The advantages of the method are computational simplicity and real-time performance.

5.4.1 Problem Setup

Given two consecutive frames, with point correspondences detected in each frame, the objective is to determine which points are from stationary objects and which are from moving objects. It is assumed that the essential matrix $E$, along with its rotation $R$ and translation $t$ have already been calculated using the iterative method described in Section 5.3.

This is a much simpler problem when the camera is stationary, since all motion observed in the image plane comes from moving objects in the world frame. However, for a moving camera, the problem becomes more difficult because object motion and camera motion both cause apparent motion in the image plane. The challenge is determining the source of this motion. In other words, the goal is to design a detector $\phi(p_i)$ which returns 1 if the $i$th point is moving and 0 otherwise. The output of the motion detector is used as an input to a tracking algorithm in order to produce target estimates.
5.4.2 Motion Detection Algorithm

The essential matrix relates points in one image to the other image with the epipolar constraint. In other words, the essential matrix maps a point in one image to a line in the other image. Where the point in the other image appears along this line depends on the real-world depth of the point from the camera. As the camera translates, points that are closer to the camera will appear to move more than the points that are far away. This effect is known as parallax.

There are two degrees of freedom for the apparent motion of each point in the image plane. One of these degrees of freedom can be explained by the epipolar constraint if the real-world point is stationary. However, motion along this degree of freedom can also be explained by object motion in the world frame. Hence the source of any movement along this degree of freedom is ambiguous without additional information. The second degree of freedom for apparent motion of points in the image plane is perpendicular to the epipolar constraint. Thus the only possible source of motion along this degree of freedom is movement in the real-world frame.

Note that except in degenerate cases, each point in the image will move in a different direction due to parallax. For example, when the camera is moving towards the scene or away from the scene, the points move along radial epipolar lines which intersect at the center of the image. As a result, it can be helpful to describe the motion of the points due to parallax using a vector field. Define the parallax field as a unit vector assignment to each point in the image plane that points in the direction the point would move due to parallax. The second degree of freedom is perpendicular to the parallax field, thus we will define the perpendicular field as a 90 degree clockwise rotation of each of these vectors.

These two vector fields give basis vectors at each point that will be used to decompose apparent motion in the image plane into the two degrees of freedom described above. We will denote the velocity in these two directions as the parallax velocity and the perpendicular velocity respectively.

Before considering the effects of parallax, however, we must compensate for the rotation of the camera. One method of compensating for the rotation of the camera is to use a Euclidean homography. Like the regular homography matrix, the Euclidean homography also maps points in one image to points in the other image using homogeneous coordinates, but operates on image
plane coordinates instead of pixel coordinates. The Euclidean homography is given by

\[
H_e = \left( R + \frac{tn^\top}{d} \right) \gamma,
\]

where \( t \) is a translation vector, \( n \) is the normal vector of the plane which the homography describes, \( d \) is the distance to this plane, and \( \gamma \) is a scale factor.

The homography requires normalizing each homogeneous point after left-multiplying the point by the homography matrix. To condense notation, define \( g(p) \) as an operator that normalizes homogeneous points, so that

\[
g(p) \triangleq \frac{p}{p_z} = \begin{bmatrix} \frac{p_x}{p_z} \\ \frac{p_y}{p_z} \\ 1 \end{bmatrix}.
\]

Compensating for the rotation alone gives

\[
\hat{p}_2^2 = g(H_ep_1^1) = g(Rp_1^1),
\]

where \( \hat{p}_2 \) is the estimated location of the \( i \)th point in the next camera frame.

Note that this estimated location is where the point would be in the second image if it had infinite depth. The remaining point velocity is the sum of the true velocity of the point and the apparent velocity due to parallax. Subtracting the estimated point and true point in the second image gives the remaining velocity

\[
v_2^2 = p_2^2 - \hat{p}_2^2.
\]

We can then use the essential matrix to calculate the parallax and perpendicular fields. The essential matrix equation (5.1) can be rewritten as a line in standard form, giving

\[
ax_i^2 + by_i^2 + \gamma = 0,
\]

where \( a = (Ep_1^1)_1, b = (Ep_1^1)_2 \), and \( \gamma = (Ep_1^1)_3 \).
It can be shown that the vector $\begin{bmatrix} a & b \end{bmatrix}^T$ is perpendicular to this epipolar line and the vector $\begin{bmatrix} -b & a \end{bmatrix}^T$ is parallel to it. Assuming the essential matrix is formed from the relative pose as given in Equation (5.2), this parallel vector will have the correct sign and will be pointing in the direction of parallax. Making these vectors unit vectors gives the parallax field

$$f_{\parallel} \triangleq \begin{bmatrix} b \\ -a \end{bmatrix} / \sqrt{a^2 + b^2}$$

and the perpendicular field

$$f_{\perp} \triangleq \begin{bmatrix} a \\ b \end{bmatrix} / \sqrt{a^2 + b^2}.$$

