EVENT-BASED VISUAL-INERTIAL ODOMETRY USING SMART FEATURES

THESIS

Zachary P Friedel, First Lieutenant, USAF

AFIT-ENG-MS-20-M-021

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT-ENG-MS-20-M-021

EVENT-BASED VISUAL-INERTIAL ODOMETRY USING SMART FEATURES

THESIS

Presented to the Faculty

Department of Electrical and Computer Engineering

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

in Partial Fulfillment of the Requirements for the

Degree of Master of Science in Electrical Engineering

Zachary P Friedel, B.S.E.E.

First Lieutenant, USAF

March 26, 2020

AFIT-ENG-MS-20-M-021

EVENT-BASED VISUAL-INERTIAL ODOMETRY USING SMART FEATURES

THESIS

Zachary P Friedel, B.S.E.E.
First Lieutenant, USAF

Committee Membership:

Robert C Leishman, Ph.D
Chair

Joseph A Curro, Ph.D
Member

Clark N Taylor, Ph.D
Member

Abstract

Event-based cameras are a novel type of visual sensor that operate under a unique paradigm, providing asynchronous data on the log-level changes in light intensity for individual pixels. This hardware-level approach to change detection allows these cameras to achieve ultra-wide dynamic range and high temporal resolution. Furthermore, the advent of convolutional neural networks (CNNs) has led to state-of-the-art navigation solutions that now rival or even surpass human engineered algorithms. The advantages offered by event cameras and CNNs make them excellent tools for visual odometry (VO).

This document presents the implementation of a CNN trained to detect and describe features within an image as well as the implementation of an event-based visual-inertial odometry (EVIO) pipeline, which estimates a vehicle's 6-degrees-of-freedom (DOF) pose using an affixed event-based camera with an integrated inertial measurement unit (IMU). The front-end of this pipeline utilizes a neural network for generating image frames from asynchronous event camera data. These frames are fed into a multi-state constraint Kalman filter (MSCKF) back-end that uses the output of the developed CNN to perform measurement updates. The EVIO pipeline was tested on a selection from the Event-Camera Dataset [1], and on a dataset collected from a fixed-wing unmanned aerial vehicle (UAV) flight test conducted by the Autonomy and Navigation Technology (ANT) Center.

Table of Contents

Page

List of Figures

Figure Page

Figure Page

List of Tables

I. Introduction

1.1 Problem Background

Modern navigation solutions used in practical applications substantially rely on the use of the Global Positioning System (GPS), as when fully operational, no other technology can currently achieve a similar performance. However, this performance is dependent on receiving reliable, unobstructed signals in a time when these signals can easily be jammed or spoofed. To mitigate the risks GPS-only solutions present, in both military and commercial applications, the Autonomy and Navigation Technology (ANT) Center at the Air Force Institute of Technology (AFIT) has invested in a variety of alternative navigation solutions that aim to achieve performance on par with GPS. One of these research avenues, and the focus of this research, is visual odometry (VO) [2, 3], which utilizes visual sensors to estimate pose over time.

1.1.1 Classical Cameras and Feature Detection

VO requires a camera affixed to a moving vehicle whose imagery is utilized for detecting distinguishable landmarks, or features. These features are ideally tracked over multiple frames to estimate and/or correct the estimate of motion of the camera, and subsequently of the vehicle, relative to those features. Therefore, the performance of a VO solution depends on and is limited by the camera and the algorithm that detects and associates features. Cameras that produce images at a low frames-per-second (fps) rate can miss critical information between frames, especially during rapid

movement, while cameras with high fps output require more operational power and processing time, limiting their use in real-time scenarios. Other consequences for a low or high fps rate, respectively, include motion blur and redundant information. Additionally, the synchronous nature of a camera's pixel functionality causes them to be limited in dynamic range, meaning their ability to capture light and dark areas in the same image is restricted.

Until recently, the best techniques for detecting and matching features across image frames relied on carefully hand-crafted algorithms [4, 5, 6, 7, 8, 9]. Most of these methods exhibit excellent repeatability when the scale and viewpoint change, or the images are blurred. However, their reliability degrades significantly when the images are acquired at different illumination conditions and in different weather or seasons [10]. Over recent years, though, methods based in machine learning have started to outperform these traditional feature detection methods in all conditions [11, 12, 13, 14, 15].

1.1.2 Event-Based Cameras

Event-based cameras are a novel type of visual sensor that operate under a unique paradigm, providing asynchronous data on the log-level changes in light intensity for individual pixels. This hardware-level approach to change detection allows these cameras to achieve ultra-wide dynamic range and high temporal resolution, all with low power requirements, addressing some of the shortfalls of classical cameras. Furthermore, the data rate is reflective of the changes in the event camera's field of view, with rapid movement and/or highly textured scenes generating more data than slow movement and/or uniform scenes. However, due to the unique output provided by event cameras, it cannot be directly used in proven VO pipelines and requires manipulation beforehand.

1.2 Research Objectives

The primary goal of this research is to estimate a vehicle's 6-degrees-of-freedom (DOF) pose changes utilizing an affixed event-based camera with an integrated inertial measurement unit (IMU) and features and descriptors generated from a neural network. Consequently, a secondary objective is separately training and evaluating the neural network, specifically a convolutional neural network (CNN) inspired by the work in [11], to reliably detect and match features across challenging viewpoint and illumination changes. The goal of this secondary objective is to develop a feature detector that generalizes well across various conditions to overcome some of the deficiencies of classical algorithms, with an emphasis on downward-facing, event-based aerial imagery.

The overall method consists of a front-end neural network developed by Rebecq et. al [16, 17] that generates image frames from event camera output. The CNN developed in this research then feeds matched features from these images to a back-end that uses an extended Kalman filter (EKF) to estimate the system states. Specifically, the filter is an implementation of the multi-state constraint Kalman filter (MSCKF) influenced by the work in [18] with adjustments inspired by [19]. This event-based visual-inertial odometry (EVIO) pipeline is subsequently evaluated on multiple datasets: a selection from the Robotics and Perception Group (RPG) Event-Camera Dataset [1] and a portion of a fixed-wing unmanned aerial vehicle (UAV) flight test conducted by the ANT Center.

1.3 Document Overview

This document is organized as follows. Chapter II provides an overview of relevant background information, including the basics of VO and a summary of its limitations, an overview of the operating concept behind event-based cameras and recent research

into these sensors, and the foundation of machine learning including current VO research in this field. Chapter III details the process of training a CNN that produces feature points and descriptors, and the method of evaluating this network. Chapter III also describes the MSCKF algorithm and the datasets used to analyze the entire EVIO pipeline. Chapter IV presents the results of evaluating variations of the feature detector/descriptor network, as well as the results of the EVIO pipeline. Finally, Chapter V discusses the conclusions drawn from the results, and possible adjustments to the methodology that can improve each aspect of this research.

II. Background and Literature Review

This chapter provides the foundational concepts and literature required for understanding subsequent chapters. It is organized as follows: Section 2.1 covers the basics of visual odometry (VO), including a summary of its limitations. Section 2.2 describes the operating concept for event-based cameras and a summary of recent research with these sensors. Section 2.3 explains the theory behind artificial neural networks (ANNs) and illustrates their usefulness for VO through recent research.

2.1 Visual Odometry

Visual odometry (VO), coined in 2004 by Nister [20] for its similarity to the concept of wheel odometry, is the process of estimating the egomotion of a body (e.g., vehicle, human, or robot) using visual information from one or more cameras attached to the body. Although some of the first applications of VO in the early 1980's dealt with offline implementations, such as Moravec's groundbreaking work that presented the first motion estimation pipeline whose main functioning blocks are still used today [21], only in the early 21st century did real-time working systems flourish, leading to VO proving useful on the Mars exploration rovers [22]. Currently, VO has proven a useful supplement to other navigation systems such as Global Positioning System (GPS), inertial measurement units (IMUs), and laser odometry [2].

2.1.1 Camera Model

The basis of VO requires modeling the 3-D geometry of the real world onto a 2-D image plane. The most common method used is perspective projection with a pinhole camera model, in which the image is formed by projecting the intersection of light rays through a camera's optical center onto a 2-D plane [2], as seen in Figure 1.

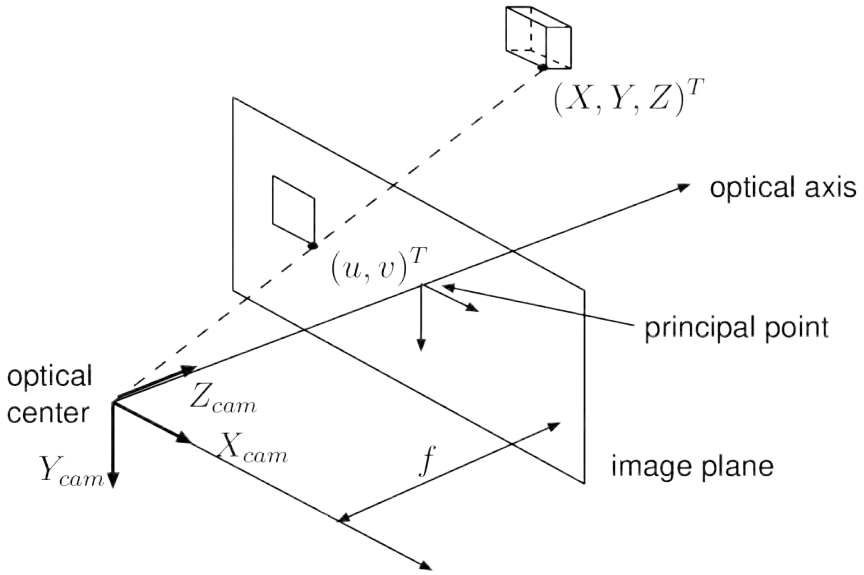

Figure 1: Perspective Projection Pinhole Camera Model: 3-D objects in space are projected through an optical center onto a 2-D image plane.

This allows the mapping of 3-D world coordinates $[X, Y, Z]^T$ in the camera's reference frame to 2-D pixel coordinates $[u, v]^T$ via the perspective projection equation

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = K \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \frac{f}{Z} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \frac{f}{Z} \tag{1}$$

where f/Z is the depth factor used to convert to normalized coordinates (generally, $f = 1$), f_x and f_y are the focal lengths, K is the camera calibration matrix, and c_x and c_y are the pixel coordinates of the projection center [2].

However, Equation (1) does not solve the issue of radial distortion caused by the camera lens, which can be modeled using a higher order polynomial. The derivation of this model, which explains the relationship between distorted and undistorted pixel coordinates, can be found in computer vision textbooks such as [23].

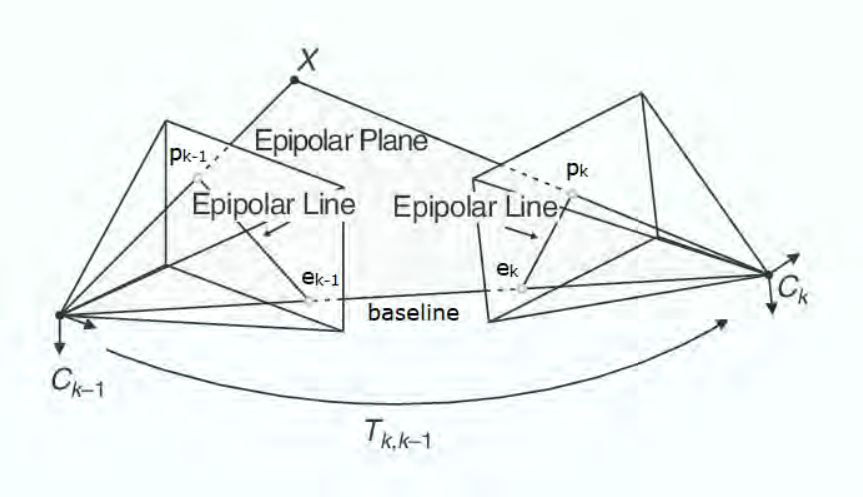

Figure 2: Epipolar Geometry uses camera optical centers and points seen in space by multiple images to form an epipolar plane which provides the means to estimate pose change between two coordinate frames (k-1, k).

2.1.2 Epipolar Geometry

Epipolar geometry, illustrated in Figure 2, is defined when a specific point in space (X) is visible in two or more images and is used to estimate the 6-degrees-of-freedom (DOF) pose change between camera coordinate frames at discrete time instances $k-1$ and k. C_{k-1} and C_k represent the optical centers of the camera and the baseline between them intersects each image plane at epipoles e_{k-1} and e_k. The epipolar plane contains the object point X in 3-D space as well as the normalized coordinate projections of that point, p_{k-1} and p_k on the successive image planes. An essential matrix $\mathbf{E}$ condenses the epipolar geometry between the images providing a relationship between image points in one image with epipolar lines in the corresponding image. In doing so the essential matrix also provides a direct relationship between the image points p_{k-1} and p_k known as the epipolar constraint, which is represented as the function $p_k^T \mathbf{E} p_{k-1} = 0$ [2]. Through singular value decomposition (SVD), the essential matrix

contains the camera motion estimate up to a multiplicative scalar of the form

$$\mathbf{E} \simeq \left[\mathbf{t}_k^{k-1}\times\right] \mathbf{R}_{k-1}^k \tag{2}$$

where, if $\mathbf{t}_k^{k-1} = [t_x, t_y, t_z]^T$ is the translation vector that describes the location of C_{k-1} in the k reference frame, then $\left[\mathbf{t}_k^{k-1}\times\right]$ is a skew symmetric matrix of the form

$$\left[\mathbf{t}_k^{k-1}\times\right] = \begin{bmatrix} 0 & -t_z & t_y \\ t_z & 0 & -t_x \\ -t_y & t_z & 0 \end{bmatrix} \tag{3}$$

and $\mathbf{R}_{k-1}^k$ is the direct cosine matrix (DCM) that rotates points in the $k-1$ reference frame into the k reference frame [24].

2.1.3 Feature Detection and Description

To identify points that can be matched across multiple images, unique visual characteristics, called features, are required [2, 3, 23]. The most common types of features extracted for VO are point detectors, such as corners or blobs, since their position in an image can be measured accurately [4, 5, 6, 7, 8, 25]. Corners are defined by a point at the intersection of two or more edges, while a blob is an image pattern that differs from its surroundings in terms of intensity, color and texture. Good feature detection is defined by localization accuracy, repeatability, computational efficiency, robustness to noise, and invariance to photometric and geometric changes [3]. Scale invariant feature transform (SIFT) [4] is a popular blob detector that blurs the image by convolving it with 2-D Gaussian kernels of various standard deviations, taking the differences between these blurred images and then identifying local minima or maxima. On the other hand, the Harris [8] detector is a popular corner detector that

uses a corner response function to find points greater than a specified threshold and selects the local maxima.

Once features are detected, the next step is to convert the region around these features into compact descriptors that are used to match the features across multiple images. One of the most widely used descriptors is the SIFT [4] descriptor, which decomposes the region around a feature into a histogram of local gradient orientations, forming a 128 element descriptor vector. SIFT suffers from long computation time, however, so other descriptors were developed to address this issue such as binary robust independent elementary features (BRIEF) [9], a binary descriptor that uses pairwise brightness comparisons sampled from the region around a feature.

There are many other feature detectors and descriptors that exist such as Shi-Tomasi [25], features from accelerated segment test (FAST) [7], speeded up robust features (SURF) [5], oriented FAST and rotated BRIEF (ORB) [6] and binary robust invariant scalable keypoints (BRISK) [26]. Each one has its pros and cons, so the appropriate selection of which algorithms to use highly depends on computational constraints, real-time requirements, environment type, etc.

2.1.4 Motion Estimation

Once features are identified in sequential images and matched to obtain an estimate of the essential matrix, the rotation matrix and translation vector can be extracted as described in Section 2.1.2. These values are used to form the Special Euclidean (SE(3)) transformation matrix described in Equation (4).

$$\mathbf{T}_{k-1}^{k} = \begin{bmatrix} \mathbf{R}_{k-1}^{k} & t_{k}^{k-1} \\ \mathbf{0}_{1\times 3} & 1 \end{bmatrix} \tag{4}$$

Assuming a set of camera poses C_k $(k = 0, ..., n)$, the current pose C_n with respect

to the starting pose C_0 can be determined by multiplying the transformation matrices, $\mathbf{T}_{k-1}^{k}$ $(k = 1, ..., n)$ to obtain $\mathbf{T}_0^n$ which contains the rotation and translation between the origin pose and current pose [2]. Obtaining the correct translation (i.e. the translation that describes the location of C_n in the $k = 0$ reference frame) can then be computed by transforming $\mathbf{t}_n^0$ to $\mathbf{t}_0^n$ via

$$\mathbf{t}_0^n = (\mathbf{R}_0^n)^T \cdot -\mathbf{t}_n^0. \tag{5}$$

However, since the essential matrix is only a scaled representation of the relation between two poses which results in a homogeneous translation vector, other methods are required to obtain properly scaled transformations. For monocular VO, one method of obtaining scale is to triangulate matching 3-D points X_{k-1} and X_k from subsequent image pairs. From the corresponding points, the relative distances between any combination of two points i and j can be computed via

$$r = \frac{||X_{k-1,i} - X_{k-1,j}||}{||X_{k,i} - X_{k,j}||}. \tag{6}$$

The relative distances for many point pairs are computed and the median or mean is used for robustness, and the resulting relative scale is applied to the translation vector $\mathbf{t}_{k-1}^{k}$ [2]. Scale can also be obtained by incorporating measurements from other sensors such as GPS, IMUs, light detection and ranging (LIDAR), etc [18, 19, 27, 28].

2.1.5 Limitations

For VO to work efficiently, sufficient illumination and a static scene with enough texture should be present in the environment to allow apparent motion to be extracted [2]. In areas that have a smooth and low-textured surface floor, directional sunlight and lighting conditions are highly considered, leading to non-uniform scene lighting.

Moreover, shadows from static or dynamic objects or from the vehicle itself can disturb the calculation of pixel displacement and thus result in erroneous displacement estimation [29, 30]. Also, small errors in the camera model and calculation of rotation and translation, described in Section 2.1.1 and Section 2.1.2, accumulate over time leading to a drift in the pose estimation.

Additionally, the quality of VO in high-speed scenarios is constrained by the temporal resolution of frame-based cameras due to the missed information between images from a limited frame rate. Increasing the frame rate to capture more of this information not only requires more operational power, but also more time to process increased amounts of data. Even high-quality frame-based cameras suffer from some level of motion blur in high-speed scenarios. Furthermore, it is impossible to instantaneously measure the light intensity with frame-based cameras, which require some length of integration or exposure time. The exposure time impacts the ability of the camera to capture both high-illuminated and low-illuminated objects in the same scene, i.e. high dynamic range (HDR) [31, 32]. These issues with VO also restrict the performance of other machine vision applications, like object recognition/tracking or robotic controls. These shortcomings led to the investigation of event-based cameras, whose advantages are described in Section 2.2.

2.2 Event-Based Cameras

Event-based cameras, also known as neuromorphic cameras or dynamic vision sensors (DVSs), are a novel approach to capturing visual information and are based on the development of neuromorphic electronic engineering, which aims to develop integrated circuit technology inspired by the brain's efficient data-driven communication design [33]. These cameras are the result of efforts to emulate the asynchronous and continuous-time nature of the human visual system [34]. Event cameras are effective

in addressing the common performance limitations in VO and other machine vision applications using traditional frame-based cameras [35, 36, 37].

2.2.1 Operating Concept

Event cameras operate by an array of complementary metal-oxide semiconductor (CMOS) pixels, seen in Figure 3, asynchronously and independently measuring light intensity in a continuous fashion. The output of this sensor at a given pixel is an event e, triggered by a log-level change in light intensity at a specified change threshold,

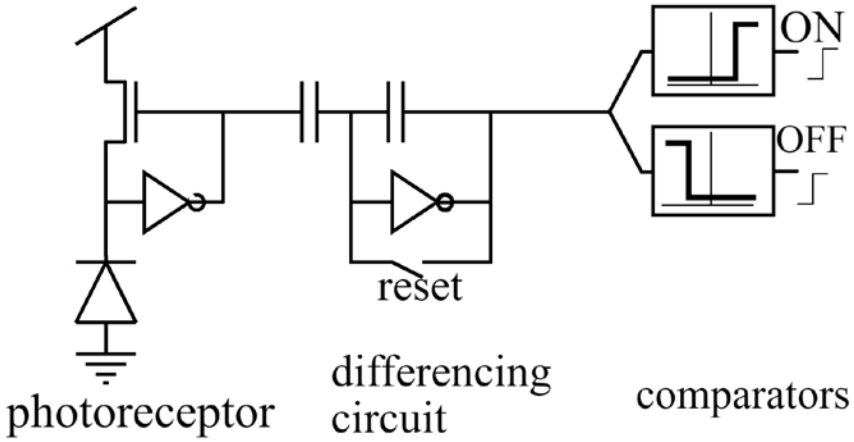

Figure 3: Abstract Schematic of Event Camera Pixel: The photoreceptor continuously measures incoming light intensity, the differencing circuit amplifies the changes and the comparators output a ON or OFF signal for a rise or fall in intensity, respectively.

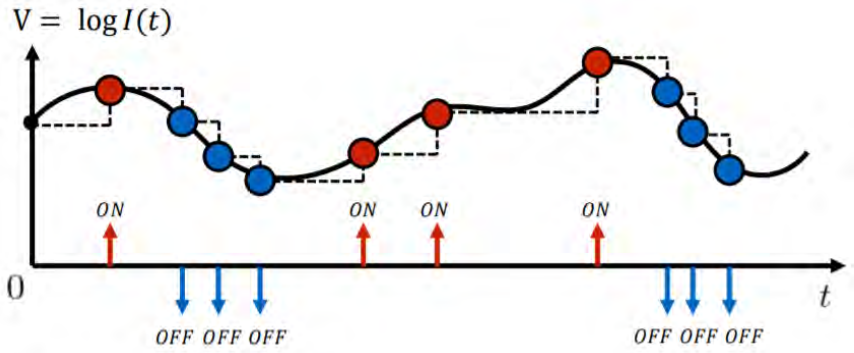

Figure 4: Event Camera Principle of Operation: The camera outputs an ON or OFF event corresponding to a rise or fall in the log-level light intensity past a specified threshold.

which is characterized by a tuple including the time stamp t, the pixel coordinates (x, y) and the event polarity p. The event polarity is a binary value that represents an ON/OFF or a rise/fall in light intensity [38], illustrated in Figure 4.

This unique operating paradigm allows event cameras to be data-driven sensors (their output depends on the amount of motion or brightness change in a scene), as opposed to standard cameras which output images at a constant rate (e.g. 30 fps), and results in event cameras offering significant advantages. Their hardware level approach to sensing light intensity changes allows for a very high temporal resolution and low latency (both in the order of microseconds) [31]. Event cameras can therefore capture very fast motions without motion blur typical of frame-based cameras. Figure 5 illustrates the high temporal fidelity of an event camera as well as the effect of rapid versus slow movement in a scene on the amount of events produced [39].

Furthermore, the independent nature of event camera pixels allows for a HDR on the order of 140 dB, compared to 60 dB for standard cameras. This enables them to adapt to very dark and very light stimuli simultaneously, not having to wait for a

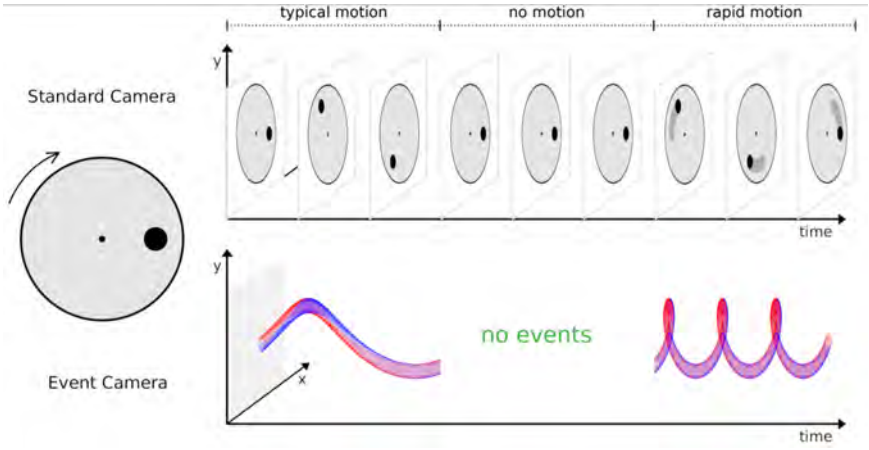

Figure 5: Standard vs. Event Camera: The high temporal resolution of event cameras allow them to capture missing information between successive image frames. Additionally, they do not produce redundant data when there is no motion. [39].

global shutter, which is illustrated in Figure 6 [31]. Due to event cameras transmitting only brightness changes, they also offer lower power requirements (10 mW [31]) and lower bandwidth requirements (200 Kb/s on average [32]).

However, since the output of event cameras is fundamentally different than that of conventional cameras, frame-based vision algorithms designed for image sequences are not directly applicable. Thus, novel algorithms are required to process event camera output and unlock the advantages the sensor offers.

2.2.2 Current Research with Event Cameras

To directly utilize VO algorithms that work with frame-based images, most research into event cameras has involved using the event data output to reconstruct scene intensity into a "psuedo-frame" that resembles a traditional image [16, 17, 40, 41, 42]. Early methods involved processing a spatio-temporal window of events that included obtaining logarithmic intensity images via Poisson reconstruction, simultaneously optimising optical flow and intensity estimates within a fixed-length, sliding spatio-temporal window using a preconditioned primal-dual algorithm, and integrating events over time while periodically regularising the estimate on a manifold defined by the timestamps of the latest events at each pixel [40, 43, 44, 45]. However, taking a spatio-temporal window of events imposes a latency cost, and choosing a time-

Figure 6: High Dynamic Range of Event Cameras: The classical camera is unable to capture the shadowed area (left). The other three images represent the output of different image reconstruction algorithms using batches of events. These event-based images show that an event-camera is more resilient to lighting differences which allow it to detect objects throughout the entire scene.

interval or event batch size that works robustly for all types of scenes is non-trivial. Therefore, the intensity images reconstructed by the previous approaches suffer from artifacts as well as lack of texture due to the spatial sparsity of event data.

Building on these methods to improve image reconstruction quality, other research has made use of IMUs by taking into account the specific event time and the estimated pose through inertial navigation system (INS) mechanization to generate event images that compensate for camera movement, resulting in scenes with stronger contrasting edges [46, 47].

More recent and state-of-the-art methods incorporate the use of machine learning to train models that use event timestamps, pixel location and polarity to output high quality image reconstructions [16, 48].

Additionally, there has been plenty of research that has addressed estimating ego-motion through the use of front-end event cameras. This has included methods that directly utilize event camera output [49, 50, 51], and more recent methods that incorporate IMU information to accomplish event-based visual-inertial odometry (EVIO) [47, 52], with Hybrid-EVIO utilizing both event-based camera data and intensity images, improving robustness in both stationary and rapid-movement scenarios [53].

2.3 Machine Learning

Machine learning involves techniques that allow a computer to calculate a model when the explicit form of the model is unknown, by extracting patterns from empirical data [54]. There are two main classes of machine learning, supervised and unsupervised. This section will focus on supervised learning, in which a model is formed by learning to map the patterns from input data to output data given example input-output combinations [55]. Unlike equations derived from theory whose parameters offer inference into how the input affects the output, the parameters learned by com-

plex machine learning models, such as the artificial neural network (ANN) model type, often have little meaning. Although machine learning only started to flourish in the 1990s [56], it has become increasingly popular due to its ability to solve and generalize specific problems better than human engineered algorithms.

2.3.1 Artificial Neural Networks

ANNs are a complex and robust type of machine learning model that are able to approximate some function $f(\cdot)$ given the model has enough neurons, or perceptrons [54]. Perceptrons are the fundamental units of an ANN that take one or multiple inputs and calculate the dot product between those inputs and a set of weights. This value is then fed through an activation function to output a single value. Since most complex problems are non-linear in nature, models usually utilize non-linear activation functions, as they allow the model to create elaborate mappings between the network's inputs and outputs, which is essential for learning and modeling complex data. The most common type of non-linear activation function is the rectified linear unit (ReLU) which simply multiplies negative input values by zero.

The inputs to a perceptron can either be the model inputs or the outputs of other perceptrons. This concept leads to the idea of multilayer perceptrons (MLPs), which are the quintessential deep learning models [54]. MLPs, the most common type of ANN architecture, has perceptrons arranged in layers where each perceptron in a layer has as its input the output of every perceptron in the previous layer, as illustrated in Figure 7. Layers of perceptrons not acting as the input or output layer are called hidden layers. Combined with non-linear activation functions and a large enough architecture, MLPs are able to approximate any function [57].

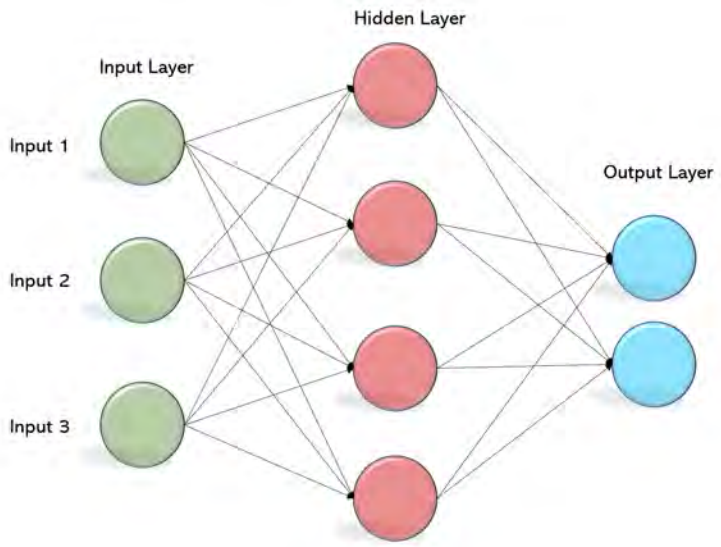

Figure 7: Basic MLP Architecture: The output of each perceptron acts as the input to perceptrons in the next layer.

2.3.1.1 Training

To obtain a model that approximates some function successfully, the weights of each perceptron must be "learned", which is accomplished through a variant of stochastic gradient descent (SGD) such as Adam [58] or RMSProp [59]. These variants, also known as optimizers, determine how the weights will be updated based on the loss function [56]. A loss function determines how well an estimate matches the target or "truth" data. The optimizer takes the derivative of a loss function with respect to all the weights in the model, then updates the weights in the negative direction of the gradient to minimize the loss function. Finally, a learning rate is multiplied by the gradient to control the magnitude of the weight change for each update [56]. Training is usually accomplished by performing SGD on batches of the dataset, while iterating over the dataset multiple times, or epochs, in order create a model with the highest potential of approximating the desired function.

2.3.1.2 Generalization

Training an ANN can be misleading in the sense that the error between the estimate and target will almost always decrease as training continues. However, the true measure of a model is how well it performs on data it has never seen before. This is why most machine learning processes utilize three different datasets: training, validation and test. The training set is used for updating the weights while the validation set is used for hyper-parameter selection, such as the number of layers, learning rate, optimizer, etc. The test set is therefore used to determine the generalization ability of the model and how close it is to approximating an objective function [56]. The validation set can additionally be used to determine if a model is over-fitting during the training process, which is a common problem in machine learning. Over-fitting occurs when there is over-optimization on the training data, and the model ends up learning representations that are specific to the training data and do not generalize to data outside of the training set [56]. The validation set can therefore be used to determine when over-fitting occurs as the error on this set will stop decreasing as the model begins to over-fit.

2.3.2 Convolutional Neural Networks

Convolutional neural networks (CNNs) are a specialized type of ANN for processing data that has a grid-like topology, such as images, which are essentially just 2-D or 3-D grids of pixels. Contrary to traditional neural networks which connect every input unit to every output unit, CNNs have sparse interactions due to their use of a weighted kernel that convolves or transforms patches of the input via tensor product operations. This allows a CNN to perform fewer operations when computing the output and store fewer parameters, which reduces the memory requirements of the model and improves its statistical efficiency compared to a fully connected network

[54]. When the number of input units (i.e. pixels in the case of images) is in the hundreds of thousands or millions, this becomes highly significant.

CNNs operate by sliding a kernel, which is a small window typically of size 3×3, over a 3-D image tensor, with two spatial axes (height and width) and a channel axis, and applying the same transformation to all possible 3×3 patches. The channel axis often has a depth of three or one, corresponding to an RGB or grayscale image, respectively. The output of each transformation is a vector, which are then spatially reassembled into a 3-D output tensor of size $height \times width \times output\ depth$, similar to the input. The difference is the channels axis is now a filter axis (the output depth being a design parameter) where each filter represents an aspect of the input data. At a low level, this could be the presence of corners in an image or at a high level, the presence of a face [56]. Figure 8 illustrates the process just described.

This unique operating concept allows CNNs to learn local patterns in the input images as apposed to global patterns learned by fully connected networks. Consequently, the patterns CNNs learn are translation invariant, meaning once they learn a pattern in one part of an image they can recognize it in another part without having to relearn the pattern, therefore requiring fewer training samples to learn representations. Additionally, CNNs can learn spatial hierarchies of patterns, meaning they can learn increasingly complex and abstract visual patterns. These two properties are the main reason CNNs are the most universally used tool in computer vision applications today [56].

In addition to activation functions common in all neural networks, CNNs almost always contain pooling operations after one or more convolutional layers, whose goal is to make the representations learned by a CNN become invariant to small translations of the input by downsampling the output of the convolutional layers [54]. Pooling is similar to convolutional operations except that instead of transforming small patches

via a learned transformation each window is transformed via a hard coded tensor operation, such as taking the maximum or average value. Pooling is an important tool as it allows for the reduction of the number of parameters to process, and induces spatial-filter hierarchies by making successive convolution layers look at increasingly large windows in terms of the fraction of the original input [56].

Finally, batch normalization layers are also a common tool in any ANN. These layers are able to adaptively normalize data even as the mean and variance change over time during training, helping with the gradient propagation and thus allowing for deeper networks [56].

2.3.3 Current Visual Odometry Research with Neural Networks

Since the basis of common VO techniques requires reliably detecting and matching features across images, a plethora of machine learning studies in the field of

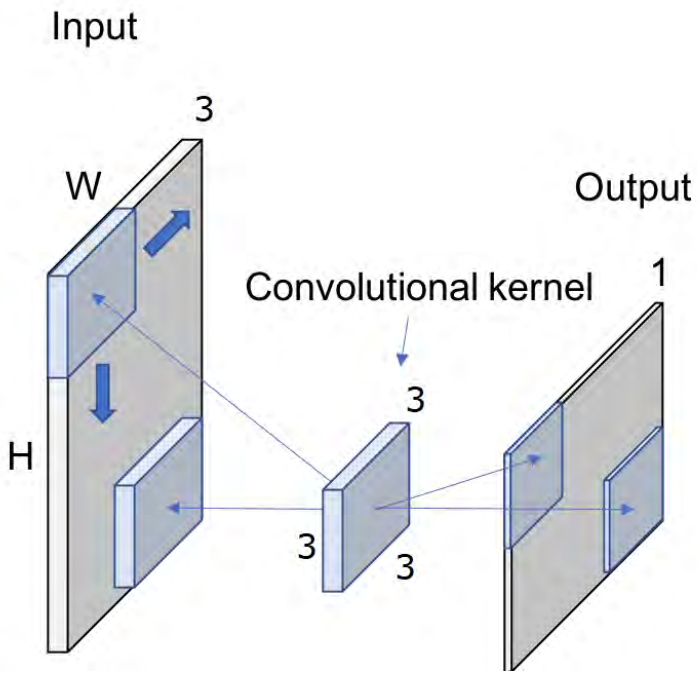

Figure 8: A convolutional kernel transforms patches of the input and spatially reassembles them. In this figure, the output depth or number of filters is one.

VO have focused on training neural networks to accomplish that task, leading to state-of-the-art results. A few studies that follow the interest point detection and description method closely are SuperPoint [11], D2-Net [60] and learned invariant feature transform (LIFT) [15], the latter however requiring multiple networks to accomplish different tasks and additional supervision from a Structure from Motion system. Other methods that are similar in their ability to match image substructures are universal correspondence network (UCN) [61] and DeepDesc [14], however they do not explicitly perform interest point detection.

On the other hand, studies have focused on building an end-to-end VO models that estimate egomotion directly such as in [62, 63, 64, 65]. These methods are unique compared to the methods mentioned previously in that they are not fully-convolutional, utilizing fully connected and recurrent layers in addition to convolutional layers. Furthermore, they do not use IMU data to support their pose estimation solution, a common and essential tool for state-of-the-art VO algorithms, which limits their usefulness.

III. Methodology

Preamble

The methodology described in this chapter is inspired by a combination of research to build an event-based visual-inertial odometry (EVIO) pipeline that utilizes features and descriptors from a neural network to feed into a back-end multi-state constraint Kalman filter (MSCKF). The front-end used in this work applies a recurrent-convolutional neural network (RCNN) to reconstruct videos from a stream of event data [16, 17], whose output is subsequently fed into a fully-convolutional network constructed similarly to the SuperPoint model [11]. This convolutional neural network (CNN) outputs features and descriptors, which along with data from an inertial measurement unit (IMU), is fed into a MSCKF inspired by the works of Mourikis et al. [18] and Sun et al. [19]. Section 3.1 describes the programming tools used throughout this research. Section 3.2 explains the method of training and evaluating the CNN, while Section 3.3 details the implementation of the MSCKF. The EVIO pipeline was tested on a select dataset from the Robotics and Perception Group (RPG) Event-Camera Dataset [1], as well as on a dataset collected with a dynamic and active-pixel vision sensor (DAVIS) 240C event camera on a fixed-wing unmanned aerial vehicle (UAV) flight test conducted at Camp Atterbury, Indiana. Additional details on these datasets is described in Section 3.4.

3.1 Programming Platform

The Python programming language [66] was used with multiple specialized libraries throughout this work mainly for its compatibility with the TensorFlow framework [67]. Specifically, TensorFlow version 1.13.1 and its implementation of Keras [68] was utilized for training and employing the neural networks used in this study.

TensorFlow's implementation of Keras is TensorFlow's high-level application programming interface (API) for building and training deep learning models and is used for fast prototyping, state-of-the-art research, and production [67].

3.2 CNN Design

As mentioned previously, the design of the CNN architecture that detects and describes feature points is primarily inspired by the SuperPoint model [11] with several adjustments made in the training process in an attempt to improve performance on an event camera dataset taken from a fixed-wing UAV.

3.2.1 Training Datasets

Training the full model required multiple intermediate rounds of training where each round utilized a different dataset. Additionally, two overall methods were studied, which differed in the datasets used for training.

The first method followed the steps described by DeTone et al. [11], which first involved developing a synthetic dataset consisting of 66,000 unique grayscale geomet-

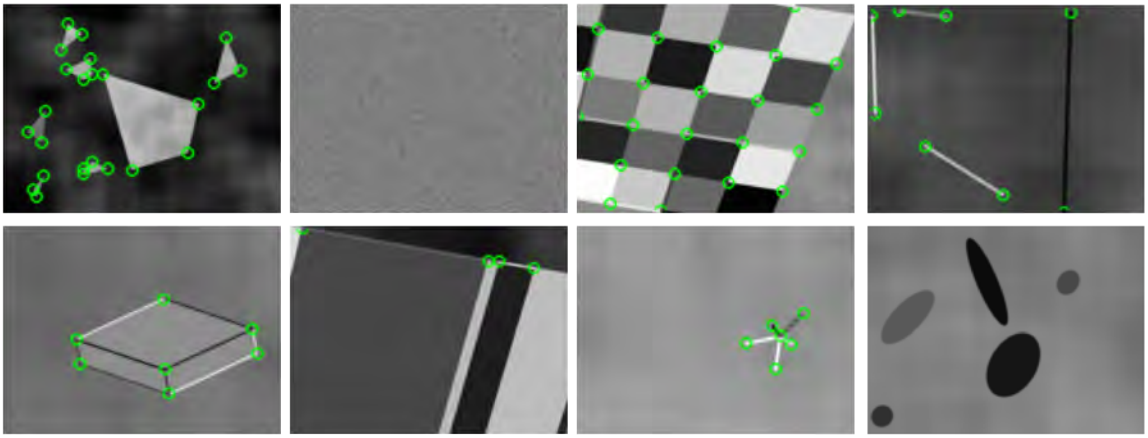

Figure 9: Synthetic Shapes Dataset: Examples of "shapes" in the synthetic dataset with corresponding ground truth locations. From top left to bottom right: polygons, Gaussian noise, checkerboard, lines, cube, stripes, star, ellipses.

ric "shapes" with no ambiguity in the interest point locations, or target labels. The shapes in this dataset include checkerboards, cubes, ellipses, lines, polygons, stars, stripes and gaussian noise. Each shape is represented by 10,000 variations except for ellipses, stripes and Gaussian noise which are represented by 3,000, 2,000 and 1,000 variations, respectively. Ellipses and Gaussian noise do not include interest point labels to help the network learn to ignore noise and blobs. The number of stripes variations was reduced due to the lower amount of labels compared to other shapes. An illustration of each shape in the dataset can be seen in Figure 9. Subsequent models were trained using the MS-COCO 2014 dataset (converted to grayscale), which includes 82,783 images of "complex everyday scenes containing common objects in their natural context" [69].

The second method is similar to the first except ∼20% of the MS-COCO dataset

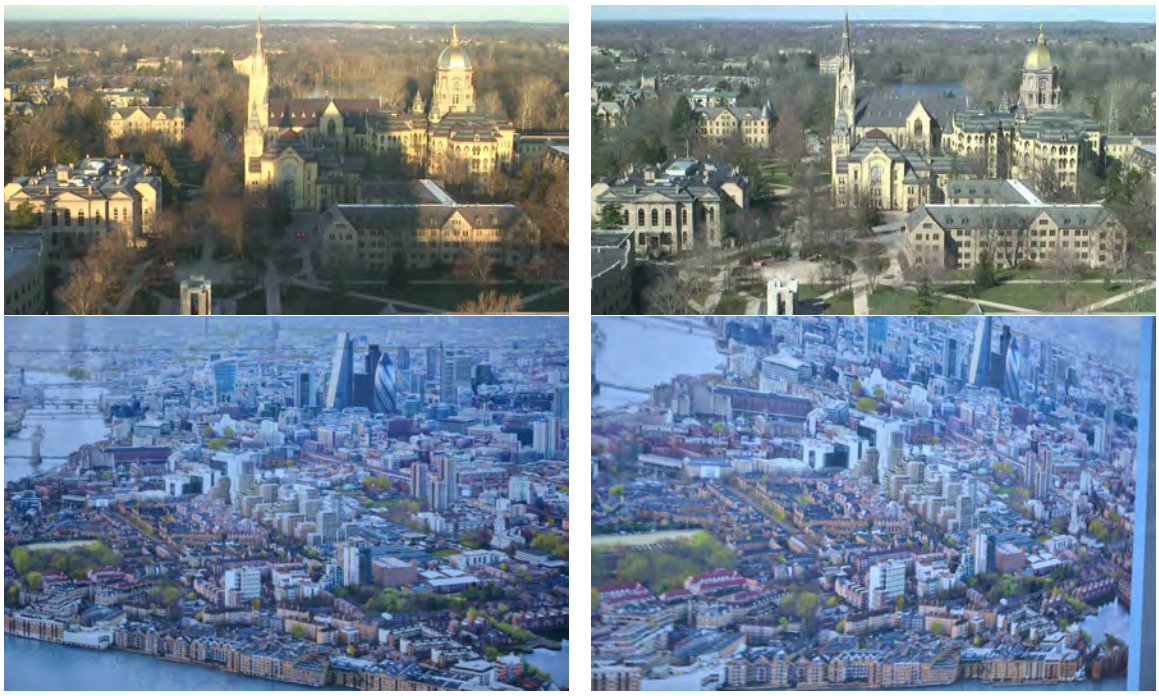

Figure 10: HPatches Dataset: Example images from the HPatches dataset. The top two represent an illumination change while the bottom two represent a viewpoint change.

was replaced with reconstructed images from event data taken aboard the UAV flight test, described in more detail in Section 3.4, to tailor the full model's performance to event-based aerial imagery.

3.2.2 Validation Datasets

The validation sets simply consisted of 3,200 held out images from their respective datasets. Contrary to the training sets, the validation set images were not modified in any way during pre-processing, as described in Section 3.2.5.

3.2.3 Evaluation Datasets

Two evaluation datasets were used to determine the model's ability to repeatably detect features across illumination and/or viewpoint changes and its ability to correctly estimate a homography change. The first dataset used was HPatches [70], which contains 116 scenes with each scene containing a reference image and 5 other images taken at different viewpoint or illumination conditions for a total of 696 unique images. A ground truth homography relates each image in a scene to the reference image. The first 57 scenes exhibit changes in illumination and the other 59 scenes have viewpoint changes, as illustrated in Figure 10. HPatches was used as a comparison to the work in [11], since that is what was used for their evaluation. The second dataset consists of 786 scenes from the reconstructed event images taken aboard the UAV, illustrated in Figure 11. Each scene is made up of a base image and a warped version of that image created by a random homography selection limited to small changes in rotation and translation. This second dataset was utilized for the evaluation of the CNN model on event-based aerial imagery.

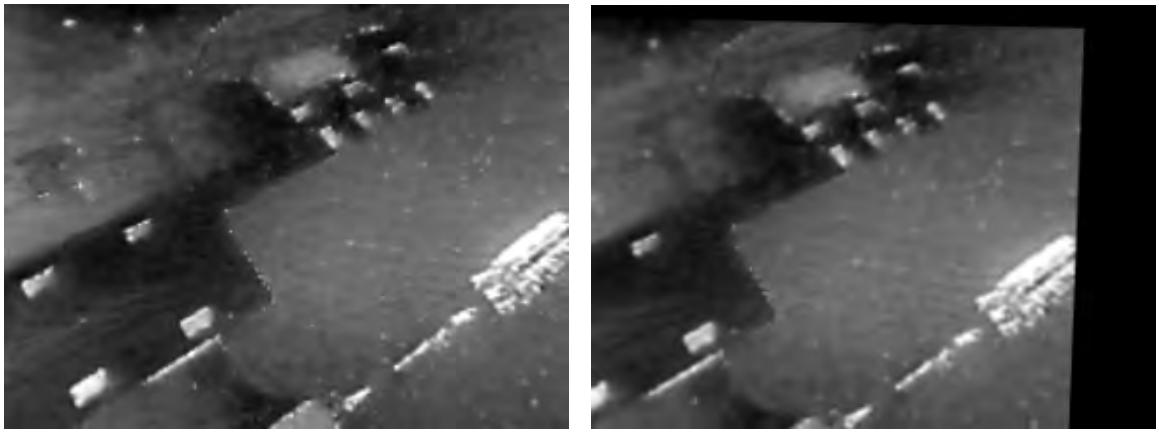

Figure 11: UAV Dataset: Example image pair from the UAV event-based imagery dataset.

3.2.4 Model Architecture

A fully convolutional neural network which operates on a full sized grayscale images and produces interest point detections accompanied by fixed length descriptors in a single forward pass was developed. The model has a single, shared encoder to process and reduce the input image dimensionality. After the encoder, the architecture splits into two decoder heads, which learn task specific weights, one for interest point detection and the other for interest point description.

The encoder consists of eight convolution layers that use 3×3 kernels and have output depths of 64-64-64-64-128-128-128-128. Every two layers − for the first six convolutional layers − there is a 2×2 max pooling layer that operates with a stride of two for a total of three max pooling layers. The detector head has a single 3×3 convolutional layer of depth 256 followed by a 1×1 convolutional layer of depth 65. The descriptor head has a single 3×3 convolutional layer of depth 256 followed by a 1×1 convolutional layer of depth 256. All convolution layers in the network are followed by rectified linear unit (ReLU) non-linear activations and batch normalization layers. An illustration of the CNN architecture is shown in Figure 12.

Given that an input image has a height, H, and a width, W, the output of

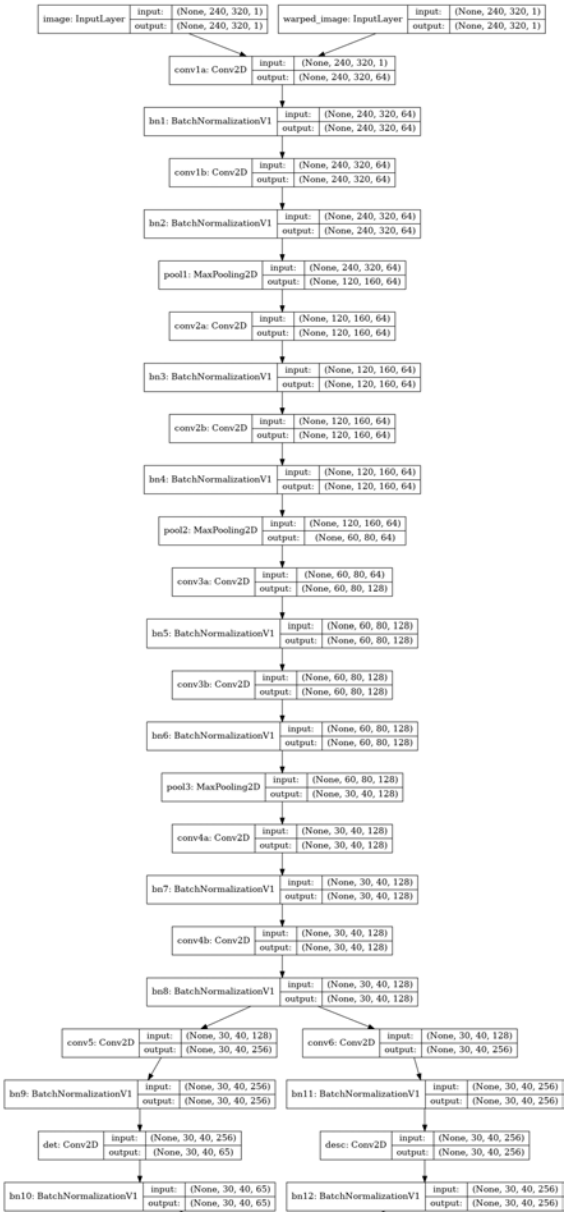

Figure 12: CNN Model Architecture: The model consists of a detector and descriptor head with specific weights, but a shared encoder base that allows for shared computation and representation across two tasks.

the detector head will therefore be a tensor sized $[H_c, W_c, 65]$ where $H_c = H/8$ and $W_c = W/8$. Each 65 length vector along the channel dimension corresponds to 8×8 non-overlapping regions of pixels plus an extra unit that corresponds to no interest point being detected in the 8×8 region. When extracting feature points

for evaluation, a softmax function is applied along the channel dimension to obtain probabilities of "pointness" for each pixel, and then the 65th dimension is removed. The resulting tensor of size $[H_c, W_c, 64]$ is then reshaped back into $[H, W]$ and pixels above a probability of $1/65$ are considered feature points. Additionally, non-maximum suppression is applied to the points to avoid clustering of features.

The output of the descriptor head subsequently outputs a tensor of shape [H_c, W_c, 256] which corresponds to a sparse descriptor tensor of the input image, i.e. a descriptor for each 8×8 non-overlapping patch. When exporting descriptors for evaluation, bi-cubic interpolation is performed to obtain a dense descriptor tensor of shape $[H, W, 256]$ and then each descriptor is L2-normalized. Additionally, the sparse descriptors are L2-normalized in the descriptor loss function described in Section 3.2.4.1.

3.2.4.1 Loss Function

The total loss is the weighted sum of intermediate losses, including the interest point detector loss, L_p and the descriptor loss, L_d, which allows both decoder heads to synergize during the training process. During training, the CNN uses pairs of synthetically warped images which have pseudo-ground truth interest point locations and the ground truth correspondence from a randomly generated homography, $\mathcal{H}$, relating the two images. such that if $\mathbf{x}$ is a set of homogeneous interest points in the base image, then

$$\mathbf{x}' = \mathcal{H}\mathbf{x} \tag{7}$$

are the same set of points in the warped image. This allows for the optimization of the two losses simultaneously, given pairs of images [11].

Given the outputs from the detector and descriptor heads are $\mathcal{X} \in \mathbb{R}^{H_c \times W_c \times 65}$ and

$\mathcal{D} \in \mathbb{R}^{H_c \times W_c \times 256}$, respectively, the final loss is

$$L(\mathcal{X}, \mathcal{Y}, \mathcal{X}', \mathcal{Y}', \mathcal{D}, \mathcal{D}', \mathcal{S}) = L_p(\mathcal{X}, \mathcal{Y}) + L_p(\mathcal{X}', \mathcal{Y}') + \lambda L_d(\mathcal{D}, \mathcal{D}', \mathcal{S}) \tag{8}$$

where the second detector loss corresponds to the image with the homography $\mathcal{H}$ applied and λ is used to balance the descriptor loss, where $\lambda = 10,000$. $\mathcal{Y} \in \mathbb{R}^{H_c \times W_c}$ represents the ground truth interest point locations in a one-hot embedding format. That is, given a label of shape $[H, W]$ with a value of one representing a truth interest point and a value of zero elsewhere, $\mathcal{Y}$ is an array of indexes calculated by the channel-wise argmax of a label that has had 8×8 non-overlapping pixel regions spatially reassembled into shape $[H_c, W_c, 65]$. In the case of no ground truth points present within an 8×8 region, the index for that region would be 64. On the other hand, if there are multiple ground truth points within a region, then one point is randomly selected.

The detector loss, L_p, is a sparse softmax cross-entropy loss given by

$$L_p(\mathcal{X}, \mathcal{Y}) = \frac{1}{H_c W_c} \sum_{h=1,w=1}^{H_c, W_c} - \log \frac{e^{\mathbf{x}_{hw}(y_{hw})}}{\sum_{k=1}^{65} e^{\mathbf{x}_{hw}(k)}} \tag{9}$$

where $\mathbf{x}_{hw}$ and y_{hw} are the individual entries in $\mathcal{X}$ and $\mathcal{Y}$ that represent each 8×8 region.

The descriptor loss, L_d, is applied to all pairs of descriptor cells $\mathbf{d}_{hw} \in \mathcal{D}$ from the first image and $\mathbf{d}'_{hw} \in \mathcal{D}'$ from the warped image. The homography-induced correspondence between each (h, w) and (h', w') region is

$$s_{hw} = \begin{cases} 1, & \text{if } ||\mathcal{H}\mathbf{p}_{hw} - \mathbf{p}'_{hw}|| < 8 \\ 0, & \text{otherwise} \end{cases} \tag{10}$$

where $\mathbf{p}_{hw}$ and $\mathbf{p}'_{hw}$ denote the location of the center pixel in the (h, w) and (h', w') regions, respectively. The entire set of correspondences for a pair of images is therefore represented by $\mathcal{S}$. The descriptor loss is a hinge loss with positive margin $m_p = 1$ and negative margin $m_n = 0.2$, defined as

$$L_d(\mathcal{D}, \mathcal{D}', \mathcal{S}) = \frac{1}{(H_c W_c)^2} \sum_{h=1,w=1}^{H_c,W_c} \sum_{h'=1,w'=1}^{H_c,W_c} l_d(\mathbf{d}_{hw}, \mathbf{d}'_{hw}, s_{hw}) \tag{11}$$

where

$$l_d(\mathbf{d}, \mathbf{d}', s) = \lambda_d s \cdot max(0, m_p - \mathbf{d}^T \mathbf{d}') + (1 - s) \cdot max(0, \mathbf{d}^T \mathbf{d}' - m_n) \tag{12}$$

and λ_d is a weighting term to help balance the fact that there are more negative correspondences than positive ones, where $\lambda_d = 0.05$.

Due to all of the images being pre-processed with illumination and/or homography changes before being fed into the model, a mask was also applied in the loss functions that ignored pixels not present in the pre-modified image.

3.2.4.2 Metrics

Evaluating the repeatability and homography estimation ability of the model could not be done until after the model was trained, however several metrics were implemented to aide in the evaluation of the training process. These included precision, recall and a threshold version of precision and recall. Given labels, y_{true}, and predictions, y_{pred}, both of the shape $[H, W]$ where a one denotes an interest point and zero elsewhere, then precision is defined as

$$\mathrm{p} = \frac{\sum(\mathrm{y}_{true} \cdot \mathrm{y}_{pred})}{\sum \mathrm{y}_{pred}} \tag{13}$$

and recall is defined as

$$\mathrm{r} = \frac{\sum(\mathrm{y}_{true} \cdot \mathrm{y}_{pred})}{\sum \mathrm{y}_{true}} \tag{14}$$

Since Equation (13) and Equation (14) only count predictions correct if they are the exact pixel of the label, threshold versions of each were implemented to count predictions correct within a one pixel distance. Similar to the loss functions, a mask was applied to ignore pixels not present in the pre-modified image.

3.2.5 Training Process

As mentioned in Section 3.2.1, training the full CNN required multiple rounds of training. In between each round, new ground truth feature points had to be extracted using the newly trained model since the MS-COCO [69] dataset does not contain interest point labeled images, leading to a boot strapped training approach. Additionally, to improve the generalization ability of the model on real images, randomly sampled illumination and/or homography changes were applied to each image in the pre-processing step. The illumination changes consisted of an accumulation of small changes in noise, brightness, contrast, shading and motion blur. The homography changes consisted of simple, less expressive transformations of an initial root center crop of an image. These transformations include scale, translation, in-plane rotation and perspective distortion. The illumination and homography warps were not applied to the validation sets.

3.2.5.1 Synthetic Dataset Training

Since there exists no large dataset of interest point labeled imagery, the synthetic dataset described in Section 3.2.1 was developed to bootstrap the training pipeline. Ignoring the descriptor head of the full CNN, an interest point detector model (denoted *Synthetic Shapes*) was trained on the synthetic dataset for 50 epochs using a

batch size of 33. Each image was resized to 120×160 and the Adam [58] optimizer was used with default parameters, including a learning rate, $lr = 0.001$. A model was saved after each epoch and the model selected for the next step in the training pipeline was the one with the best combination of the lowest detector loss and highest precision and recall (on the validation set), with an emphasis on the loss. Generally, a lower detector loss coincided with a higher precision and recall.

3.2.5.2 Exporting Detections

To generate consistent and reliable ground truth interest points for both the MS-COCO and the event-frame supplemented MS-COCO datasets, a large number of random homographic warps were applied to each image and the average response from the model trained with the synthetic dataset was taken. The number of homographic warps, $\mathcal{N}$, was chosen to be 100 to generate more reliable ground truth points.

3.2.5.3 MS-COCO Dataset Training of Detector Network

Again, ignoring the descriptor head, an interest point detector model was trained on both versions of the MS-COCO dataset described in Section 3.2.1 (models denoted *Detector (Round 1)* and *DetectorEvents (Round 1)*) for 20 epochs using a batch size of 32. Each image was resized to 240×320 and the Adam optimizer was used with default parameters, including a learning rate, $lr = 0.001$. The epoch model with the best combination of the lowest detector loss and highest precision and recall was selected to export a new set of ground truth interest points as described in Section 3.2.5.2. With these newly labeled datasets, a second round of training was accomplished (models denoted *Detector (Round 2)* and *DetectorEvents (Round 2)*) using the same hyper-parameters as the first round, and then another new set of detections were exported for both datasets to be utilized for the final step in the training process. The multiple

rounds of training the interest point detector model was done in attempt to boost the repeatability capacity of the model.

3.2.5.4 MS-COCO Dataset Training of Full CNN

The full CNN, including the descriptor head, was finally trained on both versions of the MS-COCO dataset (models denoted *DetDesc* and *DetDescEvents*) utilizing the latest set of "ground truth" detections. This model was trained for 15 epochs using a batch size of 2 (due to memory constraints). Each image was resized to 240×320 and the Adam optimizer was used with default parameters, except the learning rate was reduced to $lr = 0.0001$ in order to improve convergence.