The component of apparent velocity in the image plane in each direction can be found using the dot product with each field. There is no need to divide by the magnitude since the fields are already unit vectors. This gives

$$v_{\parallel} = \left\langle f_{\parallel}, \begin{bmatrix} v_x \\ v_y \end{bmatrix} \right\rangle$$

and

$$v_{\perp} = \left\langle f_{\perp}, \begin{bmatrix} v_x \\ v_y \end{bmatrix} \right\rangle.$$

The perpendicular velocity can then be thresholded to determine if a particular object is moving. The parallel velocity can also sometimes help distinguish between moving and stationary objects. Since parallax always makes stationary points appear to move in the same direction as camera translation, if the parallax velocity is negative it indicates that the object is moving. Thus the decision function for whether a point is moving can be written as

$$\phi(p_i) = \begin{cases} 1 & \text{if } |v_{\perp}| > \tau \text{ or } v_{\parallel} < -\tau \\ 0 & \text{otherwise,} \end{cases}$$
where $\tau$ is the moving point threshold. Due to small errors in calculating the essential matrix and lack of camera calibration, a threshold of one pixel is often the tightest constraint that can be used.

### 5.4.3 Recursive-RANSAC

Moving points found using motion detection are then fed into recursive-RANSAC, a newly proposed algorithm for multi-target tracking [22]. At each time step recursive-RANSAC searches for new models using RANSAC. When a sufficient number of inliers are detected, a new track is initialized.

Existing tracks are propagated forward using a Kalman filter. Probabilistic data association [25] is used to account for measurement association uncertainty. Each track is given an inlier score based on the percentage of time steps in which the track is detected. Recursive-RANSAC also has a track management system that merges similar tracks and removes tracks that have an inlier score lower than the minimum threshold.

### 5.5 Results

Two video sequences were used to test the two algorithms presented in this paper. The ReSORtSAC relative pose algorithm was tested on a synthetic video sequence of a UAV inside a city. The synthetic video sequence has no moving objects, so the complete motion detection and tracking algorithm was tested on a video sequence taken from a UAV camera.

#### 5.5.1 Iterative Five-Point Algorithm

The ReSORtSAC relative pose algorithm was tested on a synthetic video sequence of a UAV inside a city generated using the BYU Holodeck simulator [53]. The two-minute video sequence (3600 frames) includes aggressive rotational and translational motions. A screenshot of the video sequence is shown in Figure 5.3.

There are three error metrics for each algorithm comparison. The rotational error is the smallest rotation angle between the true rotation and the estimated rotation. This angle is the norm
of the matrix logarithm and can be efficiently calculated by solving (5.15) for $\theta$, giving

$$e_R = \| \log R_{true} R_{est}^{-1} \|$$

$$= \frac{\text{tr} \left( R_{true} R_{est}^{-1} \right) - 1}{2}.$$

The translational error is computed by finding the angle between the true unit vector translation and the estimated unit vector translation, so that

$$e_t = \cos^{-1} \left( \hat{t}_{true}^\top \hat{t}_{est} \right).$$

Note that when calculating the rotation and translation error metrics, we do not penalize pose disambiguation errors. We consider the rotation and translation error metrics to be independent from the pose disambiguation error metrics, since there are four possible rotation-
translation pairs for which the Sampson error produces the same cost value. The rotation and translation errors are thus computed for all possible rotations and translations and smallest of these errors is returned. The pose disambiguation error metrics show the percentage of the time that the relative pose algorithm chooses the correct rotation and translation.

Both the rotational and translational error are measured in radians. The LMedS Sampson error is also computed. Unless otherwise noted, all error metrics are averaged over the entire video sequence of 3599 frame pairs.

The error over time for the OpenCV LMedS polynomial solver and the ReSORtSAC solver is shown in Figure 5.4. Both the OpenCV and ReSORtSAC solvers give low error for the UAV trajectory. Notice how the rotation error seems to be proportional to the total rotation, while the translation error becomes very large as the true translation approaches zero.

![Figure 5.4: Error over entire video sequence](image-url)
Various methods of initializing the LM optimizer were tested and compared against the OpenCV five-point polynomial solver. To compare these methods, the LMedS algorithm was run for 100 iterations at each time step for the complete video sequence. At each LMedS iteration the error of the best hypothesis was recorded. The mean error across the entire video sequence is plotted in Figure 5.5.

This result shows the importance of initializing the optimizer with a prior. The random initialization method performs the worst out of all four methods, while initializing the optimizer with a prior from the previous time step or the best LMedS hypothesis so far from the current time step significantly reduces the error. After 100 iterations, the LMedS error for the initialization methods that use prior information is comparable to the OpenCV five-point polynomial solver, despite the fact that only one hypothesis is generated per subset instead of an average of about four.

![Figure 5.5: ReSORtSAC Levenberg-Marquardt seeding methods](image-url)
Similarly, Figure 5.6 shows the error of our algorithm and the OpenCV algorithm, but with the x-axis changed to be time instead of number of iterations. When under a time constraint, ReSORtSAC significantly outperforms the OpenCV solver.