3.2.5.5 Additional Training Processes

Slightly different training processes were looked at in attempt to improve the evaluation results, whose method is described in Section 3.2.6. These processes differ in the overall steps mentioned in previous sections, but utilize the same hyper-parameters for training and exporting detections. One process is the same as described in Sections 3.2.5.1 to 3.2.5.4, however only one round of the detector network training was done using the MS-COCO datasets instead of two rounds. The resulting models are denoted *DetDescLite* and *DetDescEventsLite*, repsectively.

Another process first trained on the synthetic dataset and exported detections as described in Sections 3.2.5.1 and 3.2.5.2, but training of the interest point detector model on the MS-COCO datasets was skipped, and the full CNN was trained utilizing the first set of exported detections. These models are denoted *DetDesc (Round 1)* and *DetDescEvents (Round 1)*. A third and final additional process built off the previously mentioned method by first exporting ground truth feature points using the *DetDesc (Round 1)* and *DetDescEvents (Round 1)* models. Then, a second round

of the full model was trained utilizing those labels. These models are denoted *DetDesc (Round 2)* and *DetDescEvents (Round 2)*, respectively.

In addition to training models from scratch during each step of the training processes, models were also trained using the weights of a previous step's model as an initial starting point. For example, the pre-trained version of *DetDesc* utilized the weights of the *Detector (Round 2)* model. Figure 13 helps illustrate the different training processes examined in this research.

3.2.6 Evaluation

Evaluating the different CNNs on both HPatches [70] and the event-based aerial imagery, described in Section 3.2.3, was accomplished by assessing the models' ability to repeatably detect features across illumination and/or viewpoint changes as well as their ability to correctly estimate a homography change. The metrics from the models were then compared to classical detector and descriptor algorithms, as well

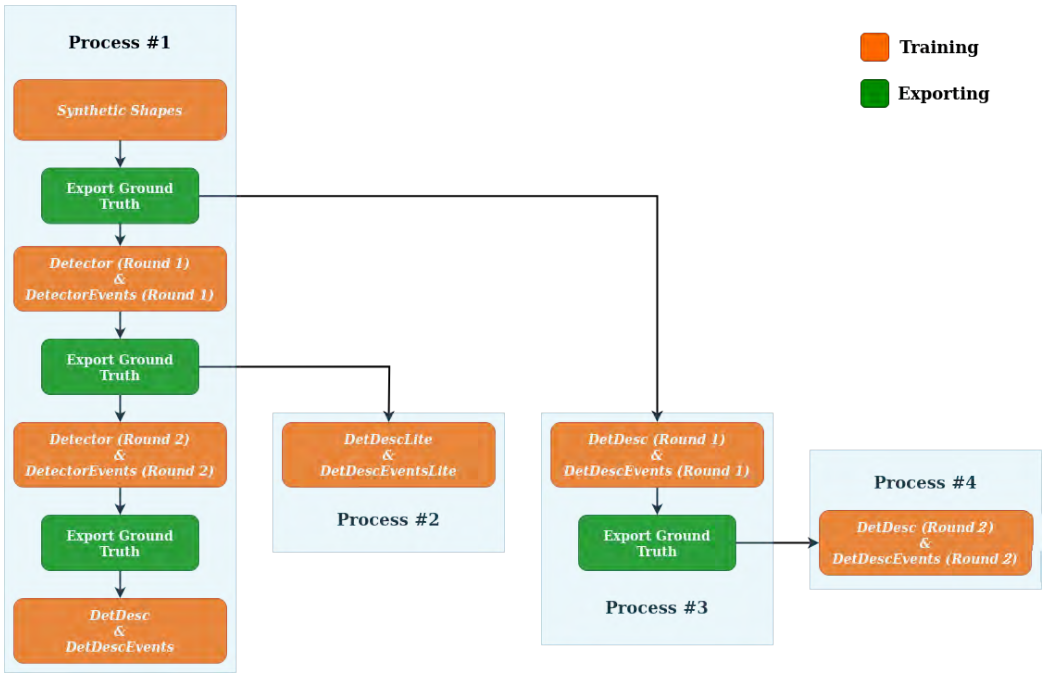

Figure 13: Training Processes: Illustration of the processes described in Section 3.2.5.

as the released SuperPoint model [11].

Repeatability was calculated by measuring the distance between extracted 2-D point centers on a pair of images related by a homography $\mathcal{H}$, with ϵ representing the correct pixel distance threshold between two points. Suppose N_1 points are are detected in the first image and N_2 points are detected in the second. Correctness for repeatability is then defined as

$$\text{Corr}(\mathbf{x}_i) = \min_{j \in (1,\ldots,N_2)} ||\hat{\mathbf{x}}_i - \mathbf{x}_j|| \leq \epsilon \tag{15}$$

where $\hat{\mathbf{x}}_i$ is an estimated point location in the second image and $\mathbf{x}_j$ is the ground truth point location in the second image calculated by applying the ground truth homography to the estimated detection in the first image. Repeatability is then the probability that a point is detected in the second image given by

$$\text{Rep} = \frac{\sum_i \text{Corr}(\mathbf{x}_i) + \sum_j \text{Corr}(\mathbf{x}_j)}{N_1 + N_2} \tag{16}$$

where in the case of $\text{Corr}(\mathbf{x}_j)$, $\hat{\mathbf{x}}_j$ is an estimated point location in the first image and $\mathbf{x}_i$ is the ground truth point location in the first image calculated by applying the inverse of the ground truth homography to the detection in the second image.

Homography estimation measures the ability of the model to estimate a homography relating a pair of images by comparing an estimated homography $\hat{\mathcal{H}}$ to the ground truth homography $\mathcal{H}$. Comparing the homographies is done by calculating how well $\hat{\mathcal{H}}$ transforms the four corners of the first image (denoted c_1, c_2, c_3, c_4) in relation to the transformed corners computed with $\mathcal{H}$, within a threshold ϵ. $\hat{c}'_1, \hat{c}'_2, \hat{c}'_3, \hat{c}'_4$ denotes the transformed corners computed with $\hat{\mathcal{H}}$ while c'_1, c'_2, c'_3, c'_4 denotes the transformed corners computed with $\mathcal{H}$. Homography correctness for all image pairs in the dataset

N is then calculated by

$$\text{Corr}_H = \frac{1}{N}\sum_{i=1}^{N}\left(\frac{1}{4}\sum_{j=1}^{4}||c'_{ij} - \hat{c}'_{ij}|| \leq \epsilon\right) \tag{17}$$

where scores range from 0 to 1. Estimated homographies $\hat{\mathcal{H}}$ are calculated using OpenCV [71] implementations to match interest points and descriptors between images and feeding these matched features to the findHomography() function with random sample consensus (RANSAC) enabled.

The repeatability evaluations resized images to 240×320 and utilized, at a maximum, the strongest 300 detections for the HPatches dataset and the strongest 100 detections for the event-based aerial imagery dataset. The best detections refers to the detections with the highest probability of "pointness" after the softmax function. The homography estimation, on the other hand, resized images to 480×640 for the HPatches dataset and 240×320 for the event-based aerial imagery dataset while both using, at a maximum, the strongest 1,000 detections. All evaluations used a non-maximum suppression value of 4, except for the homography estimation on HPatches which used a value of 8 due to the larger images.

3.3 Multi-State Constraint Kalman Filter

The MSCKF was introduced by Mourikis [18] in 2006, and the MSCKF implemented in this research is primarily inspired by that work. However, adjustments made by Sun et al. [19] were also implemented to improve results. The essence of a MSCKF is to update vehicle states from features tracked across multiple frames while propagating those states using IMU information.

3.3.1 Notation

To clarify the notation used throughout this methodology, several definitions are described below:

- $\mathbf{p}_j^A$ is the position of point j in the A frame of reference, while $\mathbf{p}_A^B$ is the position of the origin of frame A expressed in the B frame. $\mathbf{R}_A^B$ represents the direct cosine matrix (DCM) that rotates a point vector from the A frame to the B frame, $\mathbf{p}^B = \mathbf{R}_A^B \mathbf{p}^A$.

- Quaternions are in the Jet Propulsion Laboratory (JPL) [72] format,

$$\mathbf{q} = \begin{bmatrix} q_w & q_x & q_y & q_z \end{bmatrix}^T \equiv q_w + q_x\mathbf{i} + q_y\mathbf{j} + q_z\mathbf{k} \tag{18}$$

  where

$$\mathbf{i}^2 = \mathbf{j}^2 = \mathbf{k}^2 = \mathbf{ijk} = -1 \tag{19}$$

- $\mathbf{I}_N$ is an identity matrix of dimension N, and $\mathbf{0}_{M\times N}$ is a zero matrix with M rows and N columns.

- For a given vector $\mathbf{p} = \begin{bmatrix} x & y & z \end{bmatrix}^T$, the skew symmetric matrix is defined as

$$[\mathbf{p}\times] = \begin{bmatrix} 0 & -z & y \\ z & 0 & -x \\ -y & x & 0 \end{bmatrix} \tag{20}$$

3.3.2 Primary States

The primary propagated IMU states are defined by

$$\mathbf{X}_{IMU} = \begin{bmatrix} \mathbf{q}_G^I & \mathbf{b}_g & \mathbf{v}_I^G & \mathbf{b}_a & \mathbf{p}_I^G \end{bmatrix}^T \tag{21}$$

where the unit quaternion $\mathbf{q}_G^I$ describes the rotation from the inertial frame G to the IMU-affixed frame I; $\mathbf{p}_I^G$ is the position of the IMU, in meters, relative to the inertial frame; $\mathbf{v}_I^G$ is the velocity of the IMU, in meters per second, relative to the inertial frame; $\mathbf{b}_g$ and $\mathbf{b}_a$ are the 3×1 vectors describing the gyroscope biases and accelerometer biases, respectively. In this work, the initial IMU frame is set as the inertial frame.

Using the true IMU state would cause singularities in the resulting covariance matrices because of the additional unit constraint on the quaternions in the state vector [19]. Therefore, the error IMU states defined as

$$\tilde{\mathbf{X}}_{IMU} = \begin{bmatrix} \delta\theta_G^I & \tilde{\mathbf{b}}_g & \tilde{\mathbf{v}}_I^G & \tilde{\mathbf{b}}_a & \tilde{\mathbf{p}}_I^G \end{bmatrix}^T \tag{22}$$

are utilized, with standard additive error used for position, velocity and the biases. $\delta\theta_G^I$ is the attitude error (corresponding to 3 degrees-of-freedom (DOF)) associated with the error quaternion, $\delta\mathbf{q}$, which relates the estimated rotation, $\hat{\mathbf{q}}$, to the true rotation, $\mathbf{q}$, through

$$\mathbf{q} = \delta\mathbf{q} \otimes \hat{\mathbf{q}} \tag{23}$$

where $\otimes$ represents quaternion multiplication. For small values of $\delta\theta$, a decent approximation of $\delta\mathbf{q}$ can be estimated as

$$\delta\mathbf{q} \simeq \begin{bmatrix} \frac{1}{2}\delta\theta \\ 1 \end{bmatrix}. \tag{24}$$

Ultimately, N camera states are included in the state vector at any given time step k. The entire state vector is thus defined as

$$\mathbf{X}_k = \begin{bmatrix} \mathbf{X}_{IMU_k} & \mathbf{q}_G^{C_1} & \mathbf{p}_{C_1}^G & \cdots & \mathbf{q}_G^{C_N} & \mathbf{p}_{C_N}^G \end{bmatrix}^T \tag{25}$$

where $\mathbf{q}_G^{C_i}$ and $\mathbf{p}_{C_i}^G$ ($i = 1, ..., N$) are the estimates of the camera attitude and position, respectively. The error state is subsequently defined as

$$\tilde{\mathbf{X}}_k = \begin{bmatrix} \tilde{\mathbf{X}}_{IMU_k} & \delta\theta_G^{C_1} & \tilde{\mathbf{p}}_{C_1}^G & ... & \delta\theta_G^{C_N} & \tilde{\mathbf{p}}_{C_N}^G \end{bmatrix}^T. \tag{26}$$

3.3.3 State Augmentation

For each new image, the current camera pose (orientation $\mathbf{q}_G^{C_i}$ and position $\mathbf{p}_{C_i}^G$) is calculated using the IMU pose estimate (calculation method described in Section 3.3.4) by

$$\mathbf{q}_G^{C_i} = \mathbf{q}_I^C \otimes \mathbf{q}_G^{I_i} \tag{27}$$

and

$$\mathbf{p}_{C_i}^G = \mathbf{p}_{I_i}^G + \mathbf{R}_{I_i}^G(\mathbf{p}_C^I) \tag{28}$$

where $\mathbf{q}_I^C$ represents the rotation from the IMU frame to the camera frame, and $\mathbf{p}_C^I$ is the position of the camera in the IMU reference frame. $\mathbf{q}_G^{I_i}$ represents the rotation from the inertial frame to the IMU frame with a corresponding DCM of $\mathbf{R}_G^{I_i}$, where $\mathbf{R}_{I_i}^G = (\mathbf{R}_G^{I_i})^T$ represents the roatation from the IMU frame to the inertial frame. The camera pose estimates are appended to the state vector as described in Section 3.3.2.

The covariance matrix is additionally augmented through

$$\mathbf{P}_{k|k} = \begin{bmatrix} \mathbf{I}_{6N+15} \\ \mathbf{J} \end{bmatrix} \mathbf{P}_{k|k} \begin{bmatrix} \mathbf{I}_{6N+15} \\ \mathbf{J} \end{bmatrix}^T \tag{29}$$

where the Jacobian is derived from Equation (27) and Equation (28) as

$$\mathbf{J} = \begin{bmatrix} \mathbf{R}_I^C & \mathbf{0}_{3\times9} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times6N} \\ [\mathbf{R}_{I_i}^G \mathbf{p}_C^I \times] & \mathbf{0}_{3\times9} & \mathbf{I}_{3\times3} & \mathbf{0}_{3\times6N} \end{bmatrix}. \tag{30}$$

3.3.4 State Propagation

During propagation, the camera pose states and their associated error states and covariances remain unchanged. The filter propagation equations are derived by discretization of the estimated continuous-time IMU system dynamics, given by

$$\dot{\mathbf{q}}_G^I = \frac{1}{2}\mathbf{\Omega}(\omega)\mathbf{q}_G^I \ , \ \ \dot{\mathbf{b}}_g = \mathbf{0}_{3\times 1} \ , \ \ \dot{\mathbf{v}}_I^G = \mathbf{R}_I^G\mathbf{a} + \mathbf{g}^G \ , \ \ \dot{\mathbf{b}}_a = \mathbf{0}_{3\times 1} \ , \ \ \dot{\mathbf{p}}_I^G = \mathbf{v}_I^G \tag{31}$$

where ω and $\mathbf{a}$ are the IMU measurements for angular velocity and acceleration, respectively, with biases removed (i.e. $\omega = \omega_m - \mathbf{b}_g$, $\mathbf{a} = \mathbf{a}_m - \mathbf{b}_a$) and $\mathbf{g}^G$ is the gravitational acceleration. Furthermore,

$$\mathbf{\Omega}(\omega) = \begin{bmatrix} -[\omega\times] & \omega \\ -\omega^T & 0 \end{bmatrix} . \tag{32}$$

Based on Equation (31) the linearized continuous dynamics for the error IMU states follows

$$\dot{\tilde{\mathbf{X}}}_{IMU} = \mathbf{F}\tilde{\mathbf{X}}_{IMU} + \mathbf{G}\mathbf{n}_{IMU} \tag{33}$$

where $\mathbf{n}_{IMU} = [\mathbf{n}_g, \mathbf{n}_{wg}, \mathbf{n}_a, \mathbf{n}_{wa}]^T$ is the system noise. The vectors $\mathbf{n}_g$ and $\mathbf{n}_a$ represent the Gaussian noise of the gyroscope and accelerometer measurement, while $\mathbf{n}_{wg}$ and $\mathbf{n}_{wa}$ are the random walk rate of the gyroscope and accelerometer measurement biases. The covariance matrix of $\mathbf{n}_{IMU}$, $\mathbf{Q}_{IMU}$, depends on the IMU noise characteristics and

is computed off-line. Finally, $\mathbf{F}$ and $\mathbf{G}$ are given by

$$\mathbf{F} = \begin{bmatrix} -[\omega\times] & -\mathbf{I}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \\ -\mathbf{R}_I^G[\mathbf{a}\times] & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & -\mathbf{R}_I^G & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{I}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \end{bmatrix} \tag{34}$$

and

$$\mathbf{G} = \begin{bmatrix} -\mathbf{I}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{I}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & -\mathbf{R}_I^G & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{I}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \end{bmatrix}. \tag{35}$$

Every time a new IMU measurement is received, the IMU state estimate is propagated using 4th order Runge-Kutta numerical integration of Equation (31). To propagate the state covariance, the discrete time state transition matrix of Equation (33) and discrete time noise covariance matrix are computed as

$$\mathbf{\Phi}_k = \mathbf{\Phi}(t_{k+1}, t_k) = \exp(\int_{t_k}^{t_{k+1}} \mathbf{F}(\tau)\delta\tau) \tag{36}$$

and

$$\mathbf{Q}_k = \int_{t_k}^{t_{k+1}} \mathbf{\Phi}(t_{k+1}, \tau)\mathbf{G}\mathbf{Q}_{IMU}\mathbf{G}^T\mathbf{\Phi}(t_{k+1}, \tau)^T\delta\tau \tag{37}$$

respectively. Then, the propagated covariance of the IMU states is

$$\mathbf{P}_{II_{k+1|k}} = \mathbf{\Phi}_k\mathbf{P}_{II_{k|k}}\mathbf{\Phi}_k^T + \mathbf{Q}_k. \tag{38}$$

By partitioning the covariance of the entire states as

$$\mathbf{P}_{k|k} = \begin{bmatrix} \mathbf{P}_{II_{k|k}} & \mathbf{P}_{IC_{k|k}} \\ \mathbf{P}_{IC_{k|k}}^T & \mathbf{P}_{CC_{k|k}} \end{bmatrix} \tag{39}$$

where $\mathbf{P}_{II_{k|k}}$ is the 15×15 covariance matrix of the IMU states, $\mathbf{P}_{IC_{k|k}}$ is the correlation between the errors in the IMU states and the camera pose estimates and $\mathbf{P}_{CC_{k|k}}$ is the $6N \times 6N$ covariance matrix of the camera pose estimates, the full covariance propagation can be represented as

$$\mathbf{P}_{k+1|k} = \begin{bmatrix} \mathbf{P}_{II_{k+1|k}} & \mathbf{\Phi}_k\mathbf{P}_{IC_{k|k}} \\ \mathbf{P}_{IC_{k|k}}^T\mathbf{\Phi}_k^T & \mathbf{P}_{CC_{k|k}} \end{bmatrix}. \tag{40}$$

3.3.5 Image Processing Front-End

Features are detected in each image utilizing scale invariant feature transform (SIFT) [4] and the CNN model described in Section 3.2. Due to memory constraints and computational complexity, the MSCKF allows a maximum of N features to be detected in each image, distributed evenly throughout the image. Specifically, if an image is broken into a 4×5 grid of regions then, at a maximum, the strongest $N/20$ features in each of the 20 regions are utilized. This ensures that if weak features need to be discarded, then the remaining features are generally still spread out over the entire image, providing a wider field of information to use in motion estimation. Furthermore, a two-point RANSAC algorithm [73] is applied to remove outliers in temporal tracking.

3.3.6 Measurement Update

3.3.6.1 Update Mechanism

An update to the system error states, which aims to improve the estimation, occurs in one of two cases:

- When a feature that has been tracked in a number of images is no longer detected, then all of the observations of this feature are processed in the measurement update and subsequently discarded. Features that are no longer detected but have less than a certain number of observations (set as a tunable parameter) are also discarded, but not used in the measurement update.

- When the number of retained camera poses reaches some set maximum number, N_{max}. For this case, two camera states are selected and all feature observations obtained at those states are used for the measurement update. Selecting the two camera states is done through a keyframe selection strategy similar to the two-way marginalization method proposed in [74]. Based on the relative motion between the second latest camera state and the one previous to it, either the second latest or the oldest camera state is selected. This procedure is done twice to find the two camera states to remove. The latest camera state is always kept since it has the measurements for the newly detected features. This method followed the practice proposed in [19].

3.3.6.2 Measurement Model

Consider the case of a single feature, j, that has been observed from a set of camera poses ($C_i \rightarrow (\mathbf{q}_G^{C_i}, \mathbf{p}_{C_i}^G)$). Each of the observations of that feature is represented by

$$\hat{\mathbf{z}}_i^j = \frac{1}{\hat{Z}_j^{C_i}} \begin{bmatrix} \hat{X}_j^{C_i} \\ \hat{Y}_j^{C_i} \end{bmatrix} \tag{41}$$

and is related to the feature position relative to the camera frame, $\mathbf{p}_j^{C_i}$, given by

$$\hat{\mathbf{p}}_j^{C_i} = \begin{bmatrix} \hat{X}_j^{C_i} \\ \hat{Y}_j^{C_i} \\ \hat{Z}_j^{C_i} \end{bmatrix} = \hat{\mathbf{R}}_G^{C_i}(\hat{\mathbf{p}}_j^G - \hat{\mathbf{p}}_{C_i}^G) \tag{42}$$

where $\hat{\mathbf{p}}_j^G$ is the 3-D feature position in the inertial frame. This is calculated using the least squares minimization method described in Section 3.3.7.

The measurement residual is then computed as

$$\mathbf{r}_i^j = \mathbf{z}_i^j - \hat{\mathbf{z}}_i^j \tag{43}$$

where $\mathbf{z}$ and $\hat{\mathbf{z}}$ are the normalized coordinates of the feature in the given pose's reference frame and the least squares estimate (LSE), respectively. Linearizing the measurement model at the current estimate, Equation (43) can be approximated as

$$\mathbf{r}_i^j \simeq \mathbf{H}_{C_i}^j \tilde{\mathbf{X}}_{C_i} + \mathbf{H}_{f_i}^j \tilde{\mathbf{p}}_j^G + \mathbf{n}_i^j \tag{44}$$

where $\mathbf{H}_{C_i}^j$ and $\mathbf{H}_{f_i}^j$ are the Jacobians of the measurement $\mathbf{z}_i^j$ with respect to the state and the feature position, respectively, $\tilde{\mathbf{p}}_j^G$ is the error in the position estimate of j and

$\mathbf{n}_i^j$ is the zero-mean, white Guassian measurement noise. The Jacobians are given by

$$\mathbf{H}_{C_i}^j = \begin{bmatrix} \mathbf{0}_{2\times 15} & \mathbf{0}_{2\times 6} & \dots & \mathbf{J}_i^j \begin{bmatrix} [\hat{\mathbf{p}}_j^{C_i}\times] & -\mathbf{R}_G^{C_i} \end{bmatrix} & \dots & \mathbf{0}_{2\times 6} \end{bmatrix} \tag{45}$$

and

$$\mathbf{H}_{f_i}^j = \mathbf{J}_i^j \mathbf{R}_G^{C_i} \tag{46}$$

where

$$\mathbf{J}_i^j \begin{bmatrix} [\hat{\mathbf{p}}_j^{C_i}\times] & -\mathbf{R}_G^{C_i} \end{bmatrix} \tag{47}$$

is the Jacobian with respect to pose i and

$$\mathbf{J}_i^j = \frac{1}{\hat{Z}_j^{C_i}} \begin{bmatrix} 1 & 0 & -\frac{\hat{X}_j^{C_i}}{\hat{Z}_j^{C_i}} \\ 0 & 1 & -\frac{\hat{Y}_j^{C_i}}{\hat{Z}_j^{C_i}} \end{bmatrix}. \tag{48}$$

Vertically concatenating the residuals $\mathbf{r}_i^j$ of all M_j observations of this feature into $\mathbf{r}^j$, and the Jacobians $\mathbf{H}_{C_i}^j$ and $\mathbf{H}_{f_i}^j$ into $\mathbf{H}_{\mathbf{X}}^j$ and $\mathbf{H}_f^j$, respectively, results in

$$\mathbf{r}^j \simeq \mathbf{H}_{\mathbf{X}}^j \tilde{\mathbf{X}}^j + \mathbf{H}_f^j \tilde{\mathbf{p}}_j^G + \mathbf{n}^j \tag{49}$$

where $\mathbf{n}^j$ has a covariance matrix of $\mathbf{R}^j = \sigma_{im}\mathbf{I}_{2M_j}$.

As pointed out in [18], since $\mathbf{p}_j^G$ is computed using the camera poses, the error $\tilde{\mathbf{p}}_j^G$ is correlated with the camera poses in the error states, and therefore the residual $\mathbf{r}^j$ cannot be directly used in the measurement update. To overcome this, a new residual $\mathbf{r}_o^j$ is defined by projecting $\mathbf{r}^j$ on the left nullspace of $\mathbf{H}_f^j$. If $\mathbf{A}$ is considered the unitary matrix portion of the singular value decomposition (SVD) of $\mathbf{H}_f^j$, then the new residual is obtained through

$$\mathbf{r}_o^j = \mathbf{H}_o^j \tilde{\mathbf{X}}^j + \mathbf{n}_o^j = \mathbf{A}^T(\mathbf{H}_{\mathbf{X}}^j \tilde{\mathbf{X}}^j + \mathbf{n}^j). \tag{50}$$

3.3.6.3 Update Equations

At a given time step, there are L features selected by the criteria in Section 3.3.6.1 that must be processed, so each feature's residual $\mathbf{r}_o^j$ and measurement matrix $\mathbf{H}_o^j$ $(j = 1, ..., L)$ are concatenated to form

$$\mathbf{r}_o = \mathbf{H}_o\tilde{\mathbf{X}} + \mathbf{n}_o. \tag{51}$$

Since $\mathbf{H}_o$ can be exceptionally large, computational complexity is reduced by taking the QR decomposition of $\mathbf{H}_o$, denoted as

$$\mathbf{H}_o = \begin{bmatrix} \mathbf{Q}_1 & \mathbf{Q}_2 \end{bmatrix} \begin{bmatrix} \mathbf{T}_H \\ 0 \end{bmatrix} \tag{52}$$

where $\mathbf{Q}_1$ and $\mathbf{Q}_2$ are unitary matrices whose columns form bases for the range and nullspace of $\mathbf{H}_o$, respectively, and $\mathbf{T}_H$ is an upper triangular matrix. This allows Equation (51) to be rewritten as

$$\mathbf{r}_o = \begin{bmatrix} \mathbf{Q}_1 & \mathbf{Q}_2 \end{bmatrix} \begin{bmatrix} \mathbf{T}_H \\ 0 \end{bmatrix} \tilde{\mathbf{X}} + \mathbf{n}_o \tag{53}$$

which leads to

$$\begin{bmatrix} \mathbf{Q}_1^T\mathbf{r}_o \\ \mathbf{Q}_2^T\mathbf{r}_o \end{bmatrix} = \begin{bmatrix} \mathbf{T}_H \\ 0 \end{bmatrix} \tilde{\mathbf{X}} + \begin{bmatrix} \mathbf{Q}_1^T\mathbf{n}_o \\ \mathbf{Q}_2^T\mathbf{n}_o \end{bmatrix}. \tag{54}$$

The residual $\mathbf{Q}_2^T\mathbf{r}_o$ is only noise so can be discarded, resulting in the following residual for the MSCKF update:

$$\mathbf{r}_n = \mathbf{Q}_1^T\mathbf{r}_o = \mathbf{T}_H\tilde{\mathbf{X}} + \mathbf{Q}_1^T\mathbf{n}_o \tag{55}$$

where $\mathbf{Q}_1^T\mathbf{n}_o$ is a noise vector with covariance $\mathbf{R}_n = \sigma_{im}\mathbf{I}_r$, with r being the number of columns in $\mathbf{Q}_1$. The Kalman gain can then be computed as

$$\mathbf{K} = \mathbf{P}\mathbf{T}_H^T(\mathbf{T}_H\mathbf{P}\mathbf{T}_H^T + \mathbf{R}_n)^{-1} \tag{56}$$

and a state correction and an updated state covariance matrix can be computed, respectively, as

$$\Delta\mathbf{X} = \mathbf{K}\mathbf{r}_n \tag{57}$$

and

$$\mathbf{P}_{k+1|k+1} = (\mathbf{I}_{6N+15} - \mathbf{K}\mathbf{T}_H)\mathbf{P}_{k+1|k}(\mathbf{I}_{6N+15} - \mathbf{K}\mathbf{T}_H)^T + \mathbf{K}\mathbf{R}_n\mathbf{K}^T. \tag{58}$$

To help maintain symmetry in the covariance matrix, $\mathbf{P} = (\mathbf{P} + \mathbf{P}^T)/2$ was implemented after state propagation, state augmentation and measurement updates.

3.3.7 Least Squares Estimate

The estimate of a 3-D position of a tracked feature must be known before that feature can be used in the measurement update process in Section 3.3.6. The estimate of the 3-D position is calculated using a LSE approach, which utilizes Montiel's intersection approach [75] with an inverse-depth parametrization [76].

The 3-D position of the jth feature, expressed in the C_i camera frame where it is observed, is

$$\mathbf{p}_j^{C_i} = \mathbf{R}_{C_n}^{C_i}\mathbf{p}_j^{C_n} + \mathbf{p}_{C_n}^{C_i} \tag{59}$$

where C_n is the camera frame in which the feature was first observed, and $\mathbf{R}_{C_n}^{C_i}$ and $\mathbf{p}_{C_n}^{C_i}$ are the rotation and translation from the C_n frame to the C_i frame. Equation (59)

can be rewritten as

$$\mathbf{p}_j^{C_i} = Z_j^{C_n} \left(\mathbf{R}_{C_n}^{C_i} \begin{bmatrix} \frac{X_j^{C_n}}{Z_j^{C_n}} \\ \frac{Y_j^{C_n}}{Z_j^{C_n}} \\ 1 \end{bmatrix} + \frac{1}{Z_j^{C_n}} \mathbf{p}_{C_n}^{C_i} \right) = Z_j^{C_n} \left(\mathbf{R}_{C_n}^{C_i} \begin{bmatrix} \alpha_j \\ \beta_j \\ 1 \end{bmatrix} + \rho_j \mathbf{p}_{C_n}^{C_i} \right) \tag{60}$$

$$= Z_j^{C_n} \begin{bmatrix} h_{i1}(\alpha_j, \beta_j, \rho_j) \\ h_{i2}(\alpha_j, \beta_j, \rho_j) \\ h_{i3}(\alpha_j, \beta_j, \rho_j) \end{bmatrix} \tag{61}$$

where

$$h_{i1}(\alpha_j, \beta_j, \rho_j) = r_{i,11}\alpha_j + r_{i,12}\beta_j + r_{i,13} + \rho_j X_{C_n}^{C_i} \tag{62}$$

$$h_{i2}(\alpha_j, \beta_j, \rho_j) = r_{i,21}\alpha_j + r_{i,22}\beta_j + r_{i,23} + \rho_j Y_{C_n}^{C_i} \tag{63}$$

$$h_{i3}(\alpha_j, \beta_j, \rho_j) = r_{i,31}\alpha_j + r_{i,32}\beta_j + r_{i,33} + \rho_j Z_{C_n}^{C_i} \tag{64}$$

using

$$\mathbf{R}_{C_n}^{C_i} = \begin{bmatrix} r_{i,11} & r_{i,12} & r_{i,13} \\ r_{i,21} & r_{i,22} & r_{i,23} \\ r_{i,31} & r_{i,32} & r_{i,33} \end{bmatrix} \tag{65}$$

and

$$\mathbf{p}_{C_n}^{C_i} = \begin{bmatrix} X_{C_n}^{C_i} & Y_{C_n}^{C_i} & Z_{C_n}^{C_i} \end{bmatrix}^T. \tag{66}$$

Substituting Equation (61) into Equation (41) then gives

$$\hat{\mathbf{z}}_i^j = \frac{1}{h_{i3}(\alpha_j, \beta_j, \rho_j)} \begin{bmatrix} h_{i1}(\alpha_j, \beta_j, \rho_j) \\ h_{i2}(\alpha_j, \beta_j, \rho_j) \end{bmatrix} = \begin{bmatrix} g_{i1}(\alpha_j, \beta_j, \rho_j) \\ g_{i2}(\alpha_j, \beta_j, \rho_j) \end{bmatrix}. \tag{67}$$

These descriptions are then used in a least-squares minimization, with the error as $e_i = \hat{\mathbf{z}}_i^j - \mathbf{z}_i^j$, where $\mathbf{z}_i^j$ is the feature's measured normalized location and $\hat{\mathbf{z}}_i^j$ is the result

from Equation (67) after iteratively optimizing the estimated quantities $(\alpha_j, \beta_j, \rho_j)$ using

$$\begin{bmatrix} \Delta\alpha_j \\ \Delta\beta_j \\ \Delta\rho_j \end{bmatrix} = H^{-1}b \tag{68}$$

where

$$H = \sum_{i=n}^{M_j} (J_i(\alpha_j, \beta_j, \rho_j))^{-1} \Omega J_i(\alpha_j, \beta_j, \rho_j) \ , \tag{69}$$

$$b^T = \sum_{i=n}^{M_j} e_i \Omega J_i(\alpha_j, \beta_j, \rho_j). \tag{70}$$

The information matrix Ω is defined by

$$\Omega = \begin{bmatrix} \sigma_{im}^2 & 0 \\ 0 & \sigma_{im}^2 \end{bmatrix}^{-1} \tag{71}$$

and the Jacobian is

$$J_i(\alpha_j, \beta_j, \rho_j) = \begin{bmatrix} \frac{\delta g_{i1}}{\delta \alpha_j} & \frac{\delta g_{i1}}{\delta \beta_j} & \frac{\delta g_{i1}}{\delta \rho_j} \\ \frac{\delta g_{i2}}{\delta \alpha_j} & \frac{\delta g_{i2}}{\delta \beta_j} & \frac{\delta g_{i2}}{\delta \rho_j} \end{bmatrix} \tag{72}$$

where

$$\frac{\delta g_{i1}}{\delta \alpha_j} = \frac{r_{i,11}}{r_{i,31}\alpha_j + r_{i,32}\beta_j + r_{i,33} + \rho_j Z_{C_n}^{C_i}} - \frac{r_{i,31}(r_{i,11}\alpha_j + r_{i,12}\beta_j + r_{i,13} + \rho_j X_{C_n}^{C_i})}{(r_{i,31}\alpha_j + r_{i,32}\beta_j + r_{i,33} + \rho_j Z_{C_n}^{C_i})^2} \tag{73}$$

$$\frac{\delta g_{i1}}{\delta \beta_j} = \frac{r_{i,12}}{r_{i,31}\alpha_j + r_{i,32}\beta_j + r_{i,33} + \rho_j Z_{C_n}^{C_i}} - \frac{r_{i,32}(r_{i,11}\alpha_j + r_{i,12}\beta_j + r_{i,13} + \rho_j X_{C_n}^{C_i})}{(r_{i,31}\alpha_j + r_{i,32}\beta_j + r_{i,33} + \rho_j Z_{C_n}^{C_i})^2} \tag{74}$$

$$\frac{\delta g_{i1}}{\delta \rho_j} = \frac{X_{C_n}^{C_i}}{r_{i,31}\alpha_j + r_{i,32}\beta_j + r_{i,33} + \rho_j Z_{C_n}^{C_i}} - \frac{Z_{C_n}^{C_i}(r_{i,11}\alpha_j + r_{i,12}\beta_j + r_{i,13} + \rho_j X_{C_n}^{C_i})}{(r_{i,31}\alpha_j + r_{i,32}\beta_j + r_{i,33} + \rho_j Z_{C_n}^{C_i})^2} \tag{75}$$

$$\frac{\delta g_{i2}}{\delta \alpha_j} = \frac{r_{i,21}}{r_{i,31}\alpha_j + r_{i,32}\beta_j + r_{i,33} + \rho_j Z_{C_n}^{C_i}} - \frac{r_{i,31}(r_{i,21}\alpha_j + r_{i,22}\beta_j + r_{i,23} + \rho_j Y_{C_n}^{C_i})}{(r_{i,31}\alpha_j + r_{i,32}\beta_j + r_{i,33} + \rho_j Z_{C_n}^{C_i})^2} \tag{76}$$

$$\frac{\delta g_{i2}}{\delta \beta_j} = \frac{r_{i,22}}{r_{i,31}\alpha_j + r_{i,32}\beta_j + r_{i,33} + \rho_j Z_{C_n}^{C_i}} - \frac{r_{i,32}(r_{i,21}\alpha_j + r_{i,22}\beta_j + r_{i,23} + \rho_j Y_{C_n}^{C_i})}{(r_{i,31}\alpha_j + r_{i,32}\beta_j + r_{i,33} + \rho_j Z_{C_n}^{C_i})^2} \quad (77)$$

$$\frac{\delta g_{i2}}{\delta \rho_j} = \frac{Y_{C_n}^{C_i}}{r_{i,31}\alpha_j + r_{i,32}\beta_j + r_{i,33} + \rho_j Z_{C_n}^{C_i}} - \frac{Z_{C_n}^{C_i}(r_{i,21}\alpha_j + r_{i,22}\beta_j + r_{i,23} + \rho_j Y_{C_n}^{C_i})}{(r_{i,31}\alpha_j + r_{i,32}\beta_j + r_{i,33} + \rho_j Z_{C_n}^{C_i})^2} \quad (78)$$

Each iteration generates new values for $\Delta\alpha_j$, $\Delta\beta_j$ and $\Delta\rho_j$ to be applied to $\hat{\alpha}_j$, $\hat{\beta}_j$ and $\hat{\rho}_j$ on the next iteration. Once $\Delta\alpha_j$, $\Delta\beta_j$ and $\Delta\rho_j$ fall below some defined threshold, $\hat{\alpha}_j$, $\hat{\beta}_j$ and $\hat{\rho}_j$ can be used to generate a LSE of the the position of the feature in the inertial frame as

$$\hat{\mathbf{p}}_j^G = \frac{1}{\hat{\rho}_j}\mathbf{R}_{C_n}^G \begin{bmatrix} \hat{\alpha}_j \\ \hat{\beta}_j \\ 1 \end{bmatrix} + \hat{\mathbf{p}}_{C_n}^G. \quad (79)$$

3.3.8 Observability Constraint

The extended Kalman filter (EKF)-based visual-inertial odometry (VIO) for 6-DOF motion estimation has four unobservable directions corresponding to position and rotation along the gravity axis, or yaw angle. A simple implementation of MSCKF will gain spurious information on the yaw due to the fact that the linearizing point of the process and measurement step are different at the same time step [19]. As is done in [19], the consistency of the filter is maintained by applying the observability constrained EKF (OC-EKF) method [77]. This allows the filter to not heavily depend on accurate initial estimation, and camera poses in the state vector can be represented with respect to the inertial frame instead of the latest IMU frame so that the uncertainty of the existing camera states is not affected by the uncertainty of the latest IMU state during state propagation [19]. The OC-EKF method works by maintaining the nullspace, $\mathbf{N}_k$, at each time step and using it to enforce the unobservable directions. The initial nullspace as well as the nullspace at all subsequent times are

defined by

$$\mathbf{N}_0 = \begin{bmatrix} \mathbf{0}_{3\times3} & \hat{\mathbf{R}}^I_{G,0|0}\mathbf{g}^G \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & -[\hat{\mathbf{v}}^G_{I,0|0}\times]\mathbf{g}^G \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \\ \mathbf{I}_3 & -[\hat{\mathbf{p}}^G_{I,0|0}\times]\mathbf{g}^G \end{bmatrix}, \quad \mathbf{N}_k = \begin{bmatrix} \mathbf{0}_{3\times3} & \hat{\mathbf{R}}^I_{G,k|k-1}\mathbf{g}^G \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & -[\hat{\mathbf{v}}^G_{I,k|k-1}\times]\mathbf{g}^G \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \\ \mathbf{I}_3 & -[\hat{\mathbf{p}}^G_{I,k|k-1}\times]\mathbf{g}^G \end{bmatrix} \tag{80}$$

First, prior to propagating the covariance as described in Section 3.3.4, $\mathbf{\Phi}_k$ is modified according to Equation (81) - Equation (83), where $\mathbf{\Phi}_{11}$, $\mathbf{\Phi}_{31}$ and $\mathbf{\Phi}_{51}$ correspond to the first, third and fifth 3×3 blocks in the first three columns of $\mathbf{\Phi}_k$.

$$\mathbf{\Phi}_{11} = \mathbf{R}^{I,k+1|k}_{I,k|k-1} = \mathbf{R}^{I,k+1|k}_{G}\mathbf{R}^{G}_{I,k|k-1} \tag{81}$$

$$\mathbf{\Phi}_{31}\mathbf{R}^I_{G,k|k-1}\mathbf{g}^G = ([\hat{\mathbf{v}}^G_{I,k|k-1}\times] - [\hat{\mathbf{v}}^G_{I,k+1|k}\times])\mathbf{g}^G \tag{82}$$

$$\mathbf{\Phi}_{51}\mathbf{R}^I_{G,k|k-1}\mathbf{g}^G = (\delta t[\hat{\mathbf{v}}^G_{I,k|k-1}\times] - [\hat{\mathbf{p}}^G_{I,k+1|k}\times])\mathbf{g}^G \tag{83}$$

Equation (82) and Equation (83) are of the form $\mathbf{Au} = \mathbf{w}$, where $\mathbf{u}$ and $\mathbf{w}$ are nullspace elements that are fixed and $\mathbf{A} = \mathbf{\Phi}_{31}, \mathbf{\Phi}_{51}$. We seek to find the minimum perturbation of $\mathbf{A}$, $\mathbf{A}^*$, which is given by

$$\mathbf{A}^* = \mathbf{A} - (\mathbf{Au} - \mathbf{w})(\mathbf{u}^T\mathbf{u})^{-1}\mathbf{u}^T \tag{84}$$

Secondly, each feature observation Jacobian, $\mathbf{H}^j_{C_i}$, is modified such that

$$\mathbf{H}^j_{C_i}\begin{bmatrix} \hat{\mathbf{R}}^I_{G,k|k-1}\mathbf{g}^G \\ ([\hat{\mathbf{p}}^G_j\times] - [\hat{\mathbf{p}}^G_{I,k|k-1}\times])\mathbf{g}^G \end{bmatrix} = 0 \tag{85}$$

which is in the form $\mathbf{Au} = 0$. The optimal $\mathbf{A}^*$ is then computed as

$$\mathbf{A}^* = \mathbf{H}_{C_i}^j = \mathbf{A} - (\mathbf{Au})(\mathbf{u}^T\mathbf{u})^{-1}\mathbf{u}^T \tag{86}$$

and $\mathbf{H}_{f_i}^j$ is recomputed as $\mathbf{H}_{f_i}^j = -\mathbf{A}_{1:2,4:6}^*$.

3.4 Visual-Inertial Odometry Datasets

3.4.1 RPGs Boxes 6-DOF Dataset

The Robotics and Perception Group (RPG) has been one of the main contributors to event-based research, and have released the Event-Camera Dataset [1] which contains multiple DAVIS240C datasets of differing scenarios. The dataset used in this research (Boxes 6-DOF) contains the event data output, 180×240 grayscale ".png" images with the corresponding timestamps, camera calibration parameters, IMU measurements and the ground truth position and orientation information. No ground truth velocity information was provided so it was calculated as the gradient of the position information.

3.4.2 Camp Atterbury Flight Test

An event-based camera dataset was captured in October 2018 at Camp Atterbury, Indiana with a DAVIS240C attached to the belly of a 14-foot fixed wing Aeroworks 100CC Carbon CUB UAV. Other sensors also recording on the same flights were a 1280×960 color camera, a Piksi multi-GNSS module, a magnetometer and a Pixhawk autopilot flight controller. These sensors enabled high-fidelity ground truth data for the flight as well as generating multiple datasets to support future research efforts at the Autonomy and Navigation Technology (ANT) Center.

The DAVIS camera provided the 180×240 grayscale imagery, event data output

and IMU measurements, all with corresponding time stamps. The ground truth position information, in the North-East-Down (NED) frame, was extracted from the Piksi module while the orientation information was extracted from the Pixhawk output. The yaw measurements from the Pixhawk were unreliable, however, so the heading was calculated from the position information and used instead. The ground truth velocity for this dataset was also calculated as the gradient of the position information. A top-down view of this dataset can be seen in Figure 14.

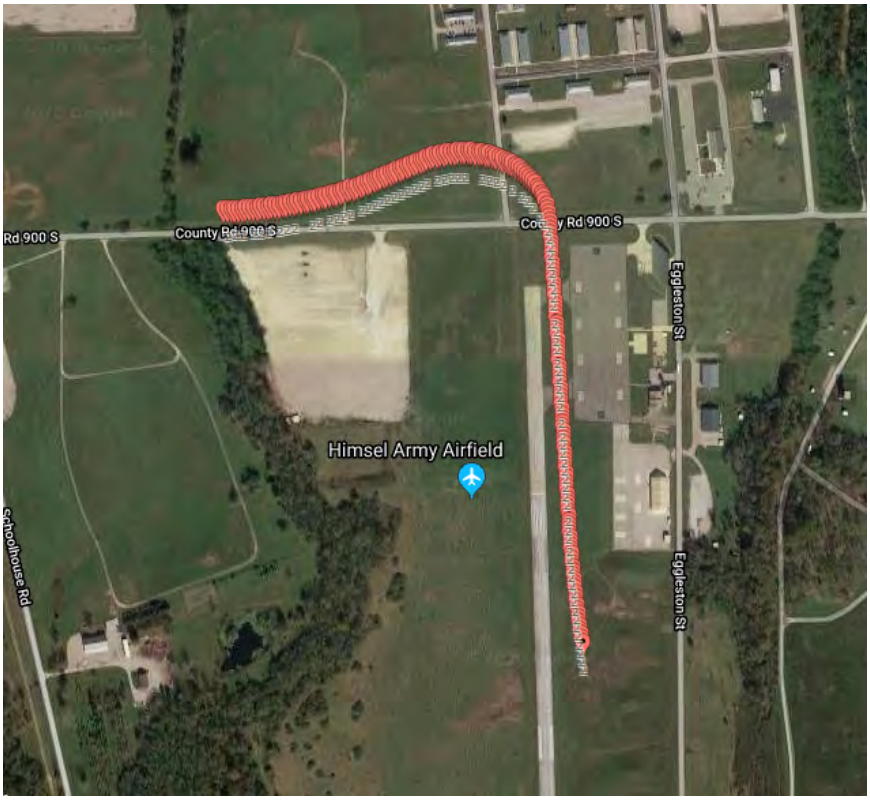

Figure 14: Camp Atterbury Flight: Top-down view of the Camp Atterbury flight test [78].

IV. Results and Analysis

Preamble

Using the methodology described in Chapter III, the convolutional neural network (CNN) that detects and describes feature points was developed and evaluated separately prior to being integrated into the full event-based visual-inertial odometry (EVIO) pipeline, which includes the multi-state constraint Kalman filter (MSCKF) back-end and the recurrent-convolutional neural network (RCNN) front-end [16, 17] to reconstruct event data into image frames. The event-based frames from both datasets described in Section 3.4 are shown in Section 4.1 in comparison to the grayscale image frames at similar instances. Section 4.2 describes the results of the CNN training, while Section 4.3 shows the repeatability and homography estimation performances of each model on HPatches [70] and the event-based aerial imagery dataset. The gyroscope and accelerometer biases had to be estimated prior to evaluating the MSCKF, and the results of this are shown in Section 4.4. Finally, the full EVIO pipeline, including grayscale and event frames, is evaluated in Section 4.5.

4.1 Event-Based Image Frames

The neural network developed in [16, 17] takes as input data from an event camera, which includes the event timestamps, pixel location and polarity of the events. It then uses a defined parameter, the fraction of events n, to calculate the number of events to use for each psuedo-frame N_e, such that

$$N_e = n \cdot H \cdot W \tag{87}$$

where H and W are the pixel height and width of the camera. Scenes with more sparse activity should use a lower n while scenes with dense activity or that contain much more varied texture should use a higher n, where n ranges from 0 to 1.

The Robotics and Perception Groups (RPGs) Boxes dataset [1] contains many highly textured images. These images cause a high rate of event output, especially in the latter half of the dataset where the camera is being moved at a rapid rate. A sample grayscale image from this dataset at a point where the camera is being moved rapidly is shown in Figure 15. Figure 16 shows the results of the RCNN created event frames at different values of n. At $n = 0.1$, the image contains more blurring when compared to $n = 0.3$ and $n = 0.7$, and while the latter two values produce similar results, $n = 0.7$ produces slightly more textured images. Each example psuedo-frame (with the exception of the right side of the first pseudo-frame) is able to produce higher contrasting images than the grayscale camera frame. The MSCKF solutions in Section 4.5 utilize event frames created with $n = 0.7$.

The Camp Atterbury dataset is from an unmanned aerial vehicle (UAV) flying at around 20 m/s at an altitude of 250 m over an airfield surrounded by a hand full of buildings and fields with a few roads. The sparseness of the scene resulted in a

Figure 15: Example Grayscale Image from Boxes Dataset: Highly textured boxes of various sizes.

Figure 16: Boxes Dataset - Event Psuedo-Frames: Event-based frames from a similar time instance as that in Figure 15. From left to right, the values of n utilized are 0.1, 0.3 and 0.7.

much lower rate of events compared to that of the Boxes dataset. A sample grayscale image from this dataset is shown in Figure 17. Figure 18 shows the results of the psuedo-frames at different values of n. At $n = 0.05$ and $n = 0.6$ the images are noisy compared to the image at $n = 0.2$ due to there being not enough events per image and too many events per image, respectively. Additionally, while each event frame better highlights the darker areas of the grayscale frame, they are all still noisy due to the sparseness of the scene. The MSCKF solutions in Section 4.5 utilize event frames created with $n = 0.2$ for this dataset.

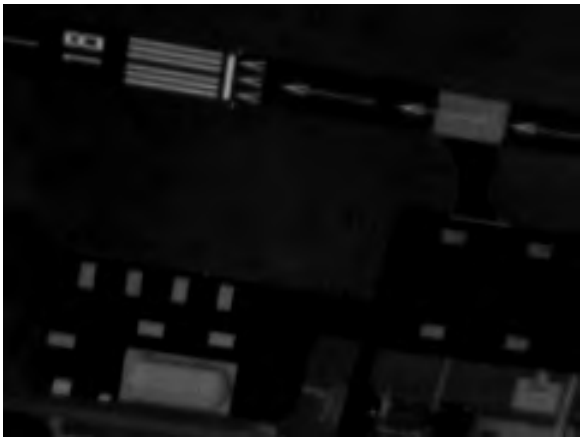

Figure 17: Example Grayscale Image from Camp Atterbury Dataset: Downward facing camera aboard an UAV flying at an altitude of 250 m.

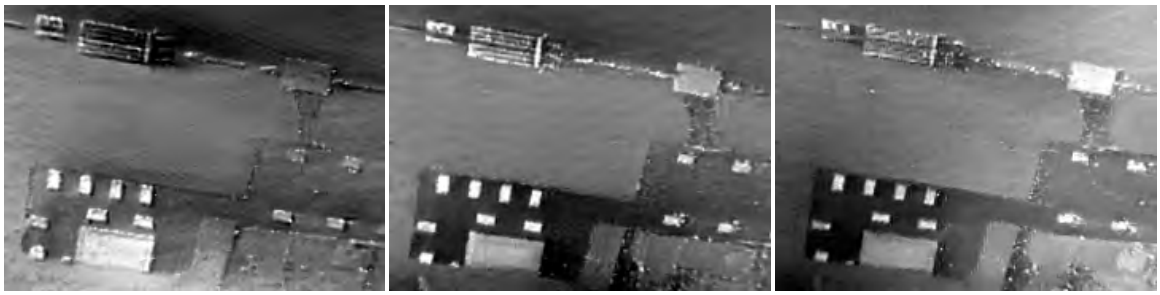

Figure 18: Camp Atterbury Dataset - Event Psuedo-Frames: Event-based frames from a similar time instance as that in Figure 17. From left to right, the values of n utilized are 0.05, 0.2 and 0.6.

4.2 CNN Training

Appendix A contains the validation set results of each model trained according to the processes described in Section 3.2.5. Figure 33 shows the results of the *Synthetic Shapes* training, Figure 34 through Figure 39 show the results of each successive model trained with the original MS-COCO dataset, and Figure 40 through Figure 45 show the results of each successive model trained with the event frame supplemented MS-COCO dataset.

When looking at the models trained without using pre-trained weights, their loss and metric values all generally follow the same trend of leveling out after $40\% - 60\%$ of the training time. On the other hand, models that used pre-trained weights generally had their loss and metric values remain level throughout the entire training process, with a few exceptions where the loss was initially large. Additionally, utilizing pre-trained weights seemed to offer no clear benefits as the models trained with and without them converged to the same loss and metric values. In one case (*DetDesc (Round 1)*) the model trained without pre-trained weights even outperformed the model trained with them. One exception to this, however, was both *DetDesc (Round 2)* and *DetDescEvents (Round 2)* benefited from using pre-trained weights as their loss and metric results were better throughout the entire training process.

Furthermore, models trained utilizing more steps in their training process (i.e. *DetDesc*) generally had worse loss values but better metric values. For example, *Detector (Round 2)* and *DetectorEvents (Round 2)* had ∼30% higher loss values compared to the round 1 detectors, but each metric was slightly better. This trend also applied to the detector and descriptor models, where *DetDesc*/*DetDescEvents* had ∼30% and ∼20% higher loss values than *DetDesc (Round 1)*/*DetDescEvents (Round 1)* and *DetDescLite*/*DetDescEventsLite*, respectively, with each metric converging to a higher value further into the training pipeline. One exception to this was that *DetDesc (Round 2)*/*DetDescEvents (Round 2)* had the highest recall values compared to the other detector and descriptor models. Precision and recall are not comprehensive metrics for evaluating feature detectors, however, and should not be heavily relied on to judge the performance of the models while training. The metrics were therefore used as just a supplement to the loss values.

Finally, the trend of the validation loss getting worse as the training process got longer hints at over-training. This means that shorter processes (*DetDesc (Round 1)* and *DetDesc (Round 2)*/*DetDescLite*) would result in a better performing feature detector and descriptor, which is reinforced by the results in Section 4.3.

Through the rationale discussed in Section 3.2.5, the models selected for each step of the training pipeline are listed in Table 1.

Table 1: Selected Models

Model	**Epoch**	
	No Pre-trained Weights	**Pre-trained Weights**
Synthetic Shapes	47	N/A
Detector (Round 1)	12	16
DetectorEvents (Round 1)	15	17
Detector (Round 2)	11	17
DetectorEvents (Round 2)	16	11
DetDesc	15	14
DetDescEvents	15	13
DetDescLite	15	15
DetDescEventsLite	13	14
DetDesc (Round 1)	15	14
DetDescEvents (Round 1)	11	15
DetDesc (Round 2)	14	12
DetDescEvents (Round 2)	14	13

4.3 CNN Evaluation

Table 2 contains the repeatability results on the HPatches [70] dataset for each model trained, a few classical detectors and the SuperPoint [11] model. The Harris [8] corner detector outperforms every other model/algorithm on the viewpoint scenes for larger threshold values ($\epsilon = 3, 5$). The second best performer (*DetDesc (Round 1)* using these threshold values is not too much worse, however, at $\sim 6 - 7\%$ smaller repeatability. One of the other models trained as part of this research (Pre-trained *DetDesc (Round 2)*) even outperforms every other algorithm/model on the viewpoint scenes when $\epsilon = 1$. A main advantage of the trained models is their ability to repeatably detect features across illumination changes as most of the trained models outperform each classical detector/SuperPoint at each of the threshold values, especially at $\epsilon = 1, 3$. An interesting note is that, generally, the trained models perform

much better at a strict threshold value ($\epsilon = 1$) for both viewpoint and illumination changes, meaning they can more accurately detect the same feature across said changes.

Table 3 contains the same information as Table 2, however for the event-based aerial dataset. Once again the Harris detector outperforms every model/algorithm at $\epsilon = 3, 5$ with Pre-trained *DetDescEvents (Round 2)* performing the best at $\epsilon = 1$. The trained models are generally on par with the classical detectors for this dataset, though slightly outperforming them at $\epsilon = 3$ (excluding the Harris detector).

When comparing the trained models' repeatability performance to one another on both evaluation datasets (Tables 2 and 3) the models that utilized the most steps in their training process (*DetDesc*/*DetDescEvents*) performed the worst, reinforcing the idea that long processes seem to result in over-fitting. While the *DetDesc (Round 2)* and *DetDescLite* type models technically used the same amount of training/exporting steps, it seems that training multiple rounds of the full detector plus descriptor after *Synthetic Shapes* is more beneficial than first training another detector based on the results in Tables 2 and 3. Additionally, the models trained with the event frame supplemented MS-COCO dataset slightly under performed their counterparts trained with the original MS-COCO dataset [69], on both evaluation datasets. This was expected for the HPatches dataset but not the aerial event frames. So, utilizing the original MS-COCO dataset for training produces models that can still generalize well in comparison to models trained for a specific type of data.

The homography estimation results on the viewpoint and illumination scenes in the HPatches dataset can be seen in Table 4, while the homography estimation results on the aerial event frame dataset can be seen in Table 5. The same trends that were present in the repeatability results are also present in the homography estimation results for both evaluation datasets when comparing the trained models to one

Table 2: HPatches Repeatability Results

Model	**Repeatability**					
	Viewpoint			**Illumination**		
	$\epsilon = 1$	$\epsilon = 3$	$\epsilon = 5$	$\epsilon = 1$	$\epsilon = 3$	$\epsilon = 5$
DetDesc	0.362	0.635	0.742	0.435	0.638	0.734
Pre-trained DetDesc	0.355	0.631	0.741	0.433	0.637	0.732
DetDescEvents	0.350	0.623	0.736	0.435	0.637	0.734
Pre-trained DetDescEvents	0.352	0.626	0.737	0.433	0.634	0.730
DetDescLite	0.374	0.676	0.777	**0.441**	0.661	0.760
Pre-trained DetDescLite	0.367	0.666	0.770	0.438	0.655	0.751
DetDescEventsLite	0.369	0.682	0.784	0.436	0.663	0.762
Pre-trained DetDescEventsLite	0.364	0.672	0.777	0.435	0.660	0.758
DetDesc (Round 1)	0.364	**0.694**	**0.795**	0.429	**0.669**	**0.772**
Pre-trained DetDesc (Round 1)	0.340	0.685	0.792	0.416	0.666	0.770
DetDescEvents (Round 1)	0.349	0.686	0.792	0.425	0.667	0.770
Pre-trained DetDescEvents (Round 1)	0.331	0.682	0.792	0.414	0.665	0.770
DetDesc (Round 2)	0.375	0.685	0.787	0.438	0.663	0.762
Pre-trained DetDesc (Round 2)	**0.377**	0.685	0.786	0.438	0.665	0.764
DetDescEvents (Round 2)	0.369	0.678	0.782	0.439	0.661	0.759
Pre-trained DetDescEvents (Round 2)	0.371	0.680	0.780	0.437	0.661	0.758
SuperPoint	0.285	0.628	0.757	**0.375**	**0.639**	**0.762**
FAST	0.294	0.630	0.786	0.363	0.574	0.716
Harris	**0.312**	**0.758**	**0.868**	0.372	0.627	0.752
SIFT	0.260	0.486	0.685	0.278	0.434	0.595

another, further reinforcing the ideas of over-fitting and training with the original MS-COCO dataset. When comparing the trained models to the classical detectors and descriptors, pre-trained *DetDesc (Round 2)* performs competitively on HPatches viewpoint scenes across all threshold values with scale invariant feature transform (SIFT) performing the best overall. However, pre-trained *DetDesc (Round 2)*, and most of the other trained models, perform consistently better on the illumination change scenes than any of the comparison algorithms.