The algebraic, single-image, and Sampson errors described in sections 5.3.4, 5.3.5, 5.3.6 were also tested. For each LMedS error function, the error between the true and estimated rotation and translation were computed. The mean of these errors over all 3599 frames is shown in Table 5.1. The algebraic error cost function performed significantly worse than the single-image error and Sampson error cost functions. The single-image error and Sampson error cost functions had comparable performance. This may be because the rotation between the two frames is small. For larger rotations the Sampson error is likely to perform better. In our optimization, we minimize the Sampson error.
Table 5.1: Relative pose cost functions

<table>
<thead>
<tr>
<th>LMedS cost function</th>
<th>Rotation error (radians)</th>
<th>Translation error (radians)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Algebraic</td>
<td>6.032e-04</td>
<td>2.352e-01</td>
</tr>
<tr>
<td>Single image</td>
<td>4.526e-04</td>
<td>1.687e-01</td>
</tr>
<tr>
<td>Sampson</td>
<td>4.573e-04</td>
<td>1.706e-01</td>
</tr>
</tbody>
</table>

Table 5.2: Gauss-Newton and Levenberg-Marquardt error comparison

<table>
<thead>
<tr>
<th>Relative pose solver</th>
<th>Rotation error (radians)</th>
<th>Translation error (radians)</th>
<th>LMedS error (Sampson)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline (OpenCV)</td>
<td>4.843e-04</td>
<td>1.742e-01</td>
<td>3.769459e-08</td>
</tr>
<tr>
<td>GN + LMedS</td>
<td>4.806e-04</td>
<td>1.731e-01</td>
<td>3.773465e-08</td>
</tr>
<tr>
<td>LM + LMedS</td>
<td>4.573e-04</td>
<td>1.706e-01</td>
<td>3.623720e-08</td>
</tr>
</tbody>
</table>

Table 5.2 compares the error of the LMedS Gauss-Newton and Levenberg-Marquardt methods. Levenberg-Marquardt has a noticeably lower error. This could imply that LM succeeds at finding a solution a larger percentage of the time. However, Figure 5.7 shows that this is not the case. LM actually converges to a solution less frequently than GN. The convergence ratio shown in Figure 5.7 is calculated by finding the percentage of total trials where the norm of the residual is smaller than 1e-8 for the given iteration of the algorithm.

The more likely explanation for the LM algorithm’s improved performance is that the LM algorithm takes smaller steps, as illustrated in Figure 5.8. When the gradients and non-linearities are large, GN is unstable. LM does a better job of dealing with large gradients and non-linearities. This means LM is more likely to converge to a nearby solution that is close to the prior from the previous time step.

RANSAC and LMedS were also compared. For RANSAC the algorithm was tested with 19 different thresholds. For LMedS, the algorithm was run once, since there is no threshold parameter to tune. For each run the average truth rotation and translation error over the entire video sequence were calculated. As shown in Figure 5.9, LMedS performs well without requiring a threshold. However, in order for RANSAC to perform as well as LMedS, the threshold must to be tuned to within an order of magnitude of the optimal threshold.
Figure 5.7: Gauss-Newton and Levenberg-Marquardt convergence

Table 5.3 shows the results of the rotation and translation disambiguation algorithms. The first row within each group of relative pose solvers shows a baseline comparison, where no method was used for pose disambiguation. The baseline method gives poor results. However, it is worth noting that ReSORtSAC is able to keep the correct rotation, even without any form of pose disambiguation. This is likely because it is seeded at the first frame with the identity rotation, and every frame thereafter the best hypothesis from the previous is reused as a seed to the optimizer.

The second row in each group shows the results when the cheirality check was used (Section 5.3.11) to determine the best out of the four possible rotation translation pairs. Though the translation direction is often correct, the rotation is correct only about half of the time. The third row in each group shows the results of using the matrix trace to determine which rotation is correct, with the cheirality check to determine the correct translation direction. This third pose disambiguation method consistently outperforms the other methods.
Figure 5.8: Gauss-Newton and Levenberg-Marquardt average step size

Table 5.3: Pose disambiguation comparison

<table>
<thead>
<tr>
<th>Solver</th>
<th>Pose disambiguation method</th>
<th>Rotation correct</th>
<th>Translation correct</th>
<th>Both correct</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpenCV</td>
<td>none</td>
<td>50.2%</td>
<td>14.7%</td>
<td>6.8%</td>
</tr>
<tr>
<td>OpenCV</td>
<td>cheirality</td>
<td>54.0%</td>
<td>93.2%</td>
<td>52.3%</td>
</tr>
<tr>
<td>OpenCV</td>
<td>trace + cheirality</td>
<td>100.0%</td>
<td>96.5%</td>
<td>96.5%</td>
</tr>
<tr>
<td>ReSORtSAC</td>
<td>none</td>
<td>100.0%</td>
<td>53.1%</td>
<td>53.1%</td>
</tr>
<tr>
<td>ReSORtSAC</td>
<td>cheirality</td>
<td>40.9%</td>
<td>92.1%</td>
<td>40.8%</td>
</tr>
<tr>
<td>ReSORtSAC</td>
<td>trace + cheirality</td>
<td>100.0%</td>
<td>96.5%</td>
<td>96.5%</td>
</tr>
</tbody>
</table>
Table 5.4: ReSORtSAC relative pose refinement