Table 3: Aerial Event-Frame Repeatability Results

Model	**Repeatability**		
	$\epsilon = 1$	$\epsilon = 3$	$\epsilon = 5$
DetDesc	0.673	0.811	0.840
Pre-trained DetDesc	0.671	0.816	0.845
DetDescEvents	0.660	0.813	0.844
Pre-trained DetDescEvents	0.662	0.814	0.845
DetDescLite	0.673	0.826	0.857
Pre-trained DetDescLite	0.659	0.816	0.848
DetDescEventsLite	0.645	0.820	0.859
Pre-trained DetDescEventsLite	0.643	0.814	0.849
DetDesc (Round 1)	0.635	0.830	0.871
Pre-trained DetDesc (Round 1)	0.614	0.825	0.870
DetDescEvents (Round 1)	0.627	0.820	0.861
Pre-trained DetDescEvents (Round 1)	0.612	0.824	0.868
DetDesc (Round 2)	0.674	0.833	0.869
Pre-trained DetDesc (Round 2)	0.674	**0.839**	**0.874**
DetDescEvents (Round 2)	0.667	0.829	0.862
Pre-trained DetDescEvents (Round 2)	**0.680**	0.834	0.868
SuperPoint	0.465	0.792	0.868
FAST	0.628	0.783	0.891
Harris	**0.662**	**0.885**	**0.951**
SIFT	0.643	0.724	0.826

SIFT is the best overall classical algorithm for the aerial event frame dataset, performing the best at $\epsilon = 1$. Pre-trained *DetDesc (Round 2)* is again the best performing trained model, producing results slightly better than SIFT at $\epsilon = 3, 5$.

In conclusion, the trained models perform competitively with the classical detectors in both repeatability and homography estimation when considering viewpoint changes. On the other hand, the models consistently perform better than any of the other comparison algorithms when considering illumination changes. Additionally, process four from Figure 13, as well as utilizing the original MS-COCO dataset,

produced the best results when comparing the trained models to each other.

Appendix B contains qualitative examples of the trained models versus the classical detector and descriptor algorithms used as comparisons.

Table 4: HPatches Homography Estimation Results

Model	**Homography Estimation**					
	Viewpoint			**Illumination**		
	$\epsilon = 1$	$\epsilon = 3$	$\epsilon = 5$	$\epsilon = 1$	$\epsilon = 3$	$\epsilon = 5$
DetDesc	0.325	0.702	0.800	0.589	0.919	0.982
Pre-trained DetDesc	0.342	0.658	0.803	0.579	0.937	0.975
DetDescEvents	0.298	0.685	0.800	0.572	0.919	0.961
Pre-trained DetDescEvents	0.285	0.651	0.780	0.568	0.926	0.968
DetDescLite	0.322	0.695	0.807	0.589	0.940	0.982
Pre-trained DetDescLite	0.298	0.692	0.803	0.579	0.940	0.982
DetDescEventsLite	0.298	0.685	0.814	0.579	0.937	0.982
Pre-trained DetDescEventsLite	0.288	0.685	0.814	0.568	0.940	0.979
DetDesc (Round 1)	0.322	0.702	0.817	0.579	0.930	0.979
Pre-trained DetDesc (Round 1)	0.288	0.675	0.807	0.547	0.930	0.982
DetDescEvents (Round 1)	0.302	0.698	0.807	0.568	0.933	0.982
Pre-trained DetDescEvents (Round 1)	0.275	0.671	0.817	0.488	0.933	0.982
DetDesc (Round 2)	0.336	0.685	**0.841**	0.589	0.930	**0.986**
Pre-trained DetDesc (Round 2)	**0.348**	**0.719**	0.824	**0.593**	**0.944**	0.972
DetDescEvents (Round 2)	0.308	0.702	0.817	0.565	**0.944**	0.982
Pre-trained DetDescEvents (Round 2)	0.315	0.698	0.807	0.572	0.933	0.972
SuperPoint	0.349	0.746	**0.851**	0.530	**0.919**	**0.969**
FAST-FREAK	0.288	0.590	0.712	0.551	0.758	0.789
ORB	0.061	0.386	0.536	0.232	0.530	0.593
SIFT	**0.464**	**0.769**	0.831	**0.540**	0.779	0.825

Table 5: Aerial Event-Frame Homography Estimation Results

Model	**Homography Estimation**		
	$\epsilon = 1$	$\epsilon = 3$	$\epsilon = 5$
DetDesc	0.606	0.919	0.957
Pre-trained DetDesc	0.588	0.903	0.944
DetDescEvents	0.598	0.896	0.938
Pre-trained DetDescEvents	0.565	0.893	0.944
DetDescLite	0.799	0.971	0.992
Pre-trained DetDescLite	0.753	0.972	0.987
DetDescEventsLite	0.639	0.926	0.958
Pre-trained DetDescEventsLite	0.670	0.929	0.961
DetDesc (Round 1)	0.738	0.953	0.977
Pre-trained DetDesc (Round 1)	0.713	0.935	0.967
DetDescEvents (Round 1)	0.705	0.943	0.970
Pre-trained DetDescEvents (Round 1)	0.700	0.935	0.968
DetDesc (Round 2)	0.783	0.985	0.995
Pre-trained DetDesc (Round 2)	**0.808**	**0.992**	**0.997**
DetDescEvents (Round 2)	0.718	0.990	0.995
Pre-trained DetDescEvents (Round 2)	0.769	0.989	**0.997**
SuperPoint	0.436	0.842	0.914
FAST-FREAK	0.727	0.917	0.947
ORB	0.196	0.649	0.788
SIFT	**0.870**	**0.983**	**0.990**

4.4 Initial Bias Estimates

Biases were manually estimated for each dataset described in Section 3.4 to minimize the impact of poor initial motion estimates. These estimated biases were used with the MSCKF, but only by propagating the primary states as described in Section 3.3.4, without incorporating any feature measurements or applying any update steps. Figure 19 shows that, when no bias correction is applied, the pitch estimation on the Boxes dataset quickly diverges; roll and yaw also become inaccurate,

though do not diverge as severely as pitch. Position and velocity estimates, though, significantly diverge from the ground truth within the first several seconds. Utilizing a gyroscope bias of $[0.049, 0.012, 0.002]^T\, rad/s$ and an accelerometer bias of $[0.0275, -0.135, 0.14]^T\, m/s^2$ provides the results in Figure 20. The error in orientation was drastically reduced for the entirety of the dataset, remaining within 1° of the ground truth. The position and velocity estimates were also improved, with velocity remaining within an order of magnitude of the ground truth and position diverging much less severely than without bias corrections applied. Note the change in axes limits for the position results in Figures 19 and 20.

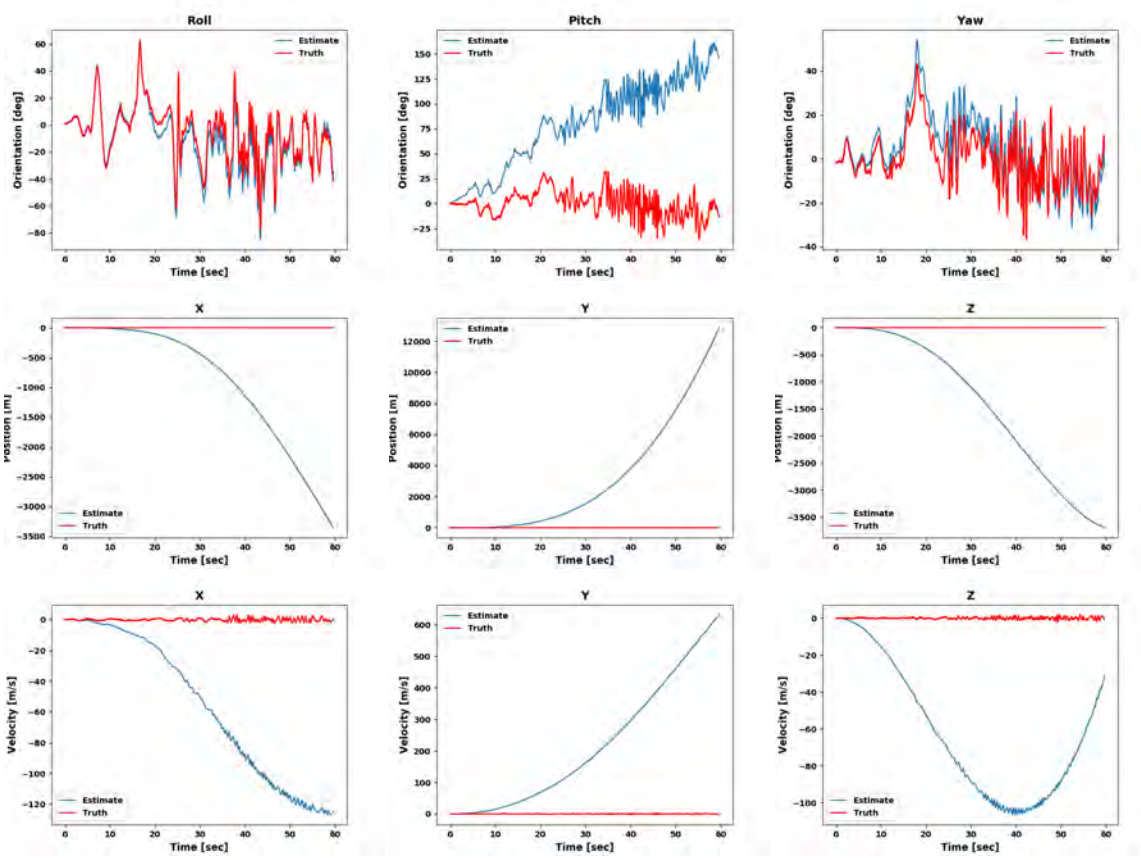

Figure 19: Boxes Dataset, IMU Propagation without Bias Correction: The roll and yaw track the truth relatively close initially, but pitch diverges quickly. The x, y and z velocities also quickly diverge leading to a severe divergence in the position estimation.

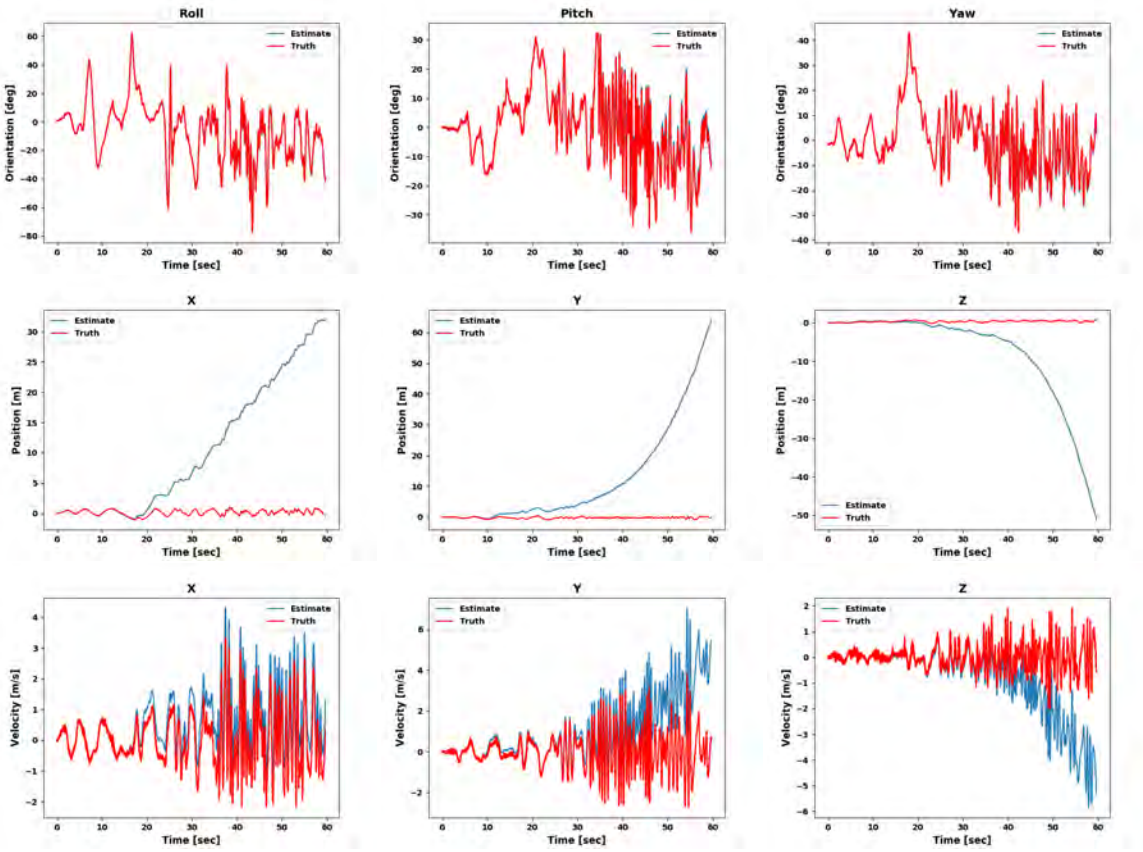

Figure 20: Boxes Dataset, IMU Propagation with Bias Correction: Applying gyroscope and accelerometer bias corrections allows for a much better estimation of orientation and velocity, while tracking position longer before diverging.

The results from propagating the inertial measurement unit (IMU) measurements for the Camp Atterbury dataset, without any bias corrections applied, are shown in Figure 21. Roll, pitch and yaw become inaccurate within the first few seconds, with yaw diverging the least out of the three orientation estimates. The East and Down velocity estimations diverge significantly from the beginning, causing the East and Down positions to diverge after the first several seconds. The velocity and position for the North axis follow the truth very closely until ∼25 seconds into the dataset where they start to diverge. Utilizing a gyroscope bias of $[0.029, 0.0075, -0.007]^T\ rad/s$ and an accelerometer bias of $[-0.06, 0.28, 1.325]^T\ m/s^2$ provides the results in Figure 22. The error in orientation was reduced overall, especially in the first 20 seconds of

the dataset, but became slightly inaccurate in the latter 20 seconds (when the UAV began the 90° turn). The North velocity and position also did not diverge as much as without bias corrections applied. The East and Down velocities and positions made significant improvements, tracking their respective ground truth relatively well compared to when no bias corrections were made.

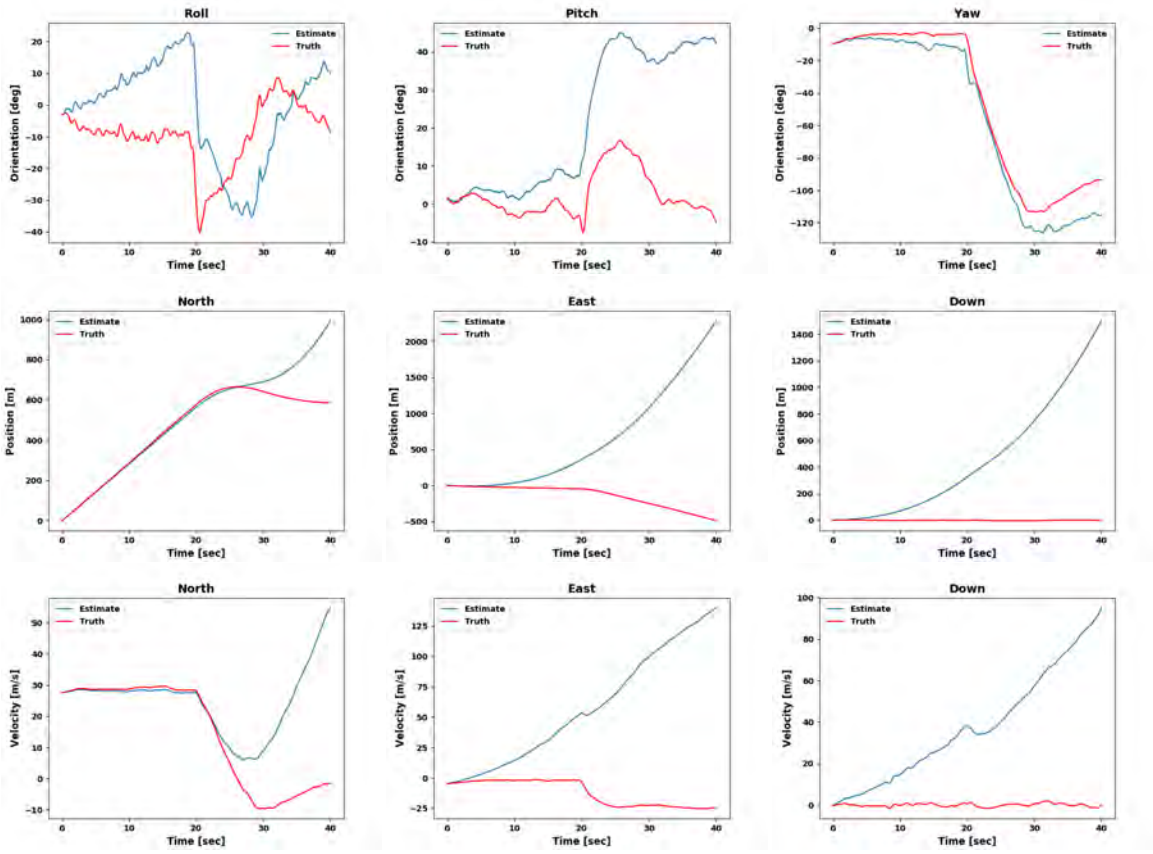

Figure 21: Camp Atterbury Dataset, IMU Propagation without Bias Correction: Each orientation becomes inaccurate quickly, although not completely diverging. The East and Down velocities and position diverge rather quickly, while the North velocity and position do not diverge until the banking turn section of the dataset.

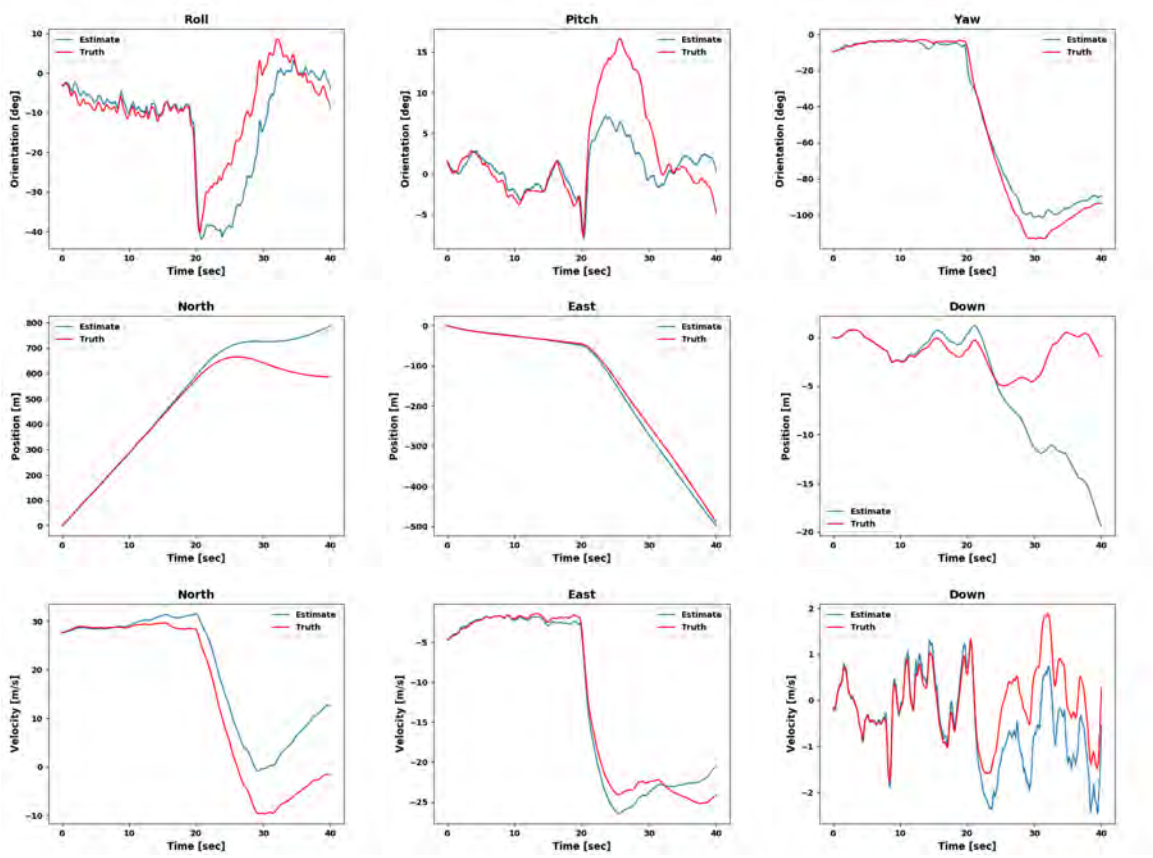

Figure 22: Camp Atterbury Dataset, IMU Propagation with Bias Correction: Applying gyroscope and accelerometer bias corrections allows for a better estimation of orientation and velocity, causing the position to diverge much less severely and less rapidly.

4.5 MSCKF Evaluation

Based on the results presented in Tables 2 to 5, the pre-trained *DetDesc (Round 2)* model was used for the MSCKF evaluation presented in this section, while SIFT was used as a baseline. Additionally, the pre-trained *DetDescEvents (Round 2)* model was used for the Camp Atterbury dataset to assess if a model trained utilizing event-frames offered any performance improvements. Each image was undistorted using provided camera intrinsic parameters. The complete MSCKF was first tested using the grayscale images before testing with event-frames, and the results in the following figures (23 - 32) display absolute orientation and position estimates along with error

plots for orientation, position and velocity. The MSCKF retained the strongest 3 features in each region for a maximum of 60 new features detected in each image.

4.5.1 Boxes Dataset

Figures 23 and 24 show the results of using SIFT and *DetDesc (Round 2)*, respectively, for the Boxes dataset with grayscale images. The velocity and position estimates are much improved for both methods compared to utilizing just the bias corrected IMU measurements. The IMU-only results in Section 4.4 had a root-mean-square error (RMSE) of $29.19m$ and final error of $88.71m$, compared to RMSEs of $1.11m$ and $1.05m$ and final errors of $2.49m$ and $2.1m$, respectively, for SIFT and *DetDesc (Round 2)*. The errors remain within the 3σ bounds for the entirety of the dataset, although the y and z position estimates start to slightly diverge from the ground truth $\sim$35 seconds into the simulation. It can be noted that in the latter part of the Boxes dataset there was an increased amount of rapid and sporadic camera movement which caused significant image blurring and less overlap in sequential images, hindering feature detection and matching. As noted by the RMSEs and as illustrated by the plots, SIFT and *DetDesc (Round 2)* perform on par with each other for the Boxes dataset with grayscale images.

Figures 25 and 26 show the results of using SIFT and *DetDesc (Round 2)*, respectively, for the Boxes dataset with event frames. Although the rapid movement in the latter half of the dataset causes blurry grayscale images, the event camera, being a data driven sensor, is able to more capably detect these rapid changes. This results in the creation of more event frames that eliminate the motion blur and improve feature detection and matching. The velocity errors are reduced and the position estimates are improved, especially in the y and z axes. Furthermore, the filter is more confident in the solution with tighter 3σ bounds on each error plot. Again, SIFT and *DetDesc*

(Round 2) perform on par for the Boxes dataset with event frames with RMSEs of $0.437m$ and $0.457m$ and final errors of $0.968m$ and $1.05m$, respectively. Note that this is $2\times$ improvement over running the same algorithm on the grayscale images, demonstrating the advantages of using an event camera in this scenario.

4.5.2 Camp Atterbury Dataset

Figures 27 to 29 show the results of using SIFT, pre-trained *DetDesc (Round 2)* and pre-trained *DetDescEvents (Round 2)*, respectively, for the Camp Atterbury dataset with grayscale images. The velocity estimates, specifically of the North and Down axes, are much improved compared to the IMU-only propagation results in Figure 22, leading to better position estimates for the complete MSCKF. The IMU-only results in Section 4.4 had a RMSE of $77.18m$ and final error of $203.65m$, compared to RMSEs of $8.96m$, $11.69m$ and $20.01m$ and final errors of $22.85m$, $12.25m$ and $59.84m$, respectively, for SIFT, pre-trained *DetDesc (Round 2)* and pre-trained *DetDescEvents (Round 2)*. Although the errors remain within the 3σ bounds for each method, these bounds are rather large revealing that the filter is not confident in its solution despite the successful results of the absolute position. Each feature detector also performs similarly overall with *DetDescEvents (Round 2)* slightly under-performing the other two.

Figures 30 to 32 show the results of using SIFT, pre-trained *DetDesc (Round 2)* and pre-trained *DetDescEvents (Round 2)*, respectively, for the Camp Atterbury dataset with event frames. The Camp Atterbury dataset contained much sparser scenes with low terrain compared to the Boxes dataset, especially in the latter half, causing the event frames to be much noisier than the grayscale images. As a result, the performance of the MSCKF is poorer, particularly in the East axis once the UAV stars turning, for each feature detector. The methods of using SIFT, pre-trained

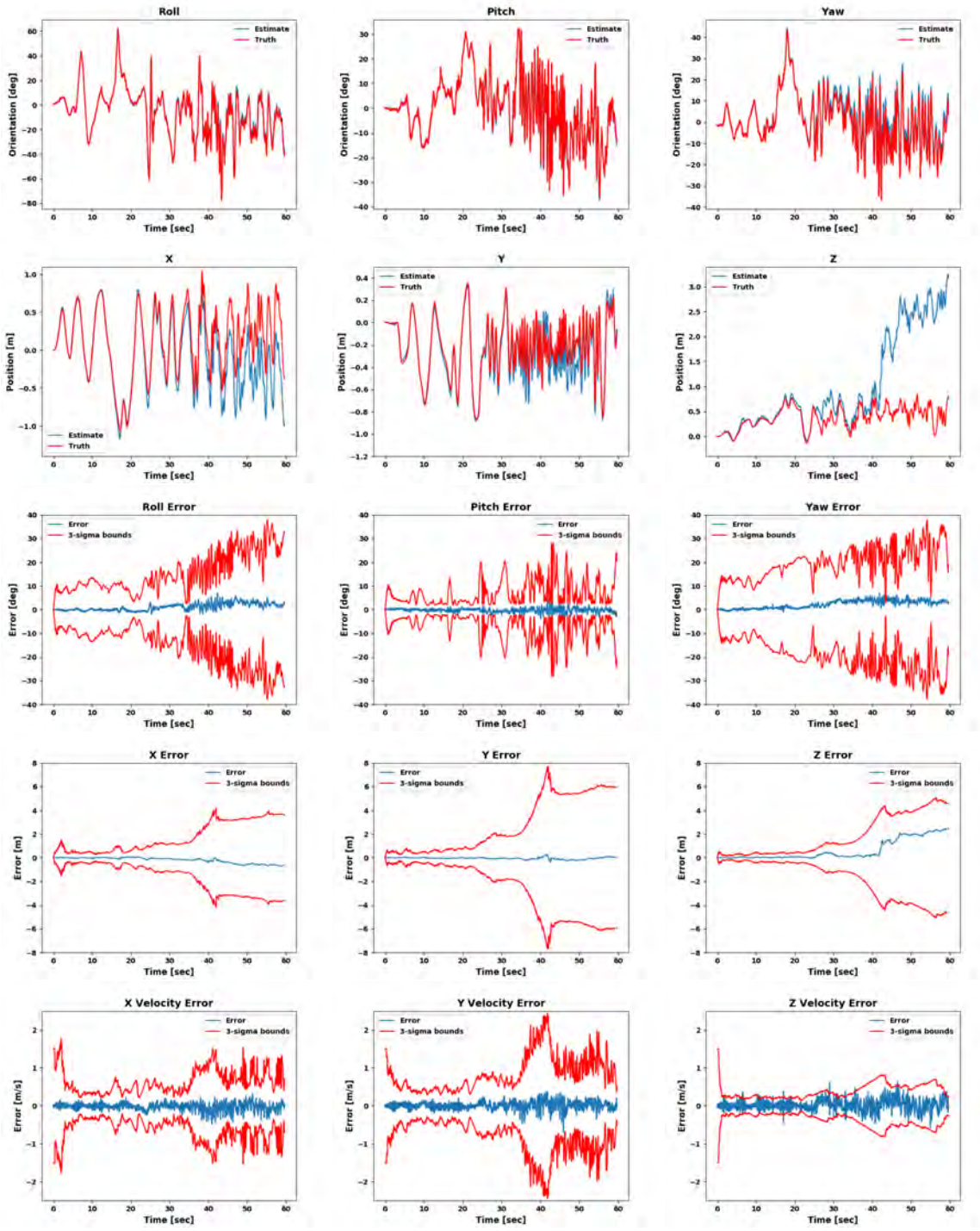

Figure 23: Boxes Dataset, MSCKF Results with Grayscale Images and SIFT: Position and velocity estimates are much improved over IMU-only propagation. The y and z position estimation starts to diverge slightly as the camera movement becomes rapid and sporadic in the latter half of the dataset, causing blurry grayscale images.

DetDesc (Round 2) and pre-trained *DetDescEvents (Round 2)*, respectively, resulted in RMSEs of $34.12m$, $22.77m$ and $30.39m$ and final errors of $77.17m$, $46.57m$ and $86.26m$. Each feature detector performs similarly overall with pre-trained *DetDesc (Round 2)* slightly outperforming the others. It can be noted that although *DetDescEvents (Round 2)* was trained with event frames from the Camp Atterbury dataset, it does not offer any performance gains compared to *DetDesc (Round 2)*.

Table 6 contains a comprehensive list of all the quantitative MSCKF results − RMSE, final error and percent error based on distance travelled − for each scenario mentioned in Sections 4.4 and 4.5. Percent error is calculated by computing the absolute error at each time step and dividing by the total distance traveled, and then taking the average value across all time steps [79]. Additionally, Appendix C contains the estimated gyroscope and accelerometer biases with 3σ bounds for each scenario presented in this section.

Table 6: MSCKF Results

Boxes Dataset [1] - Distance Travelled: 69.8 m				
Model	**Frame Type**	**RMSE [m]**	**Final Error [m]**	**% Error** [79]
IMU-only (No Bias Corrections)	-	5383.34	13804.04	5177.67 %
IMU-only (Bias Corrections)	-	29.19	88.71	26.34 %
SIFT	Grayscale	1.11	2.49	0.989 %
Pre-trained DetDesc (Round 2)	Grayscale	1.05	2.1	0.996 %
SIFT	Event	0.437	0.968	0.447 %
Pre-trained DetDesc (Round 2)	Event	0.457	1.05	0.44 %
Camp Atterbury Dataset - Distance Travelled: 1076.82 m				
Model	**Frame Type**	**RMSE [m]**	**Final Error [m]**	**% Error**
IMU-only (No Bias Corrections)	-	1290.34	3184.52	83.3 %
IMU-only (Bias Corrections)	-	77.18	203.65	4.54 %
SIFT	Grayscale	8.96	22.85	0.645 %
Pre-trained DetDesc (Round 2)	Grayscale	11.69	12.25	1.09 %
Pre-trained DetDescEvents (Round 2)	Grayscale	20.01	59.84	1.28 %
SIFT	Event	34.12	77.17	2.36 %
Pre-trained DetDesc (Round 2)	Event	22.77	46.57	1.64 %
Pre-trained DetDescEvents (Round 2)	Event	30.39	86.26	2.0 %

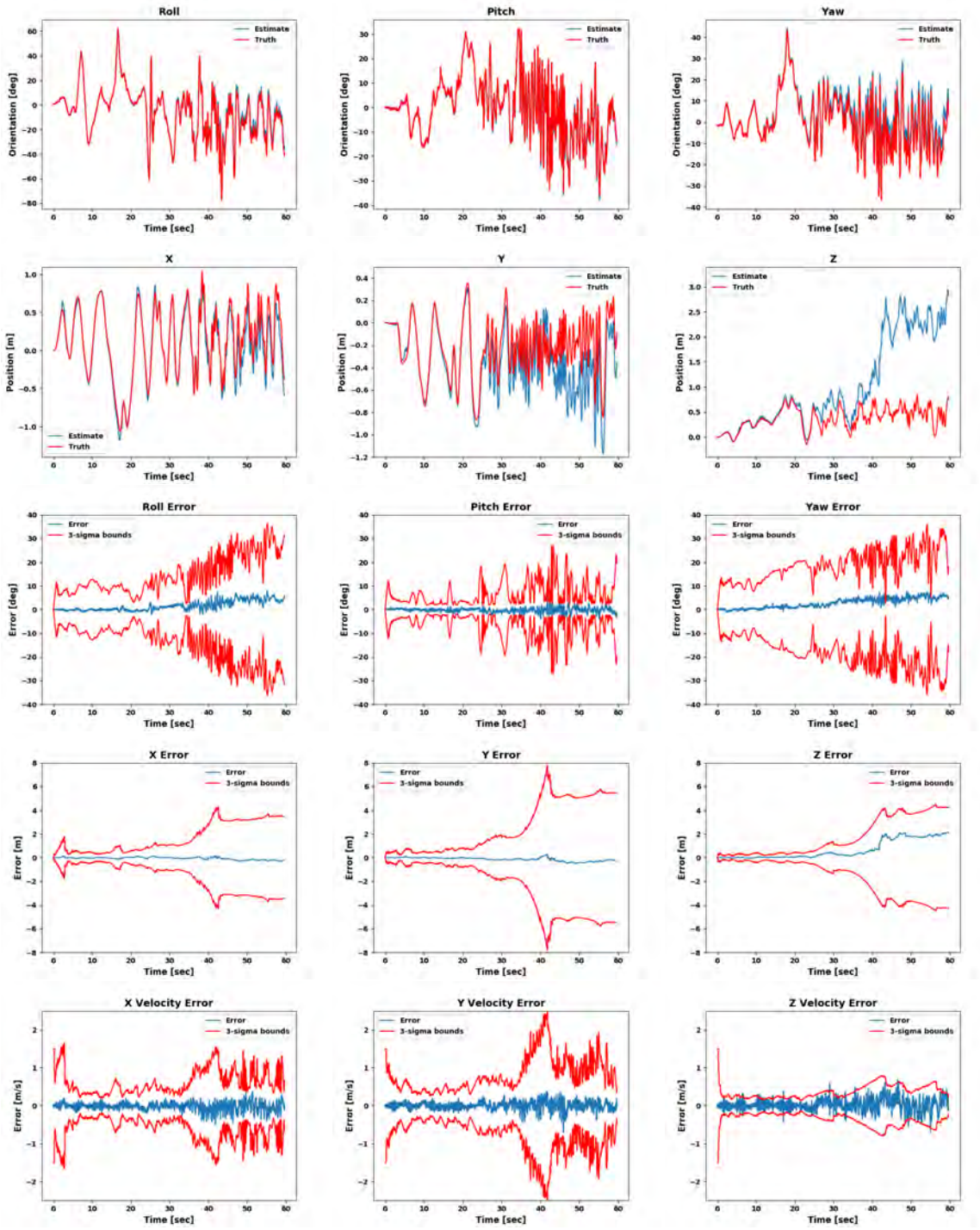

Figure 24: Boxes Dataset, MSCKF Results with Grayscale Images and pre-trained *DetDesc (Round 2)*: Position and velocity estimates are much improved over IMU-only propagation. The y and z position estimation starts to diverge slightly as the camera movement becomes rapid and sporadic in the latter half of the dataset, causing blurry grayscale images.

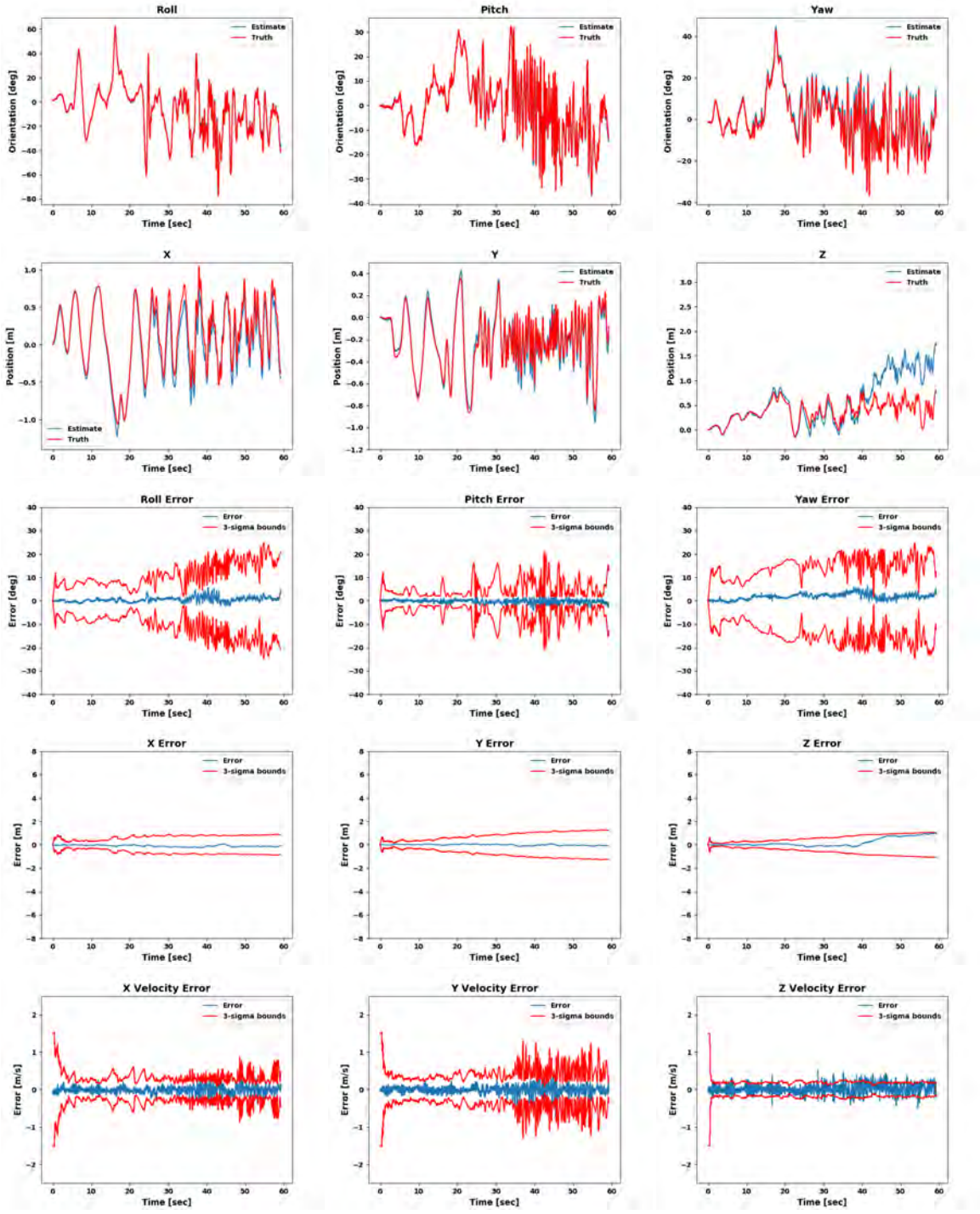

Figure 25: Boxes Dataset, MSCKF Results with Event Frames and SIFT: The event camera allows for the creation of non-blurred images during rapid movement, allowing for more efficient feature detection and matching. As a result, velocity and position estimates are improved and the filter is more confident in its solution.

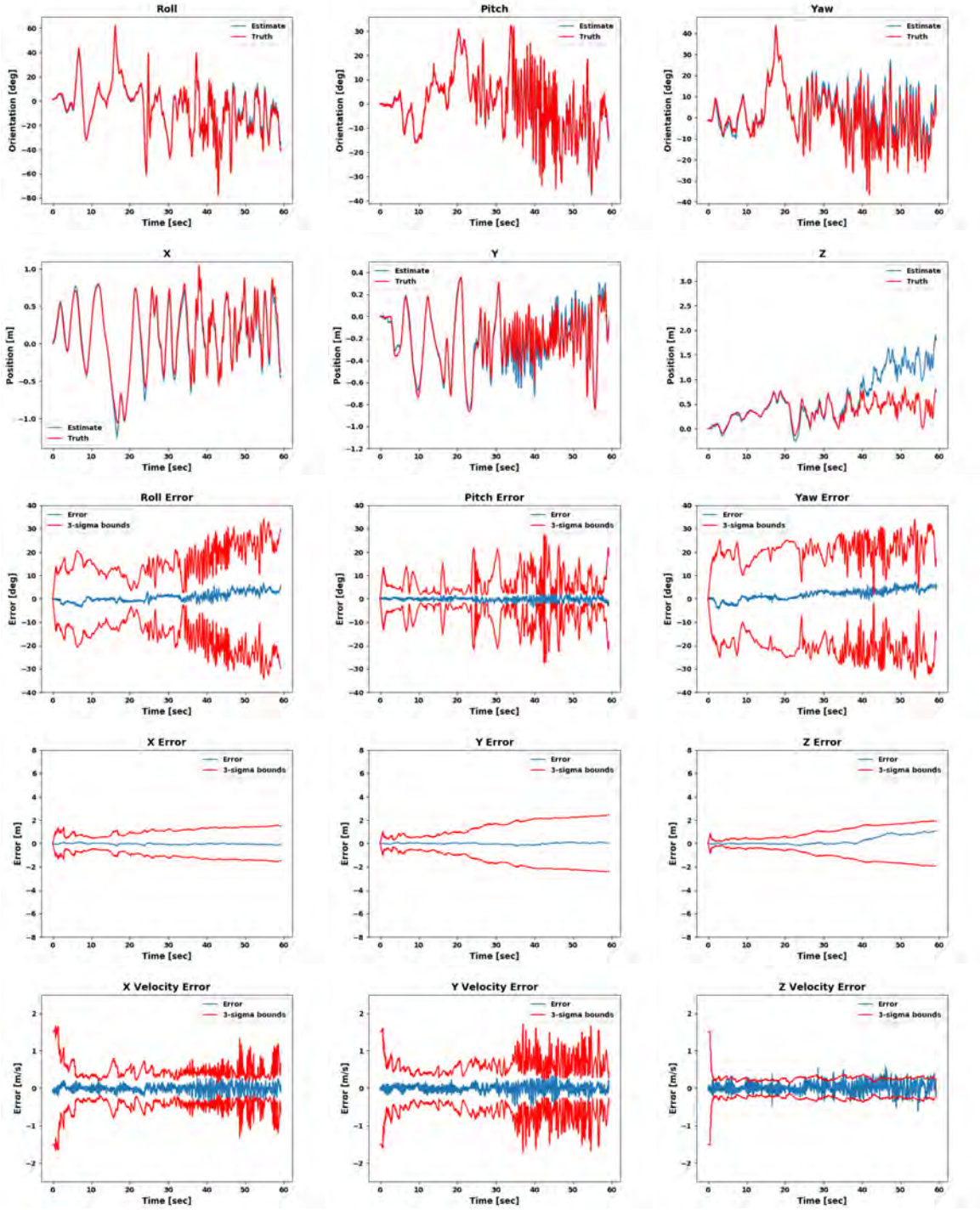

Figure 26: Boxes Dataset, MSCKF Results with Event Frames and pre-trained *DetDesc (Round 2)*: The event camera allows for the creation of non-blurred images during rapid movement, allowing for more efficient feature detection and matching. As a result, velocity and position estimates are improved and the filter is more confident in its solution.

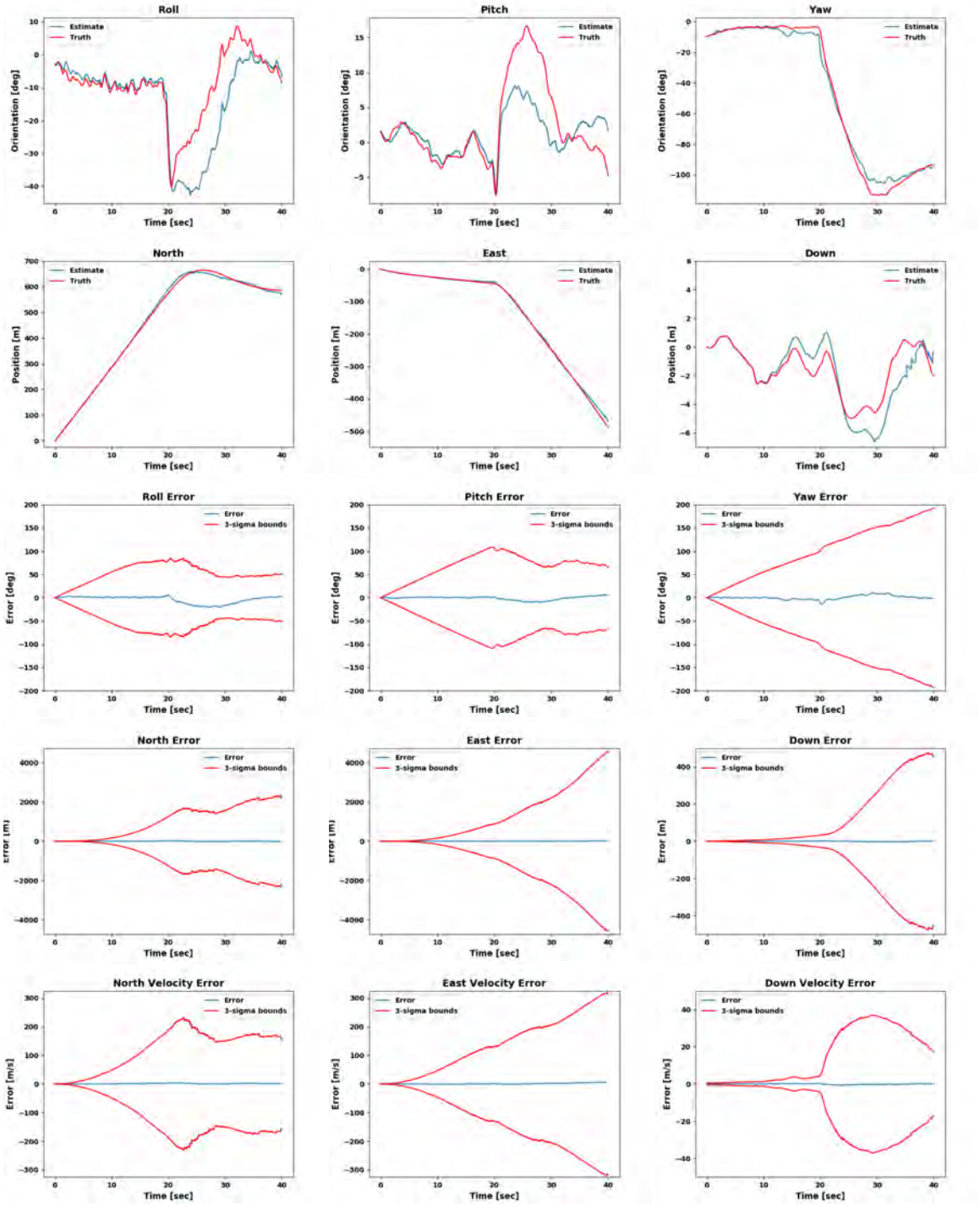

Figure 27: Camp Atterbury Dataset, MSCKF Results with Grayscale Images and SIFT: Position and velocity estimates are improved over IMU-only propagation, especially in the North and Down axes.

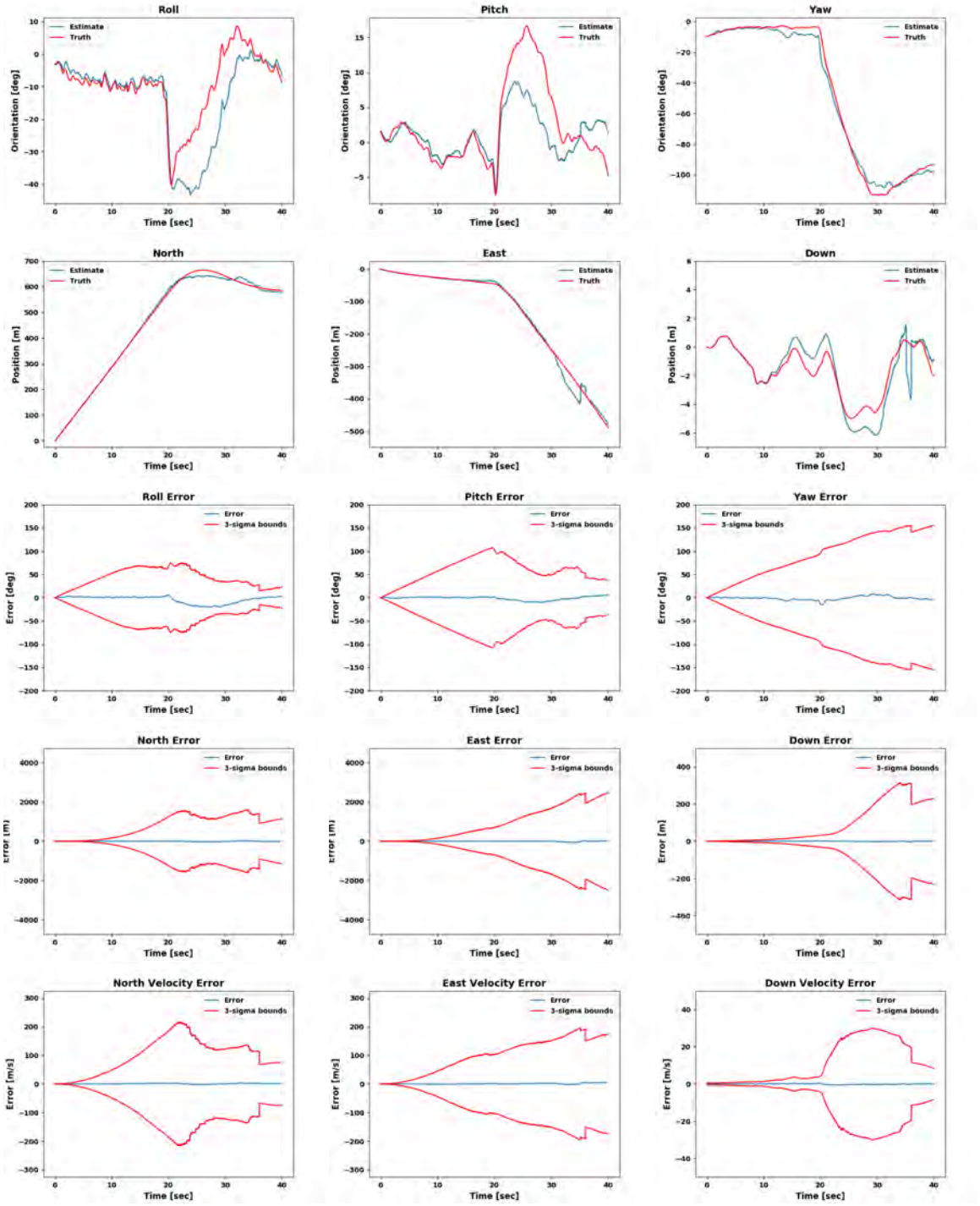

Figure 28: Camp Atterbury Dataset, MSCKF Results with Grayscale Images and pre-trained *DetDesc (Round 2)*: Position and velocity estimates are improved over IMU-only propagation, especially in the North and Down axes.

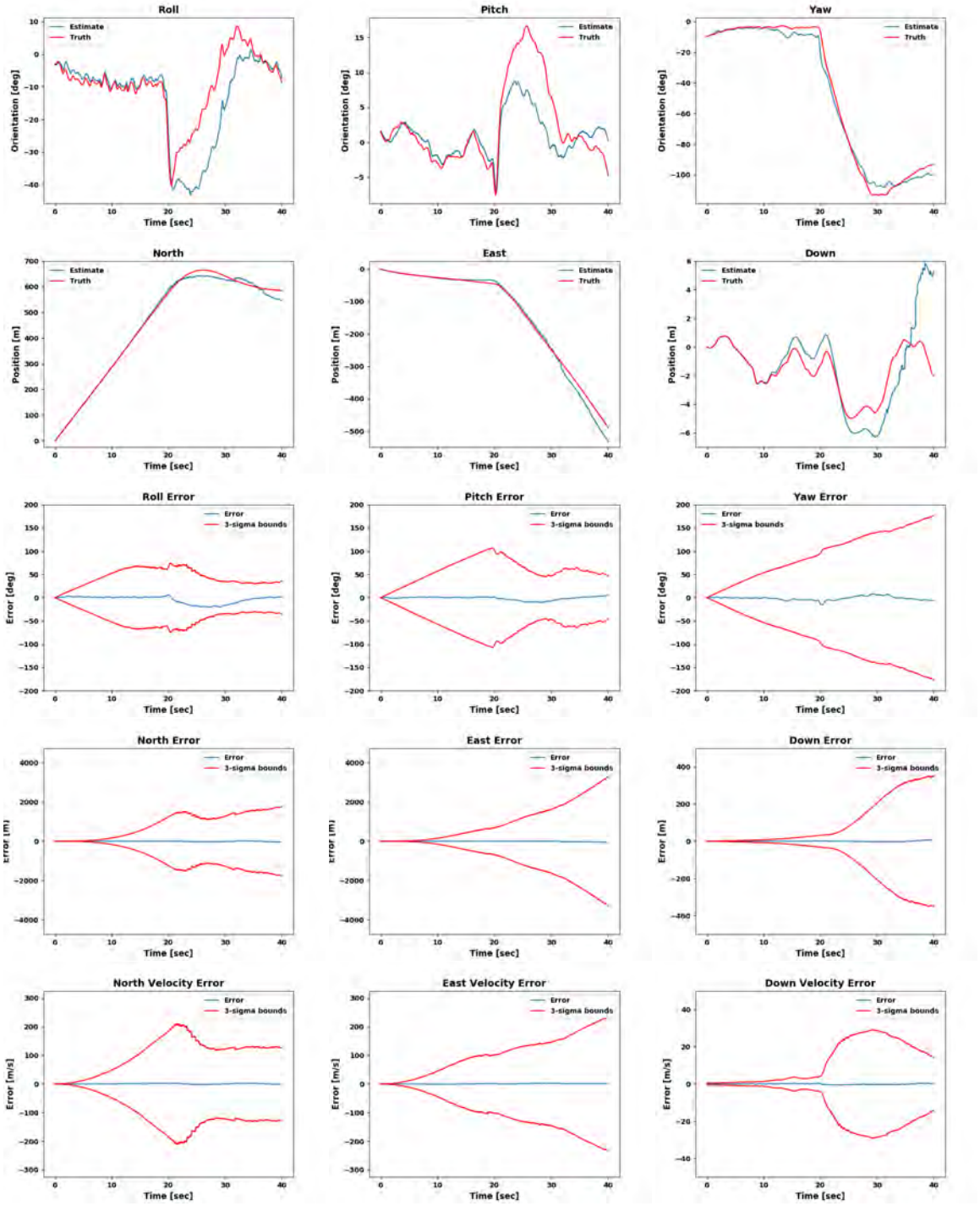

Figure 29: Camp Atterbury Dataset, MSCKF Results with Grayscale Images and pre-trained *DetDescEvents (Round 2)*: Position and velocity estimates are improved over IMU-only propagation, especially in the North and Down axes.

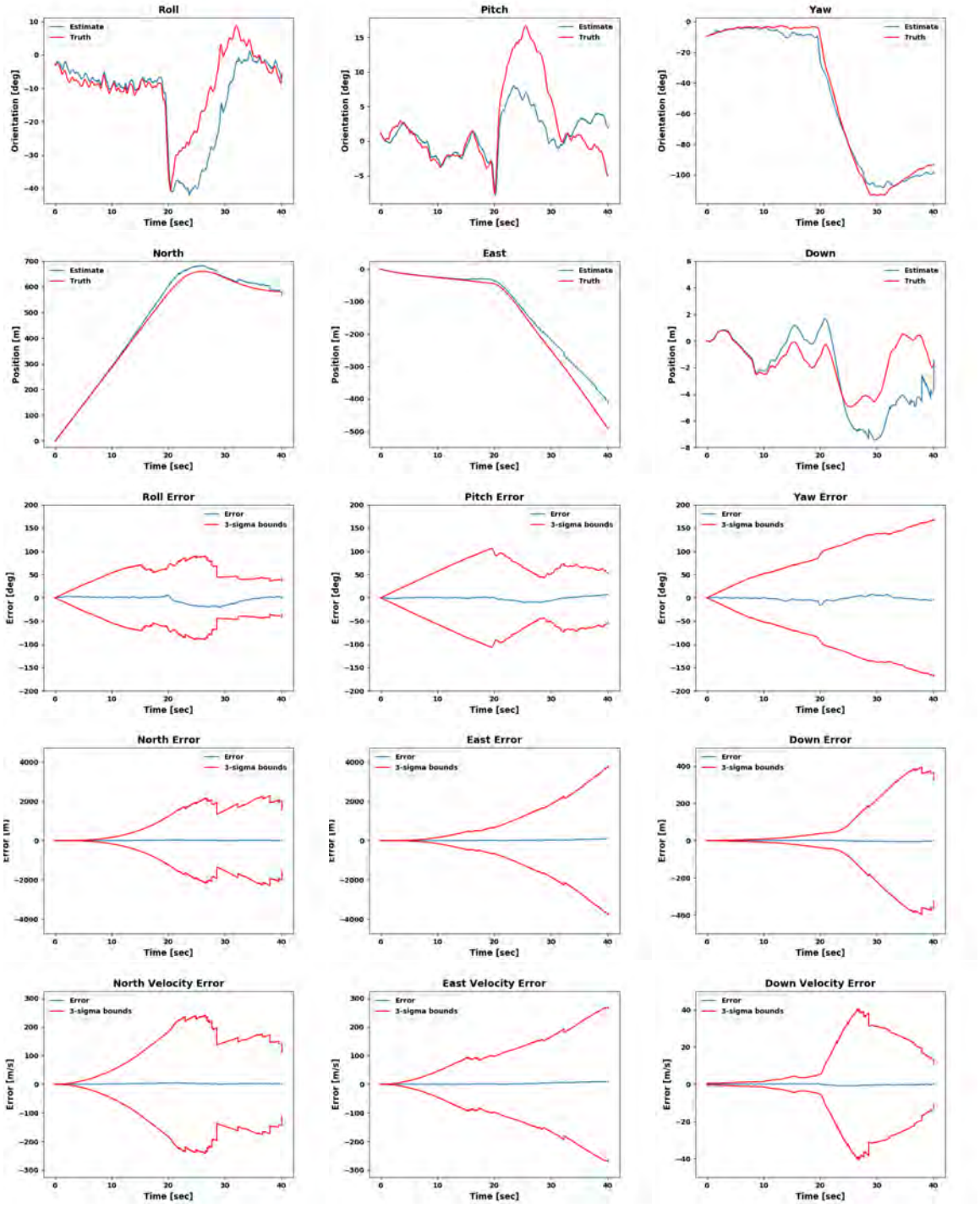

Figure 30: Camp Atterbury Dataset, MSCKF Results with Event Frames and SIFT: The sparse scene throughout the Camp Atterbury dataset causes noisy event frames, which hinders feature detection and matching. As a result, the position estimates are not as accurate compared to using grayscale images.