<table>
<thead>
<tr>
<th>Relative pose solver</th>
<th>Refine success</th>
<th>Rot err (radians)</th>
<th>Trans err (radians)</th>
<th>LMedS err (Sampson)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline (OpenCV)</td>
<td>-</td>
<td>4.843e-04</td>
<td>1.742e-01</td>
<td>3.769e-08</td>
</tr>
<tr>
<td>Before refinement</td>
<td>-</td>
<td>4.573e-04</td>
<td>1.706e-01</td>
<td>3.624e-08</td>
</tr>
<tr>
<td>After refinement</td>
<td>55.0%</td>
<td>3.779e-04</td>
<td>1.596e-01</td>
<td>3.519e-08</td>
</tr>
</tbody>
</table>

Table 5.4 compares the average error before and after refinement using only inlier points (Section 5.3.10). The refinement is defined to be successful if the new relative pose has a lower LMedS Sampson error. The new relative pose is only kept if the refinement was successful. Refining the best relative pose hypothesis significantly reduces all three error metrics.

The computation times for the relative pose solvers is shown in Table 5.5. Notice that the OpenCV polynomial solver takes much longer to generate hypotheses and also requires scoring
Table 5.5: Computation time

<table>
<thead>
<tr>
<th></th>
<th>OpenCV poly</th>
<th>ReSORtSAC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hypothesis generation</td>
<td>100 * 0.400 ms = 40.0 ms</td>
<td>100 * 22.7 us = 2.27 ms</td>
</tr>
<tr>
<td>Hypothesis scoring</td>
<td>400 * 17.2 ns = 6.89 ms</td>
<td>100 * 9.22 ns = 0.92 ms</td>
</tr>
<tr>
<td>Refinement</td>
<td>-</td>
<td>5.98 ns</td>
</tr>
<tr>
<td>Pose disambiguation</td>
<td>0.32 ms</td>
<td>0.15 ms</td>
</tr>
<tr>
<td>Total</td>
<td>47.2 ms</td>
<td>3.94 ms</td>
</tr>
</tbody>
</table>

four times as many hypotheses. The time required to score each hypothesis is proportional to the number of points detected in each frame, which for this video sequence is on average about 400 points. The relative pose solvers were benchmarked on a laptop with a 2.1 GHz Intel i7 CPU running Linux. The breakdown of the time required to generate each hypothesis set is shown in Figure 5.10. The most time-consuming part of the OpenCV solver is finding the zeros of the tenth-order polynomial. The most time-consuming part of the GN and LM solvers is the Eigen matrix solver.

![Figure 5.10: Time required to generate each hypothesis set](image-url)
5.5.2 Motion Detection and Tracking Results

The motion detection algorithm was tested on a moving camera video sequence taken from a UAV. Figure 5.11 shows the results of the motion detection algorithm. A video of these results is available at https://youtu.be/HhK-p2hXNnU. Notice how the stationary points have zero perpendicular velocity and a positive parallax velocity, while the moving points have a non-zero perpendicular velocity component.

The computation times of the motion detection and tracking algorithm are shown in Table 5.6. For faster processing the video was scaled to 640×480. The motion detection and tracking algorithm is running on a Linux desktop computer with a 4GHz Intel i7 CPU. On average about 800 points are detected, tracked, and fed to the relative pose solver each frame. Notice that the OpenCV feature detection and tracking are the most time-consuming components of the tracking algorithm and consume 70% of the total CPU usage. The complete algorithm takes 29 milliseconds to run per frame, which means it is capable of running in real-time at 34 frames per second (FPS).

![Figure 5.11: Video motion detection results. Each point position (left) and its corresponding net velocity (right) are plotted. Points with a net perpendicular velocity greater than one pixel are classified as moving points (red), while points with a velocity below this threshold are classified as stationary points (blue).](image)
Table 5.6: Motion detection and tracking computation times

<table>
<thead>
<tr>
<th>Tracking Component</th>
<th>Computation Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Good features to track</td>
<td>9.2 ms</td>
</tr>
<tr>
<td>LK optical flow</td>
<td>12 ms</td>
</tr>
<tr>
<td>Calc E (ReSORiSAC)</td>
<td>3.0 ms</td>
</tr>
<tr>
<td>Recursive RANSAC</td>
<td>0.4 ms</td>
</tr>
<tr>
<td>Other</td>
<td>4.4 ms</td>
</tr>
<tr>
<td>Total</td>
<td>29 ms (34 FPS)</td>
</tr>
</tbody>
</table>

5.6 Conclusion

In this work, we have presented an iterative five-point algorithm for solving the rotation and translation between consecutive frames capable of running in real-time. We show the importance of seeding the Levenberg-Marquardt optimizer with an initial guess and demonstrate that this initial guess significantly improves the performance to the algorithm. We have applied this algorithm to detecting motion and tracking multiple targets from a UAV and demonstrated real-time performance of this tracking algorithm on a 640×480 video sequence.
Future work includes using an IMU to improve the initial guess of the rotation and translation from the previous time step. Future work also includes applying the principles of using an initial guess to seed the optimizer to 3D scene reconstruction and more complex tracking methods. The depth of moving objects can be estimated by using nearby 3D reconstructed points, building on the method used in [54], in order to estimate the 3D position of moving targets.
CHAPTER 6. CONCLUSION