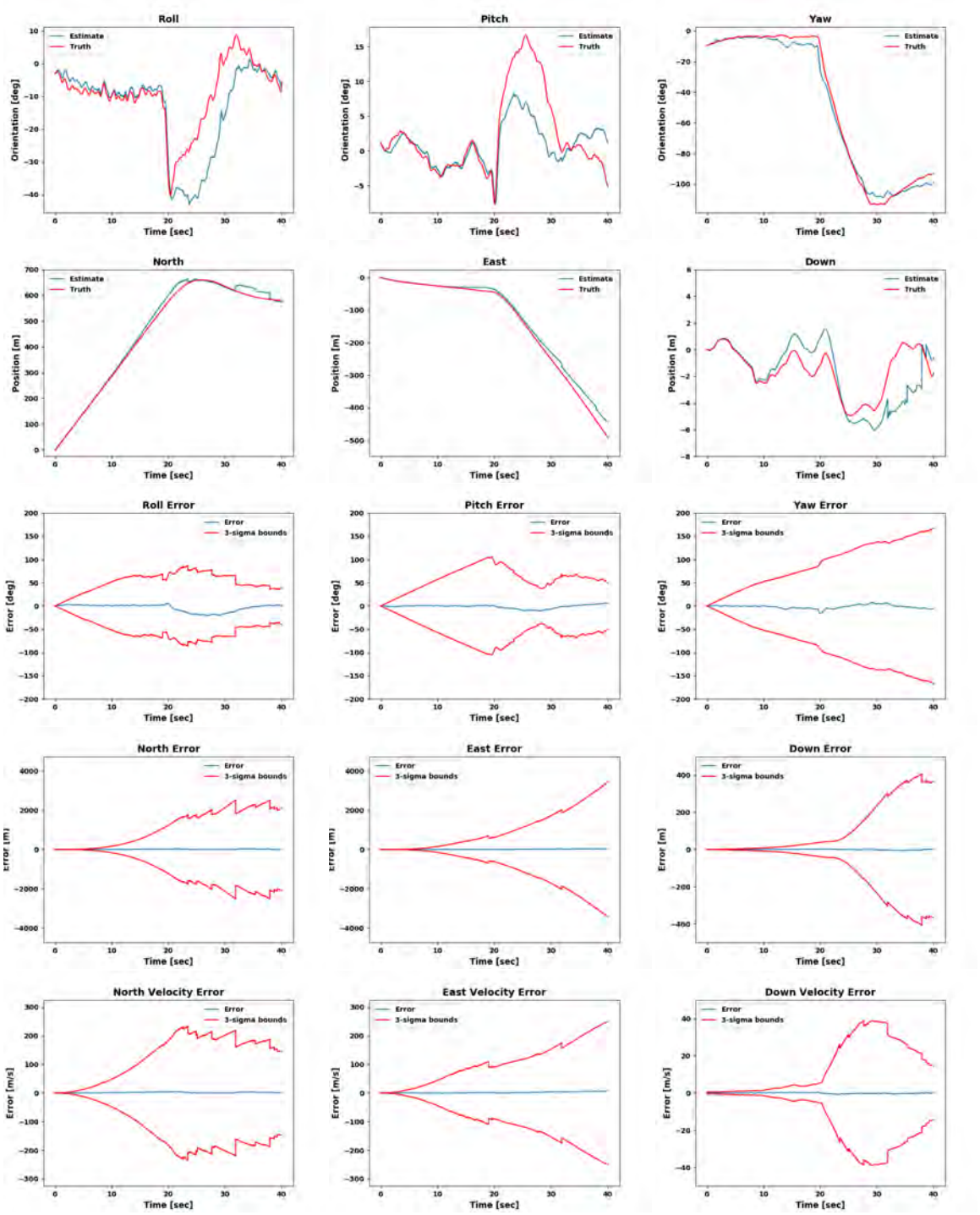

Figure 31: Camp Atterbury Dataset, MSCKF Results with Event Frames and pre-trained *DetDesc (Round 2)*: The sparse scene throughout the Camp Atterbury dataset causes noisy event frames, which hinders feature detection and matching. As a result, the position estimates are not as accurate compared to using grayscale images.

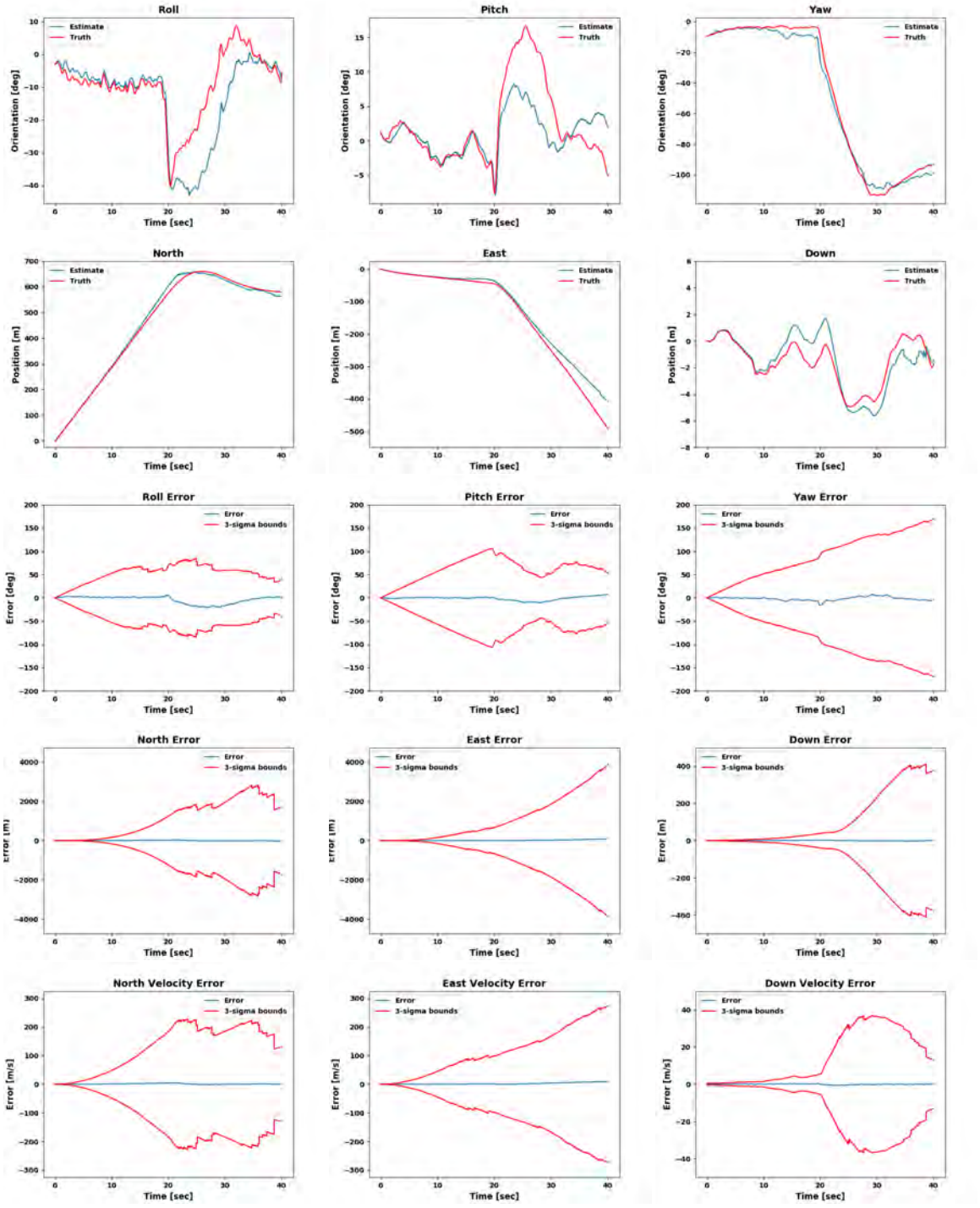

Figure 32: Camp Atterbury Dataset, MSCKF Results with Event Frames and pre-trained *DetDescEvents (Round 2)*: The sparse scene throughout the Camp Atterbury dataset causes noisy event frames, which hinders feature detection and matching. As a result, the position estimates are not as accurate compared to using grayscale images.

V. Conclusions

In this thesis, an event-based visual-inertial odometry (EVIO) pipeline, primarily motivated by the research in [18, 19], was implemented and tested. The front-end of this pipeline was fed by the output of a novel type of visual sensor, called an event-based camera, where each pixel independently and asynchronously outputs ON or OFF events for a rise or fall in log-level light intensity past an established threshold. Batches of event output from the event-based camera were converted into image frames by utilizing the artificial neural network (ANN) developed in [16, 17]. Features were generated and matched using a trained convolutional neural network (CNN), inspired by the research in [11], to produce more reliable and repeatable feature tracks to be used by the back-end of the EVIO pipeline.

The back-end of the pipeline utilized a multi-state constraint Kalman filter (MSCKF), which estimates errors for a vehicle state calculated by an inertial navigation system (INS) mechanization of an inertial measurement unit (IMU). A least-squares estimation of a 3-D feature position in a given feature track is then used to calculate the residual for Kalman filter update equations, allowing an update to the drifting IMU solution.

Prior to incorporating the CNN into the full visual odometry (VO) pipeline, it was evaluated separately using repeatability and homography estimation metrics on two different datasets, HPatches [70] and event-based aerial imagery from a fixed-wing unmanned aerial vehicle (UAV). Results show that the CNN performs the best in repeatability at strict threshold values ($\epsilon = 1$) for both datasets when considering viewpoint changes, but slightly worse than the Harris [8] corner detector at higher thresholds. The advantage of the CNN is its ability to repeatably detect features under illumination changes, in which it performs the best at all threshold values. Furthermore, the CNN under-performs scale invariant feature transform (SIFT) in

homography estimation for viewpoint scenes at $\epsilon = 1$, while performing competitively at higher thresholds. The CNN again performs the best under illumination changes for homography estimation.

Additionally, when comparing the different variations of the trained models to each other it was revealed that having too long of a training process can result in over fitting, and training multiple rounds of a detector-only model before the full model does not produce higher repeatability or homography estimation metrics. This led to different conclusions than in [11] about how to train a feature detector and descriptor network. Finally, training models that were tailored to event-based aerial imagery did not offer any performance benefits over models not tailored to that type of data. The performance metrics mentioned, however, are not completely indicative of how well each feature detector will perform in the full EVIO pipeline.

The results for the implementation of the MSCKF fed with event frames produced better vehicle state estimations than utilizing grayscale images for the Boxes dataset, but poorer estimations for the Camp Atterbury dataset. This was due to the Boxes dataset producing a high event rate, especially in the latter half, allowing the creation of a high frames-per-second (fps) output that did not suffer from motion blur, which was present in the grayscale frames. On the other hand, the Camp Atterbury dataset contained sparse scenes without any rapid movement, producing a low event rate, which is not ideal for event frame reconstruction. As a result, the event frames were noisy, hindering reliable feature matching. Additionally, the trained models did not seem to offer any significant advantages over SIFT in the EVIO pipeline for both datasets, using both grayscale and event frames.

5.1 Future Work

Adjustments to the CNN training and MSCKF implementation, as well as future evaluation methods that were not included in this research due to time constraints include:

- Collect and evaluate the EVIO pipeline on a flight test dataset taken with low lighting conditions and/or fast maneuvers to see if the event camera/CNN combination offers significant improvement over a classical camera/detector combination.
- Utilize the CNN in visual localization algorithms that compare query images to a reference dataset that spans seasons or daytime changes to emphasize the strengths of the trained model.
- Investigate feature detection methods that utilize event camera output directly, instead of reconstructing images.
- Perform a larger hyper-parameter sweep when training the models, to include different optimizers, learning rates, weight regularization and reduced architecture size.
- Utilize different training datasets or mix multiple available image datasets into one training set.
- Add other evaluation metrics, such as mean average precision, localization accuracy and matching score, to compliment repeatability and homography estimation and get a more informed evaluation of the feature detectors and descriptors.
- Train the model to detect blobs in addition to corners and edges to get a more generalized feature detector.

- Implement the method in [18] of marginalizing one-third of the camera states once the camera state buffer is full to see the performance differences.
- Implement a method that adjusts the value of n based on the event rate when reconstructing event frames to tailor the fps output to the current scene.

Appendix A. Training Results

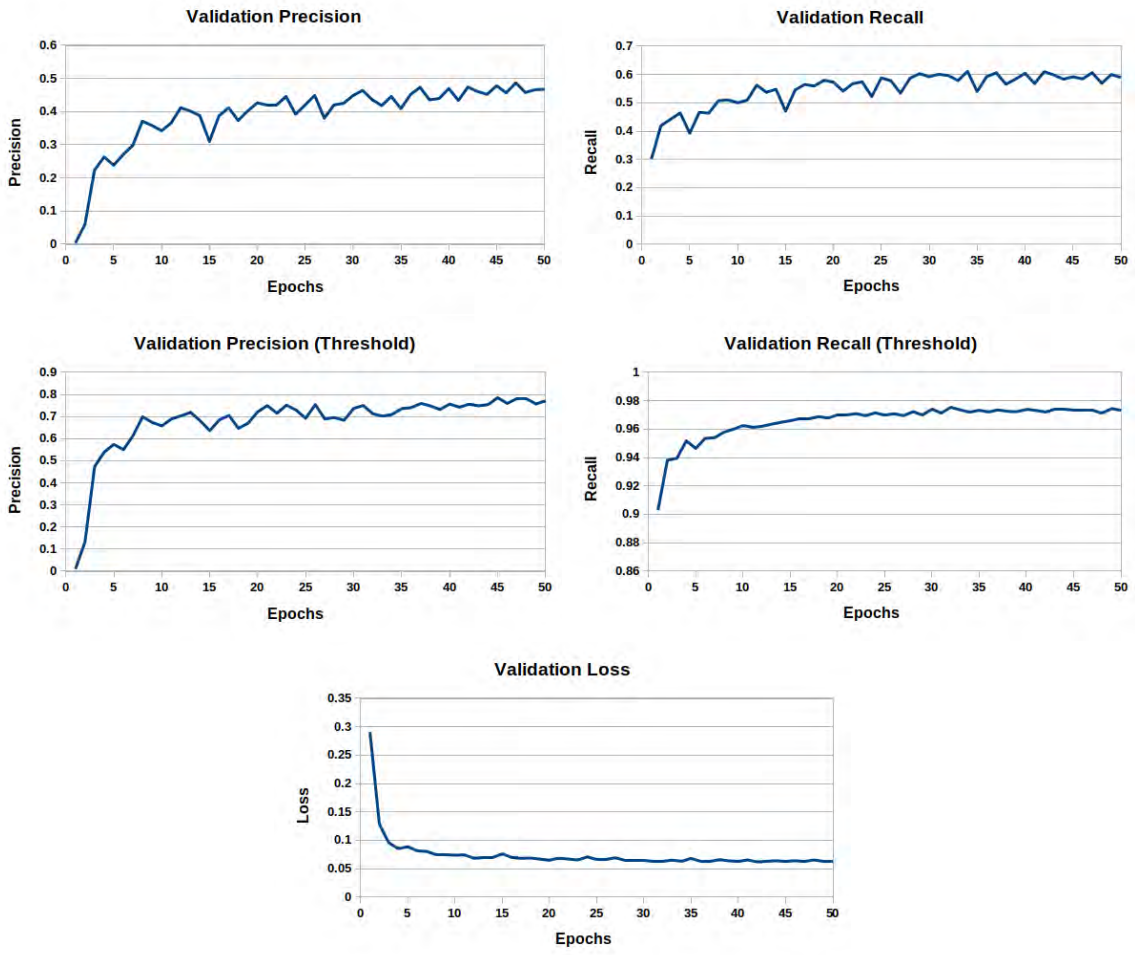

Figure 33: *Synthetic Shapes* Training: Validation set results over 50 epochs.

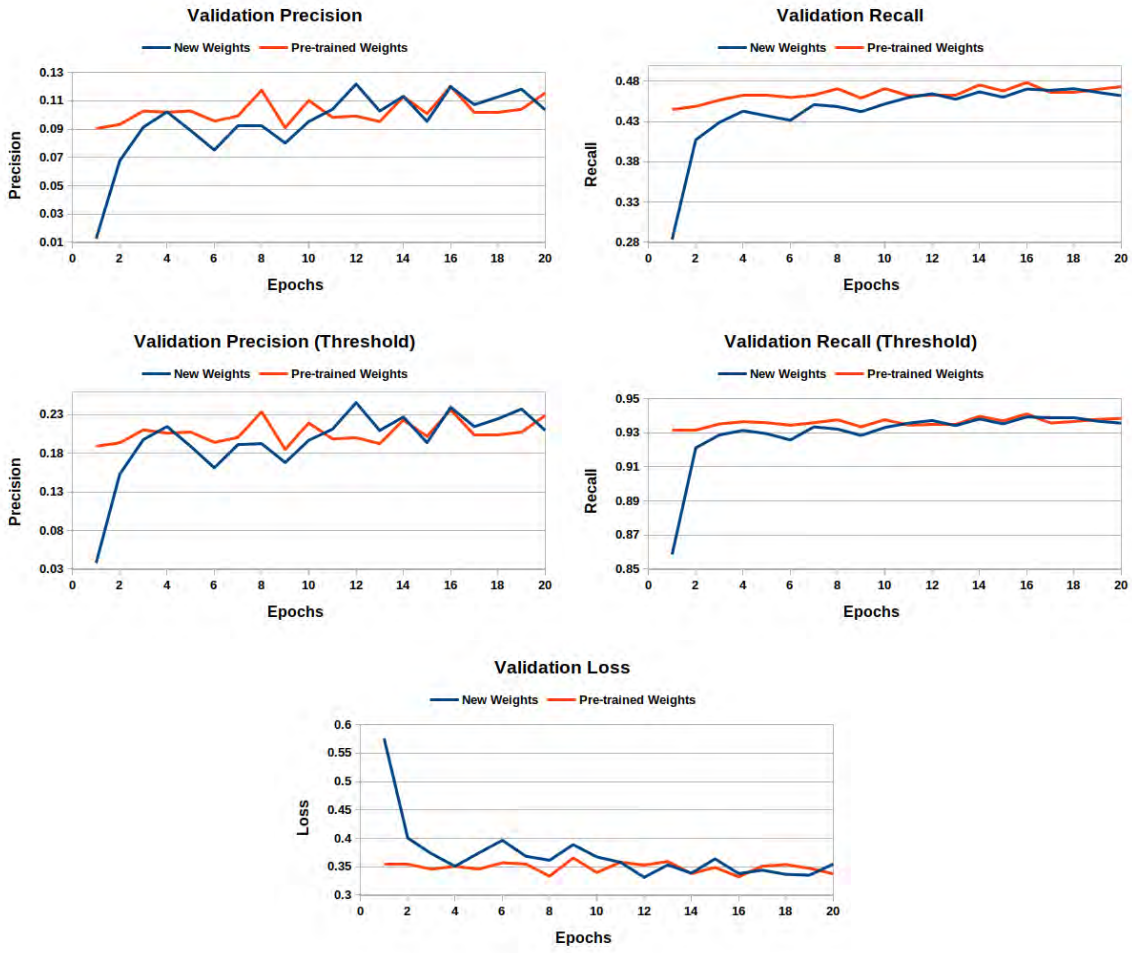

Figure 34: *Detector (Round 1)* Training: Validation set results over 20 epochs.

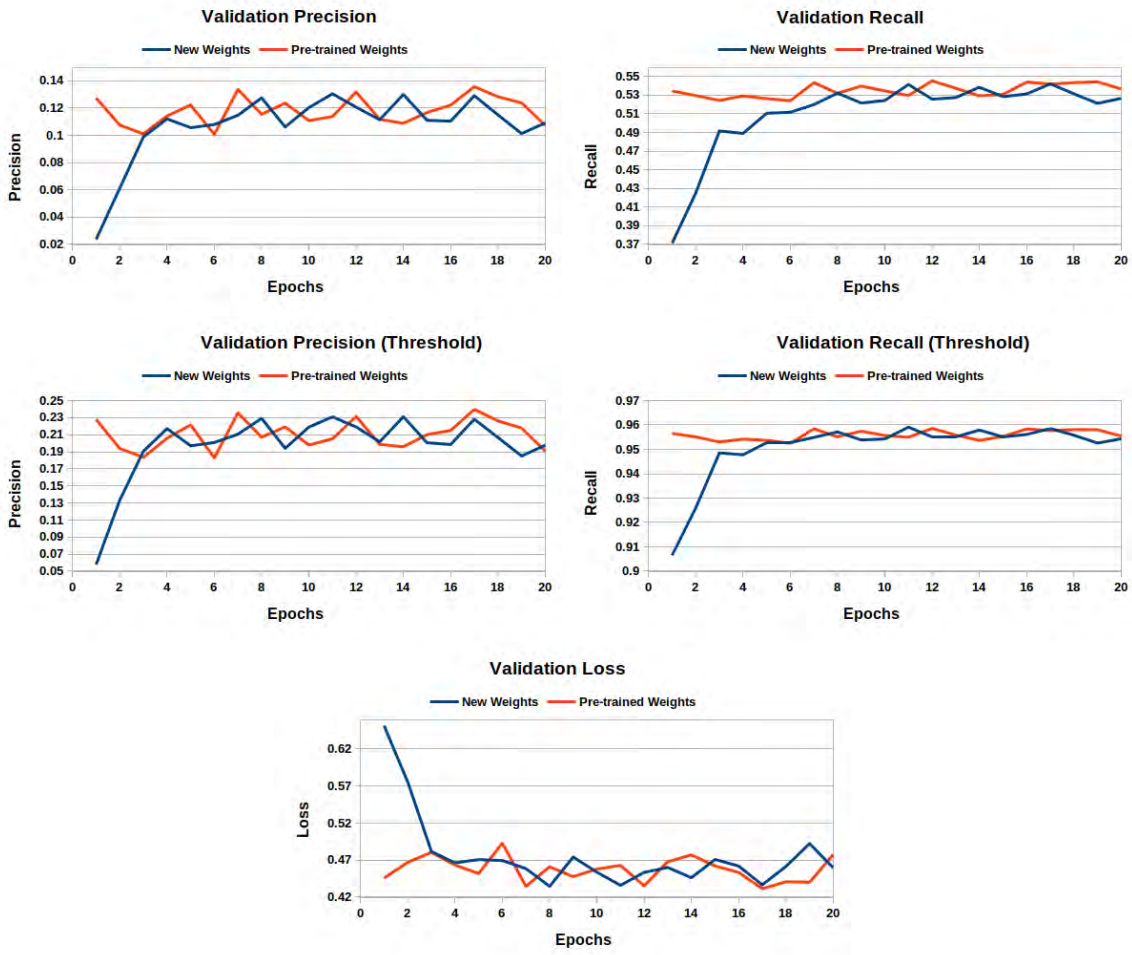

Figure 35: *Detector (Round 2)* Training: Validation set results over 20 epochs.

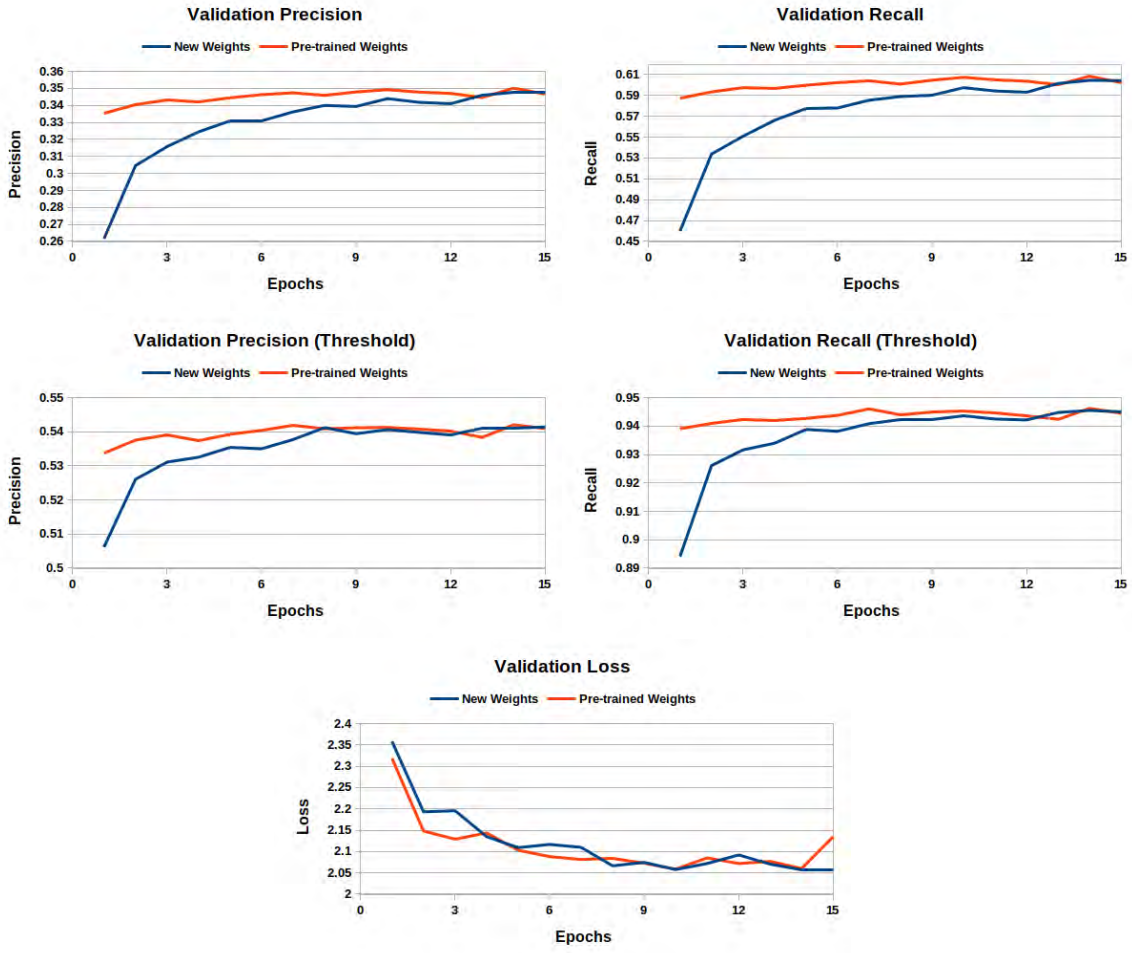

Figure 36: *DetDesc* Training: Validation set results over 15 epochs.

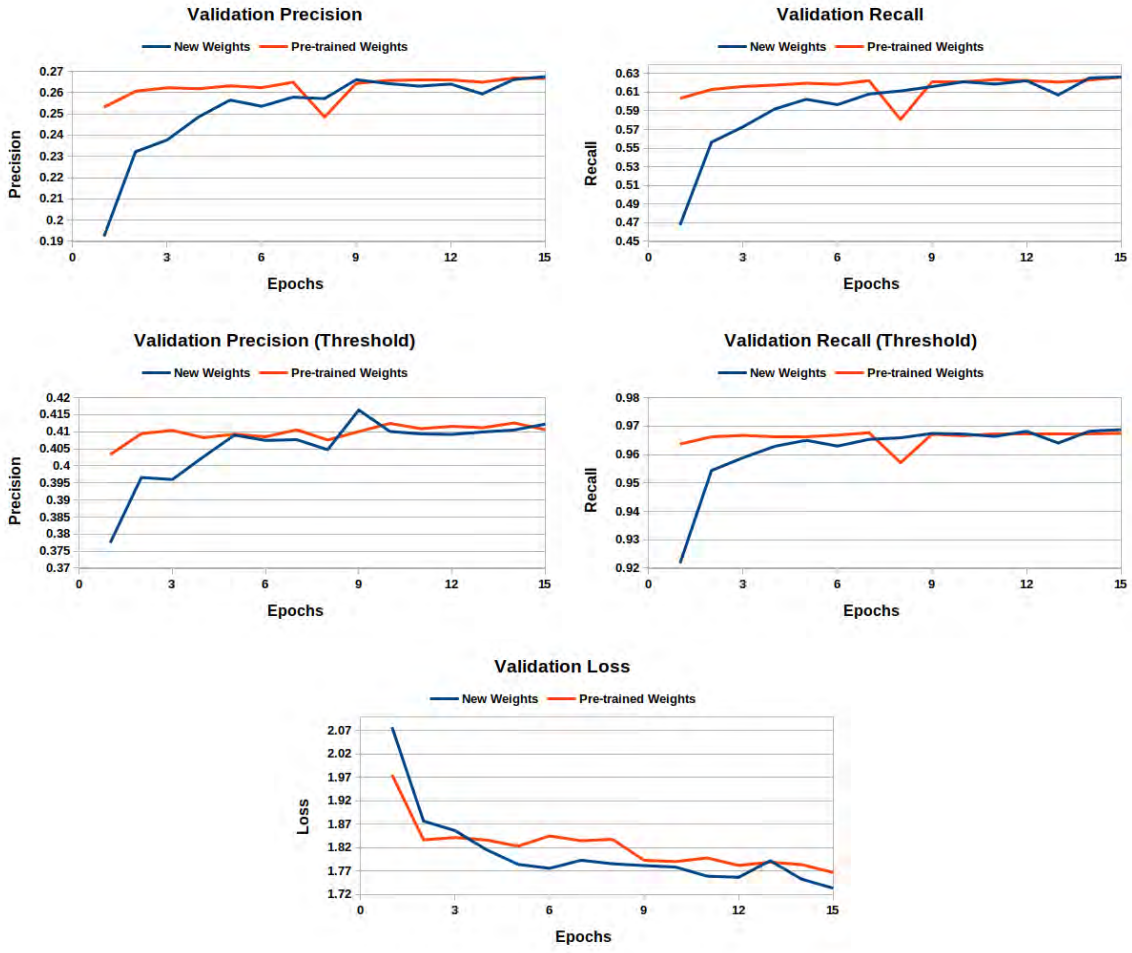

Figure 37: *DetDescLite* Training: Validation set results over 15 epochs.