In this thesis we have explored the challenges of tracking multiple targets from a camera mounted on an unmanned aerial vehicle (UAV) and have presented several algorithms to overcome these challenges. We have chosen to focus on point-based detection and tracking methods to achieve real-time performance on a UAV without requiring a GPU. In our method, Lucas-Kanade (LK) optical flow is used to generate motion detections of objects of interest in the scene. When this motion detection method is combined with long-term point-tracking methods, detections can be generated even when objects are no longer moving. This hybrid detection algorithm is shown to outperform other state-of-the-art tracking algorithms in terms of speed and track continuity.

Point-based motion detection methods generate multiple detections for each object at each time step. In our approach, we treat the detections around each target as clusters instead of a single detection source and estimate the center of the cluster using a method similar to expectation maximization. We modify the probabilistic data association filter to use membership probabilities as weights. We show that this modification improves MOTA and MOTP tracking metrics on a video sequence.

Since the target estimates are represented in the image plane, as the camera moves the target estimates must be transformed into the next frame. The position of targets are transformed using a homography. However, this $3 \times 3$ matrix produces unexpected results if used directly when transforming velocities or covariances into the next frame. The correct $2 \times 2$ transformation matrix can be calculated through differentiation and is shown to improve target tracking estimates for targets traveling on planar surfaces with simulated tracking data.

We have also presented a simple motion detection algorithm for non-planar UAV imagery. On non-planar UAV imagery, motion detection using the homography matrix results in significant artifacts due to parallax. To account for parallax, we calculate the direction each point should move due to parallax, and classify any points traveling in other directions as moving points.
Instead of calculating the essential matrix directly, we iteratively optimize the rotation and translation between consecutive frames, and from this compute the essential matrix. We show that this approach produces comparable relative pose estimates, but an order of magnitude faster, thus enabling real-time performance. We show that the complete parallax-compensated motion detection algorithm with recursive RANSAC can track objects in real-time on a $640 \times 480$ video sequence.

6.1 Contributions

The contributions of this thesis are outlined below.

- This thesis extends the MAGICC lab visual multi-target tracker to be able to track stopped objects from a moving camera.
- It improves the probabilistic data association component of the tracker to track clusters of detections with greater accuracy.
- It derives planar transformations of the velocity, covariance, and higher order terms for tracking targets in the image plane.
- It introduces a simple parallax-compensated motion detection algorithm to be used with the tracker.
- It presents a light-weight algorithm for estimating the rotation and translation between consecutive frames of a video sequence.

6.2 Future Work

The algorithms presented in this thesis have several areas for improvement. Perhaps most significantly, not all of the algorithms in this thesis are able to run concurrently. Other areas for improvement include better clustering of detections, tracking through occlusions, more reliable parallax-compensated motion detection, and a three-dimensional target tracking model.
6.2.1 Combining Tracking Algorithms

One area for improving the algorithms in this thesis is to combine them so they can all run concurrently. The improved motion detection algorithm presented in Chapter 5 is the first algorithm in this thesis designed for three dimensions. As such, some of the earlier algorithms presented in previous chapters may require adaptation in order to work well with this improved motion detection algorithm.

First, the ghost track reduction component of the stopped object tracking algorithm (Section 2.6) needs to be modified, as it currently relies on using a homography to align candidate background images to the current frame. Second, the tracking model state transformation (Chapter 4) needs to be extended to handle targets with different depths, since it relies on using a homography to transform target states between frames. Lastly, each individual algorithm runs in real-time within a tracking framework, but the algorithms may require a small amount of optimization in order to all run in real-time on the same CPU.

6.2.2 Improved Clustering in the Membership PDA Filter

One weakness of the MPDA filter is that it is sensitive to errors in cardinality. If there are multiple track estimates on a single target, the MPDA filter will assume that the estimates belong to different targets and prevent them from coalescing. Similarly, where there is only a single track estimate on multiple parallel-traveling targets, the MPDA filter will attempt to track all them with a single track estimate between the targets until the distance between them becomes too large to track all targets at the same time.

One area for improvement is to estimate the number of measurement clusters. At the moment it is assumed that there is only one measurement cluster per target. Clustering methodologies such as Dirichlet processes could be used to split clusters when the measurement distribution is bimodal, or merge clusters if the measurement distribution is unimodal. New clusters will appear as unassociated measurements in recursive RANSAC, and if consistent over time, will suggest the creation of new tracks. Similarly, if clusters are consistently merged in the clustering algorithm, it will suggest that their respective tracks should be merged.
6.2.3 Tracking Through Occlusions

The probabilistic data association (PDA) filter can handle short occlusions by propagating the state estimate forward in time, but has difficulty handling long occlusions. During long occlusions, the target estimate is easily pulled away from the true state by clutter or nearby targets.

One possible solution is to use the object’s appearance to recognize when an occlusion occurs. During the occlusion, the target estimate could coast without allowing clutter or nearby targets to pull the estimate off track. The algorithm should also search for the true target, and when found, re-associate the true target with the previous track. To reduce infeasible re-associations, a search region could be created that grows over time the longer the object has been occluded, until the object emerges from the occlusion and is re-identified (see Figure 6.1).

6.2.4 Improved Parallax-Compensated Motion Detection

The current parallax-compensated motion detection algorithm detects targets intermittently and occasionally returns a large number of false positives on certain frames of video sequences.
Two possible sources of these issues are low-quality video and poorly-calibrated cameras. Nister suggests that to obtain any benefits from using the essential matrix, the camera calibration parameters should be known to within about 10% (see [40], Figure 15). For cameras in our lab, we measure camera parameters using standard checkerboard calibration techniques. However, on video data for which we do not have access to the camera or the calibration parameters, the camera parameters are currently guessed, which can degrade performance. One potential area for improvement is to optimize camera parameters over an entire video sequence and then use these in the motion detection algorithm.

Another possible source of these issues is essential matrix model degeneracy. There are two cases of model degeneracy. First, as the translation magnitude between two consecutive frames approaches zero, the translation direction becomes unobservable. In this case, any translation direction can be used to create an essential matrix that appears to be valid. As a result, the algorithm may pick a translation direction that is in the same direction as one of the moving targets, thus causing the moving points on that target to be categorized as parallax-affected points. Second, in planar scenes there are two valid solutions to the essential matrix (see [40], see Section 4 and Table 1). If the incorrect solution to the essential matrix is used, a large portion of the points in the scene will be incorrectly labeled as moving points.

Another possible source of these issues is that only two frames are used to distinguish between stationary and moving points. If the direction of travel of the UA V and moving targets between two consecutive frames are parallel, the detections will disappear temporarily. Using multiple frames can reduce these effects. The persistent point tracking technique introduced in Chapter 2 may be useful in designing such a tracker. Ideas from 3D scene reconstruction could also be used with some modification in order to classify outliers to the 3D model as moving points. There are also various approaches presented in the literature, such as using more complex matrices to encode multi-frame constraints [13], [52].

### 6.2.5 Three-Dimensional Target Tracking Model

All tracking algorithms presented in this thesis track targets in the image plane in two dimensions. At each time step, a homography matrix is used to transform the position, velocity, and covariance of targets from the current frame to the next frame, as described in Chapter 4.
The main limitation with this approach is that the homography transformation is only valid for points and targets that exist on the planar surface represented by the homography. Target estimates that do not lie on this plane will be affected by parallax when the camera translates. Furthermore, targets moving with a constant velocity in the world frame are not necessarily constant velocity in the image plane, as explained in Section 4.6. These limitations can reduce tracking accuracy.

For these reasons, a three-dimensional model for moving targets has the potential to improve tracking accuracy. The main challenge in implementing a three-dimensional target model is that the target depth is unobservable from a monocular camera. There is a one degree of freedom ambiguity between the target’s velocity and its depth. One possible solution is to calculate the depth of moving targets from the depth of nearby stationary points in the scene, as is done in [54]. To do this, the depth of stationary features must also be estimated. This can be done in a robocentric frame, as is done in ROVIO [55], or in the world frame, as is done in SVO [56]. Tracking features in the world frame is significantly faster because the Kalman filter state propagation derivatives are simpler when represented in the world frame.
REFERENCES


[34] “Benchmark multiple object trackers (MOT) in python,” https://github.com/cheind/py-motmetrics, 2019. 31


104


APPENDIX A. DECOMPOSING THE ESSENTIAL MATRIX

For any essential matrix, there are two possible rotation matrices and a positive and a negative translation vector. These two rotation matrices $R_1$ and $R_2$ can be solved for from the essential matrix $E$ using the singular value decomposition (SVD) of $E$. This gives

\[
E = U \Sigma V^\top
\]

\[
= \begin{bmatrix}
  u_1 & u_2 & u_3
\end{bmatrix}
\begin{bmatrix}
  \sigma & 0 & 0 \\
  0 & \sigma & 0 \\
  0 & 0 & 0
\end{bmatrix}
\begin{bmatrix}
  v_1^\top \\
  v_2^\top \\
  v_3^\top
\end{bmatrix}.
\]

The essential matrix is defined up to a scale factor, so we can divide out the $\sigma$ term, so that

\[
E = \begin{bmatrix}
  u_1 & u_2 & u_3
\end{bmatrix}
\begin{bmatrix}
  1 & 0 & 0 \\
  0 & 1 & 0 \\
  0 & 0 & 0
\end{bmatrix}
\begin{bmatrix}
  v_1^\top \\
  v_2^\top \\
  v_3^\top
\end{bmatrix}
\]

\[
\frac{E}{\sigma} = \begin{bmatrix}
  u_1 & u_2 & u_3
\end{bmatrix}
\begin{bmatrix}
  1 & 0 & 0 \\
  0 & 1 & 0 \\
  0 & 0 & 0
\end{bmatrix}
\begin{bmatrix}
  v_1^\top \\
  v_2^\top \\
  v_3^\top
\end{bmatrix}.
\]

While $U$ and $V$ are orthogonal matrices, they might not be valid rotation matrices. A valid rotation matrix must also have a determinant of 1, which means the new rotated frame is still a right-hand coordinate frame. If the determinant of one of both of these matrices is $-1$, then the third vector in that matrix can be multiplied by $-1$ to make the matrix a valid rotation matrix. Changing the sign of $u_3$ or $v_3$ does not change the equation because the third singular value is
0. Equivalently, the vectors $u_3$ and $v_3$ are in the left and right null spaces, which means the only requirement is that they are orthogonal to the other unit vectors. Let

$$\alpha_u = |U|$$

and

$$\alpha_v = |V|.$$

Then the expression

$$\begin{bmatrix} u_1 & u_2 & \alpha_u u_3 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} v^\top_1 \\ v^\top_2 \\ \alpha_v v^\top_3 \end{bmatrix}$$

will be composed of two valid rotation matrices.

This form has two positive elements on the diagonal, but the essential matrix decomposition requires a skew symmetric matrix as one of the factors. If we swap $u_1$ and $u_2$ and make one of them negative, we can obtain a skew symmetric matrix in the center of the equation, giving

$$\frac{E}{\sigma} = \begin{bmatrix} -u_2 & u_1 & \alpha_u u_3 \\ 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} v^\top_1 \\ v^\top_2 \\ \alpha_v v^\top_3 \end{bmatrix}.$$

Let

$$R_u = \begin{bmatrix} -u_2 & u_1 & \alpha_u u_3 \end{bmatrix}$$

and

$$R_v = \begin{bmatrix} v_1 & v_2 & \alpha_v v_3 \end{bmatrix}.$$

Then

$$\frac{E}{\sigma} = R_u (e_3) \times R_v.$$
We can use the identity $(Rt)_{\times} = Rt \times R^\top$ to move the left rotation matrix to the right side of the equation. This gives

$$
= R_u (e_3)_{\times} R_u^\top R_u R_v \\
= (R_u e_3)_{\times} R_u R_v \\
= t_{\times} R_1,
$$

where

$$
\hat{t} = R_u e_3
$$

and

$$
R_1 = R_u R_v.
$$

If $U$ and $V$ are assumed to be valid rotation matrices, we can also rewrite the equation using a helper matrix $W$ to swap $u_1$ and $u_2$ and make one of them negative. Let

$$
W = \begin{bmatrix}
0 & 1 & 0 \\
-1 & 0 & 0 \\
0 & 0 & 1
\end{bmatrix},
$$

with

$$
W (e_3)_{\times} = \begin{bmatrix}
0 & 1 & 0 \\
-1 & 0 & 0 \\
0 & 0 & 1
\end{bmatrix} \begin{bmatrix}
0 & -1 & 0 \\
0 & 1 & 0 \\
0 & 0 & 0
\end{bmatrix} = \begin{bmatrix}
1 & 0 & 0 \\
0 & 1 & 0 \\
0 & 0 & 0
\end{bmatrix}.
$$
Then

\[
\frac{E}{\sigma} = U \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} V^\top = UW (e_3)_\times V^\top.
\]

Using the solution above, we obtain

\[
\hat{t} = UW e_3 = u_3
\]

and

\[
R_1 = UWV^\top.
\]

However, if \(U\) and \(V\) have not been corrected to be valid rotation matrices, \(R_1\) might have a determinant of -1. This can be corrected by multiplying \(R\) by negative 1. This will not change the essential matrix since it is defined up to a scale factor. The second rotation matrix can be found by making \(u_1\) negative instead of \(u_2\), which is equivalent to using \(W^\top\) instead of \(W\). This gives

\[
R_2 = UW^\top V^\top.
\]

These equations are a simple way to solve for \(R_1\) and \(R_2\) if the essential matrix has been solved for directly. However, if the essential matrix is already parameterized as the product of a skew-symmetric matrix and a rotation matrix, then there is a more direct method to solve for the second rotation matrix that does not require a decomposing the essential matrix.
APPENDIX B. SOLVING FOR $R_2$ FROM $R_1$ AND $\hat{t}$

For any essential matrix, there are two possible rotation matrices and a positive and negative translation vector. If one of the rotation matrices and the translation vector are already known, the second rotation matrix can be solved for from the first. Let us assume that $E$ is already factored as the product of a skew-symmetric matrix and a rotation matrix, so that

$$E = \hat{t} \times R.$$  \hfill (B.1)

The goal is to find the second possible rotation matrix, $R_2$. Let

$$E = \begin{bmatrix} e_1 & e_2 & e_3 \end{bmatrix}$$

and

$$R = \begin{bmatrix} r_1 & r_2 & r_3 \end{bmatrix}.$$  

Then (B.1) becomes

$$\begin{bmatrix} e_1 & e_2 & e_3 \end{bmatrix} = \hat{t} \times \begin{bmatrix} r_1 & r_2 & r_3 \end{bmatrix} = \begin{bmatrix} \hat{t} \times r_1 & \hat{t} \times r_2 & \hat{t} \times r_3 \end{bmatrix}.$$  

This gives three equations that must be satisfied for the essential matrix to be equivalent:

$$e_1 = \hat{t} \times r_1$$
$$e_2 = \hat{t} \times r_2$$
$$e_3 = \hat{t} \times r_3.$$
Observe that any translation unit vector \( \mathbf{\hat{t}} \) is in the nullspace of \( \mathbf{\hat{t}} \times \). Thus any scalar multiple of \( \mathbf{\hat{t}} \) can be added to any of the vectors \( \mathbf{r}_1 \), \( \mathbf{r}_2 \), and \( \mathbf{r}_3 \) without changing \( \mathbf{e}_1 \), \( \mathbf{e}_2 \), and \( \mathbf{e}_3 \):

\[
\begin{align*}
\mathbf{e}_1 &= \mathbf{\hat{t}} \times (\mathbf{r}_1 + \alpha_1 \mathbf{\hat{t}}) \\
\mathbf{e}_2 &= \mathbf{\hat{t}} \times (\mathbf{r}_2 + \alpha_2 \mathbf{\hat{t}}) \\
\mathbf{e}_3 &= \mathbf{\hat{t}} \times (\mathbf{r}_3 + \alpha_3 \mathbf{\hat{t}}).
\end{align*}
\]

However, a valid rotation matrix must be composed of unit vectors, so we also have the constraints

\[
\begin{align*}
\|\mathbf{r}_1 + \alpha_1 \mathbf{\hat{t}}\| &= 1 \\
\|\mathbf{r}_2 + \alpha_2 \mathbf{\hat{t}}\| &= 1 \\
\|\mathbf{r}_3 + \alpha_3 \mathbf{\hat{t}}\| &= 1.
\end{align*}
\]

This gives two solutions shown in Figure B.1. We can find \( \mathbf{r}'_i \) by subtracting the projection of \( \mathbf{r}_i \) onto \( \mathbf{\hat{t}} \) twice. This is known as the Householder transformation and gives

\[
\begin{align*}
\mathbf{r}'_i &= \mathbf{r}_i - 2 \text{proj}_\mathbf{\hat{t}} \mathbf{r}_i \\
&= \mathbf{r}_i - 2 \frac{\mathbf{r}_i \cdot \mathbf{\hat{t}}}{\|\mathbf{\hat{t}}\|^2} \mathbf{\hat{t}}.
\end{align*}
\]

Since the translation vector is a unit vector, this can be simplified to

\[
\begin{align*}
\mathbf{r}'_i &= \mathbf{r}_i - 2(\mathbf{r}_i \cdot \mathbf{\hat{t}}) \mathbf{\hat{t}} \\
&= \mathbf{r}_i - 2 \left( \mathbf{\hat{t}}^\top \mathbf{r}_i \right) \mathbf{\hat{t}}.
\end{align*}
\]
Figure B.1: Two solutions for the rotation matrix basis vectors

To visualize these vectors, imagine a plane with normal vector $\hat{r}$. The Householder transformation reflects vectors across this plane. This plane could also be thought of a mirror. The reflection of the basis vectors forming the rotation is a new matrix that produces the same essential matrix. However, it is impossible to rotate an object into the same configuration that exists on the opposite side of a mirror. It is a left-hand coordinate frame instead of a right-hand coordinate frame, and its determinant is -1 instead of 1. But if we take the negative of this orthogonal matrix we do get a valid rotation matrix. Multiplying the rotation matrix by -1 changes the essential matrix to be negative instead of positive, but still equivalent. Thus the basis vectors we are actually interested in are

$$ r'_i = -\left( r_i - 2 \left( \hat{r}^\top r_i \right) \hat{r} \right). \quad (B.2) $$
This can be seen in the Figure B.2. We are essentially reflecting the vector $r_1$ across the vector $\hat{t}$. The cross product of this new vector $r'_1$ and the translation vector $\hat{t}$ has the same magnitude because the angle magnitude and lengths of the vectors are the same. However, the angle sign is in the opposite direction, so the sign of the cross product will be negative instead of positive, thus resulting in a negative, but equivalent essential matrix.

\[
\theta' = \theta
\|
\begin{align*}
\theta_1' = \arccos(1 - 2\hat{t}^2)
\end{align*}
\]

Figure B.2: Two solutions for the rotation matrix basis vectors after enforcing the right-hand coordinate system constraint

Rotating the vector $r_1$ by 180 degrees about the vector $\hat{t}$ is the same as reflecting the vector $r_1$ across the vector $\hat{t}$. Thus an alternative way to write (B.2) using the Rodrigues formula is

\[
r'_i = \text{rot}(180, \hat{t}) r_i
= (I + \sin(180)\hat{t}_{\times} + (1 - \cos(180))\hat{t}_\times^2) r_i
= (I + 2\hat{t}_{\times}^2) r_i,
\]
which gives

\[ R' = (I + 2\hat{t}^2) R. \quad (B.3) \]

This matrix is guaranteed to be a valid rotation matrix because it is the product of two rotation matrices.