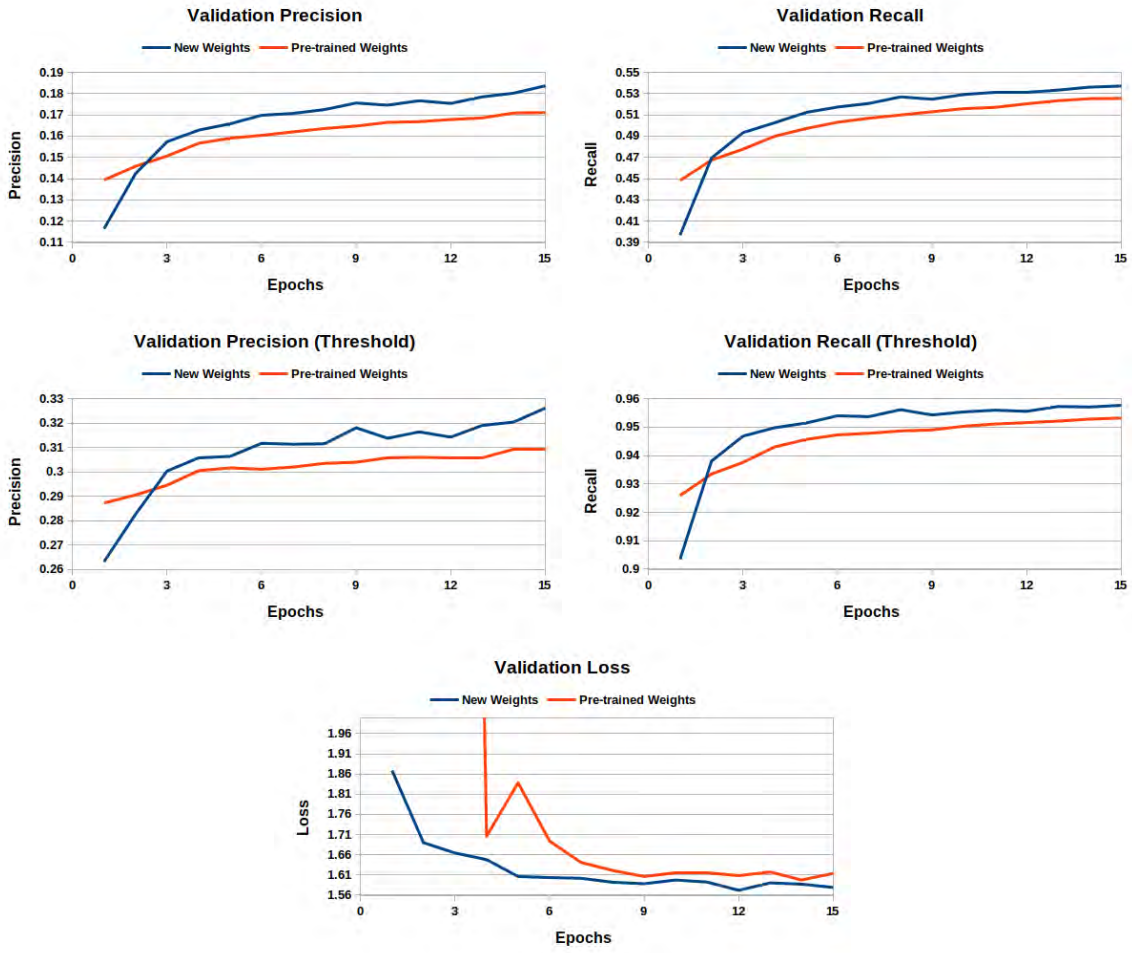

Figure 38: *DetDesc (Round 1)* Training: Validation set results over 15 epochs.

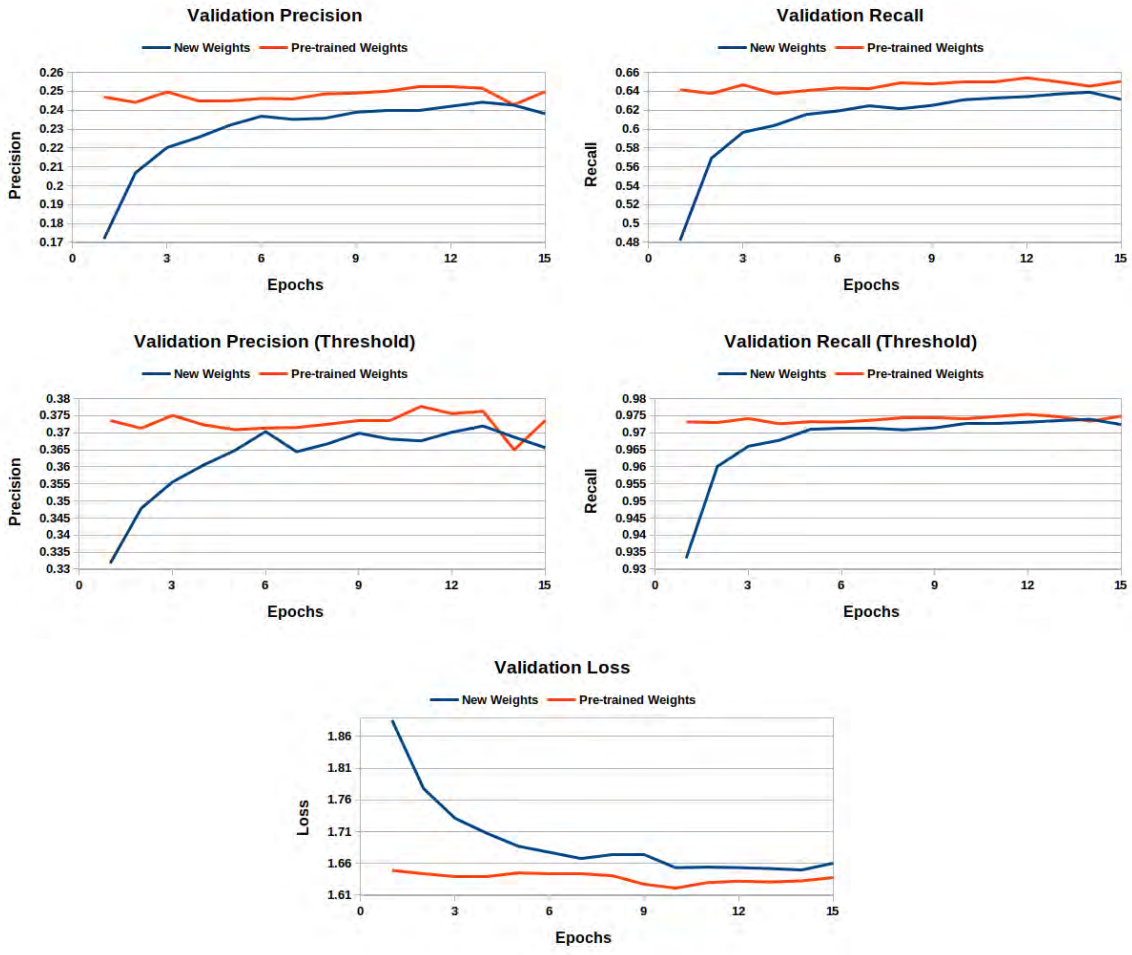

Figure 39: *DetDesc (Round 2)* Training: Validation set results over 15 epochs.

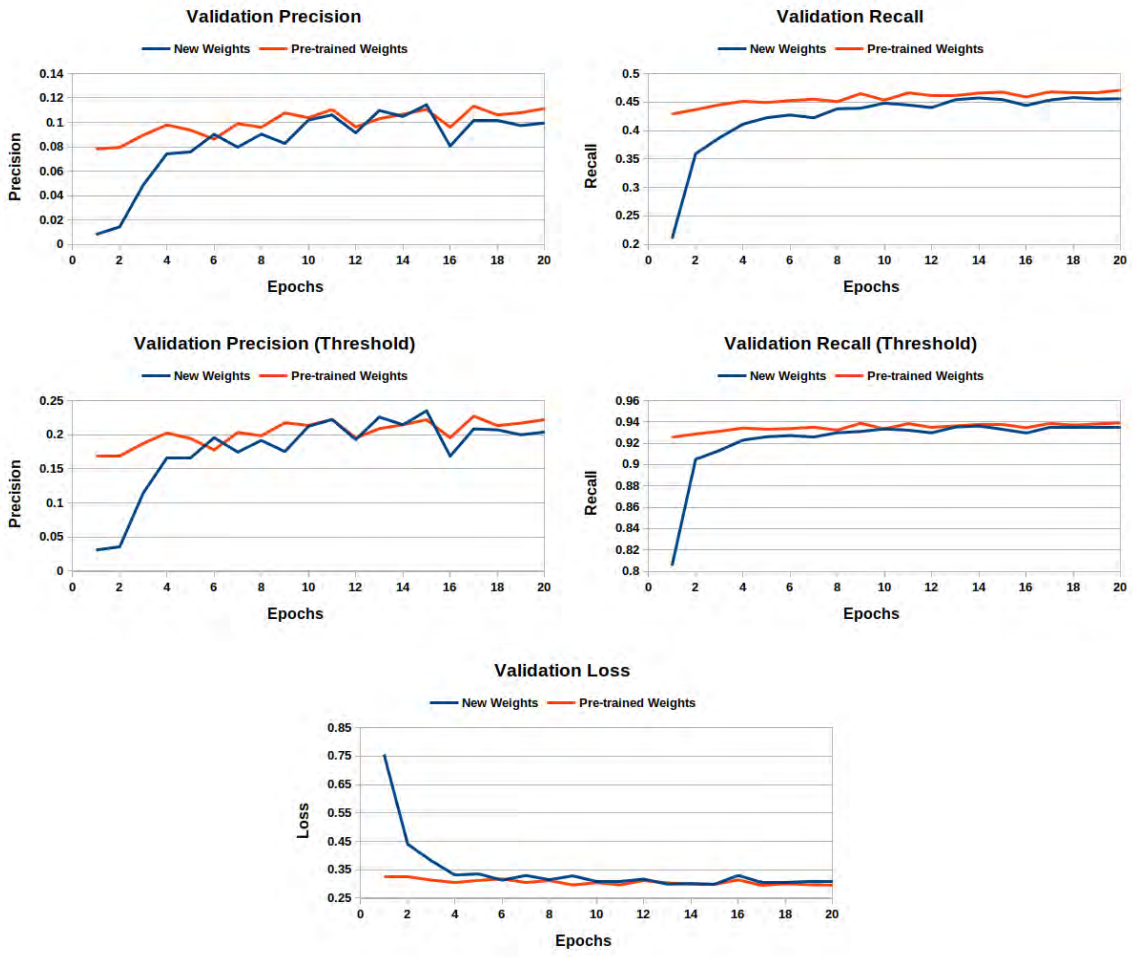

Figure 40: *DetectorEvents (Round 1)* Training: Validation set results over 20 epochs.

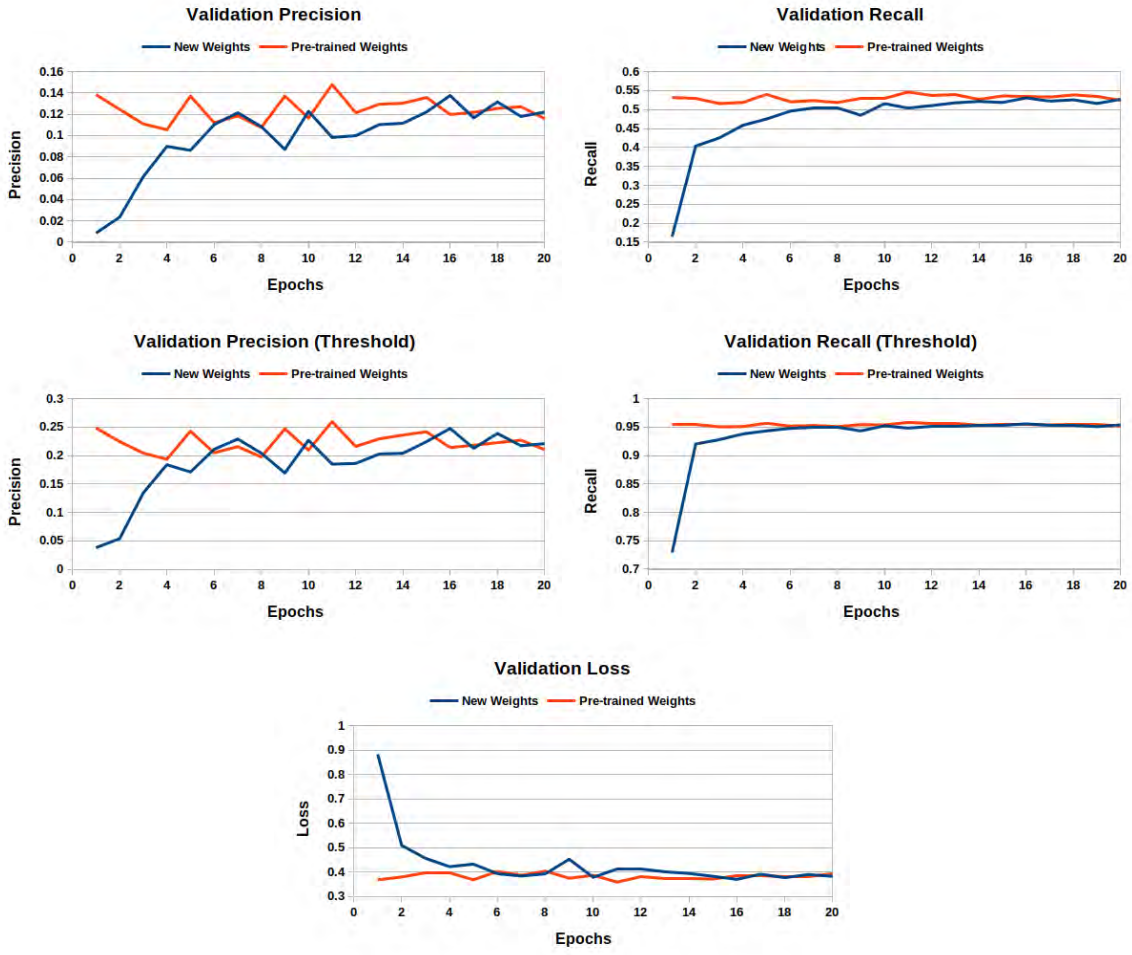

Figure 41: *DetectorEvents (Round 2)* Training: Validation set results over 20 epochs.

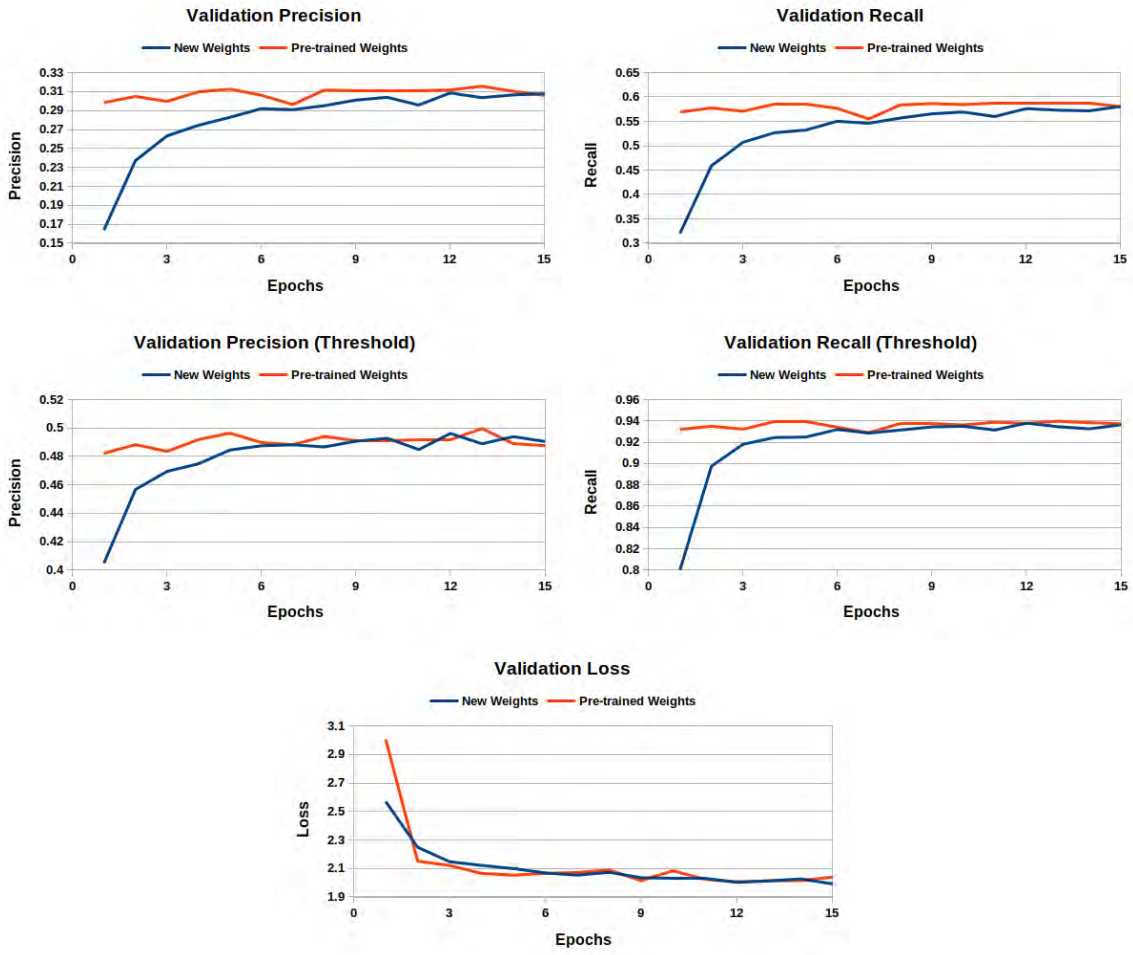

Figure 42: *DetDescEvents* Training: Validation set results over 15 epochs.

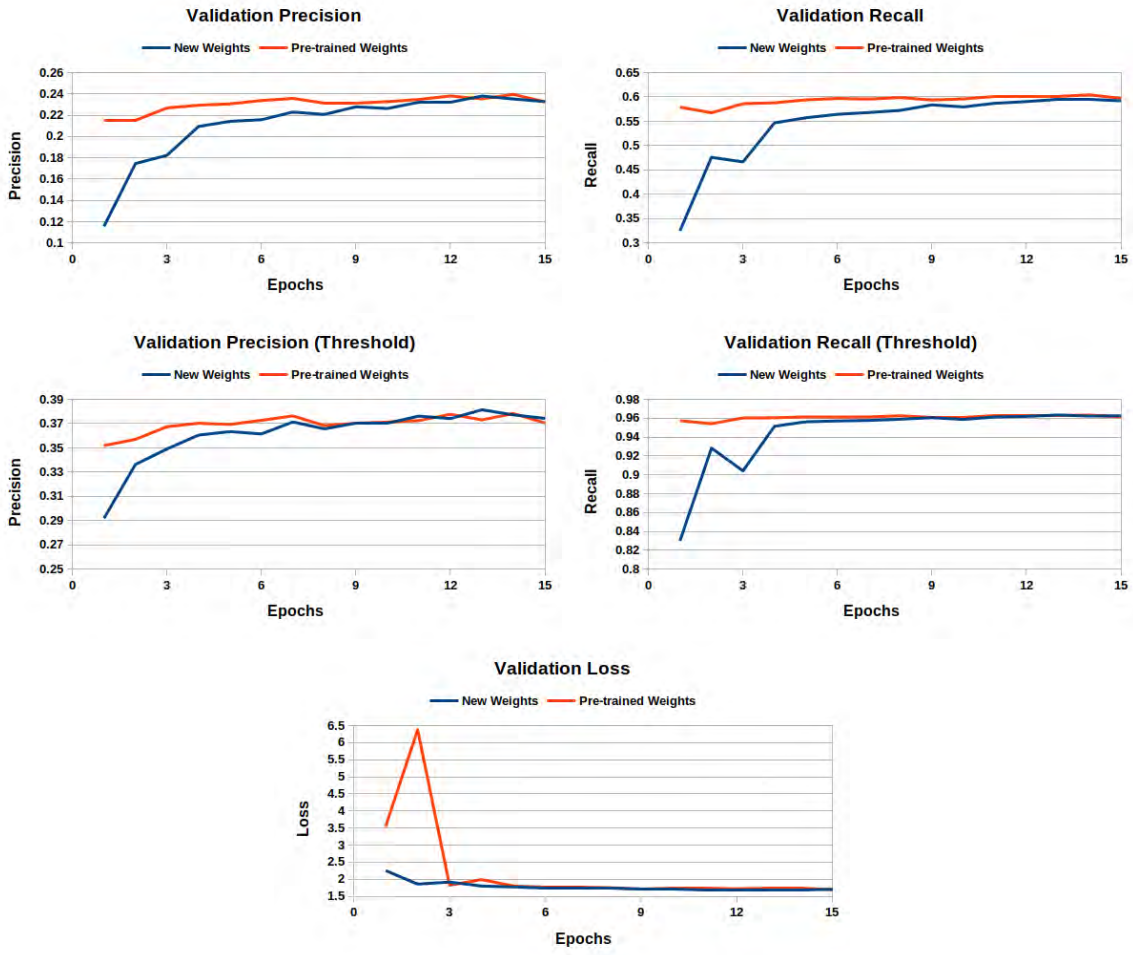

Figure 43: *DetDescEventsLite* Training: Validation set results over 15 epochs.

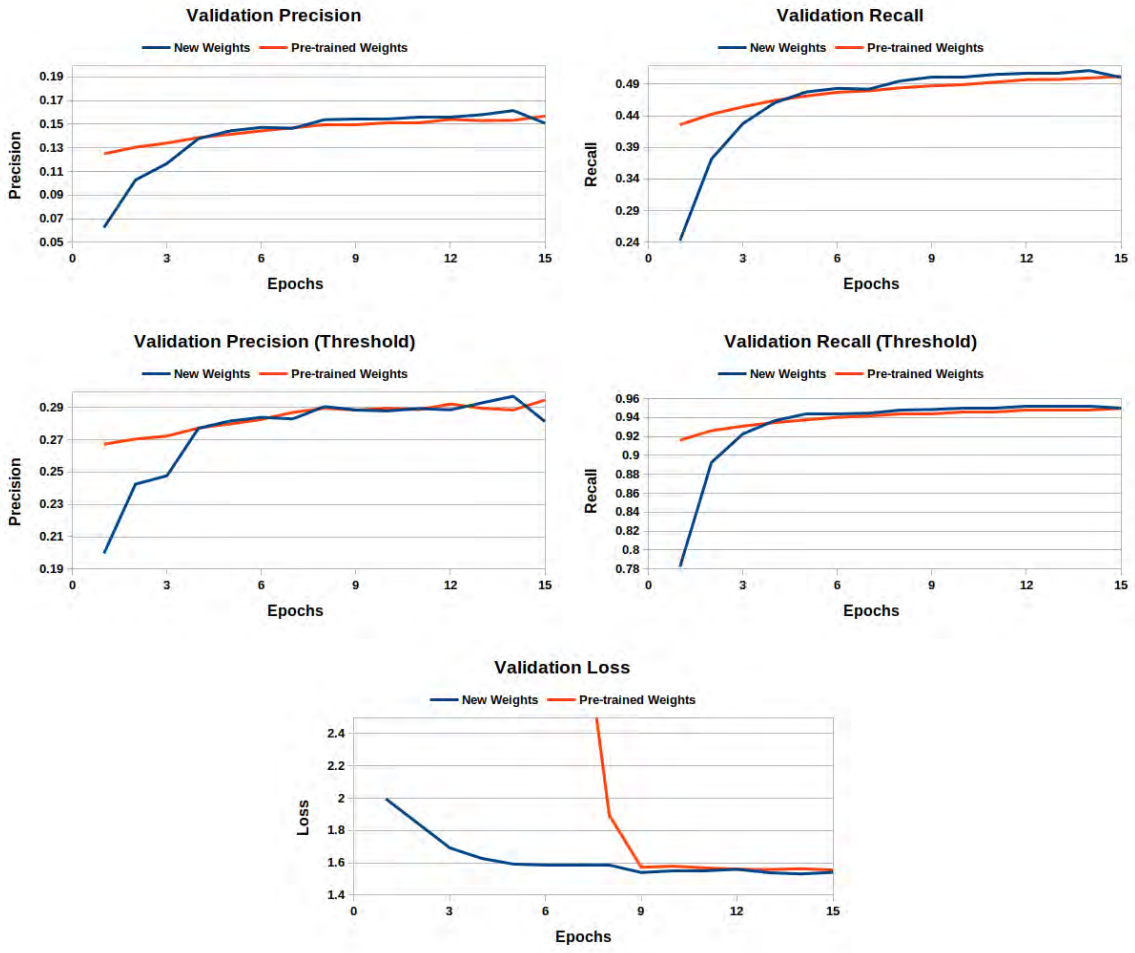

Figure 44: *DetDescEvents (Round 1)* Training: Validation set results over 15 epochs.

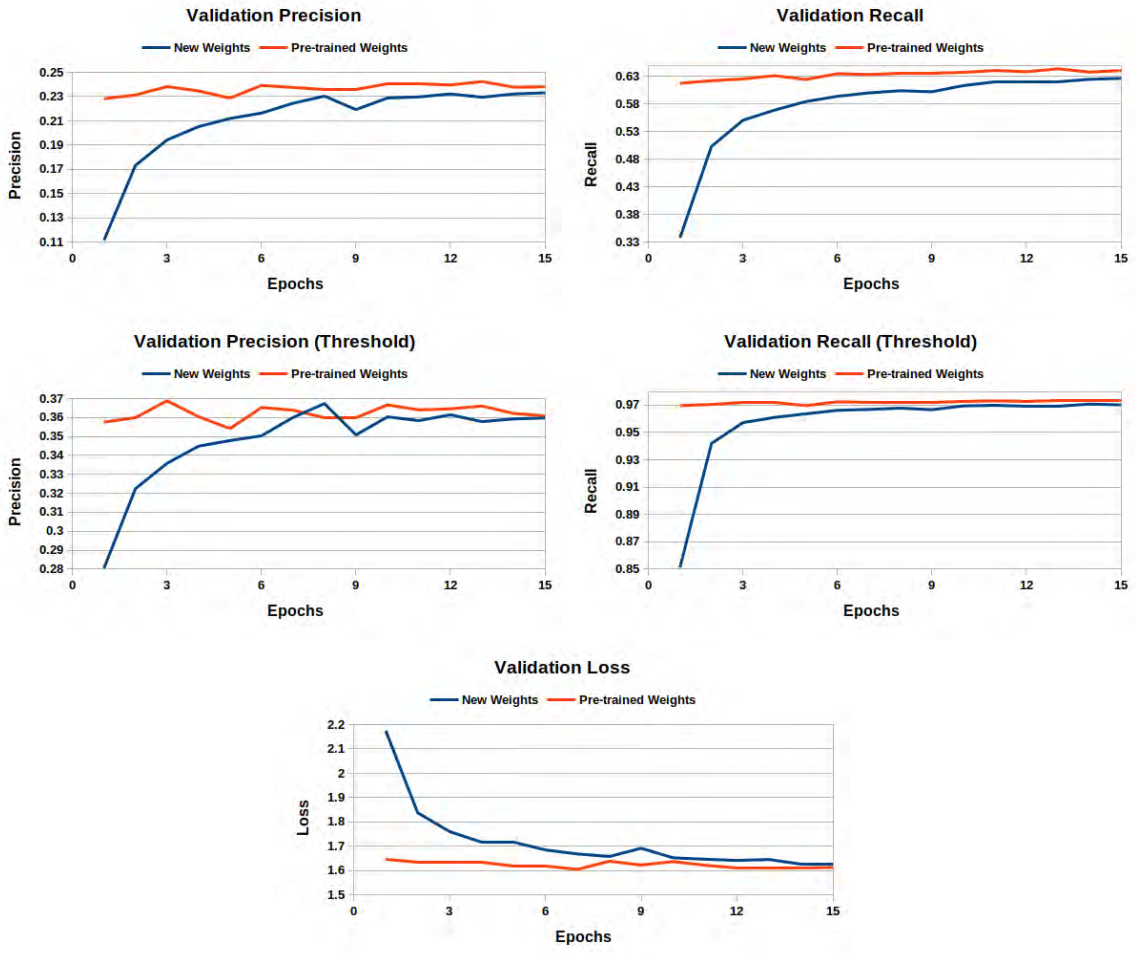

Figure 45: *DetDescEvents (Round 2)* Training: Validation set results over 15 epochs.

Appendix B. Qualitative Results

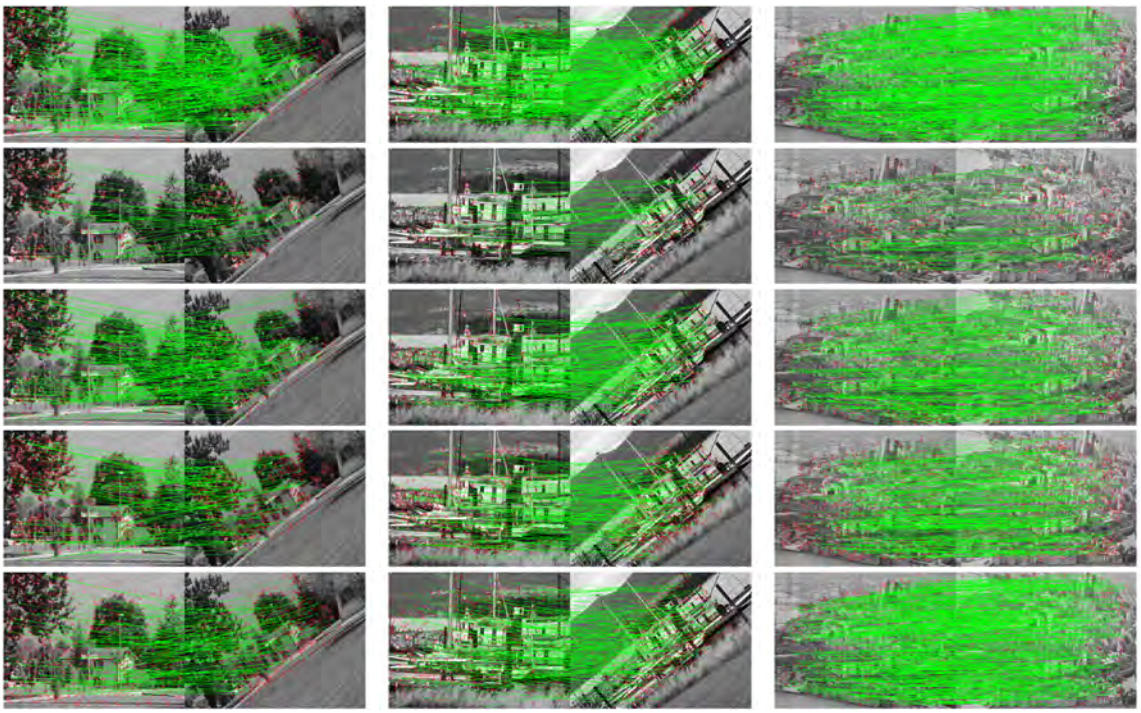

Figure 46: HPatches Viewpoint Qualitative Results: Rows from top to bottom: Pre-trained *DetDesc (Round 2)*, ORB, SIFT, FAST-FREAK, SuperPoint. The green lines show correct feature matches. The CNN model tends to produce more dense and correct matches than the others. On average for HPatches viewpoint scenes, the number of correct matches for pre-trained *DetDesc (Round 2)*, ORB, SIFT, FAST-FREAK and SuperPoint are 285.77, 42.08, 146.65, 132.0 and 238.29, respectively.

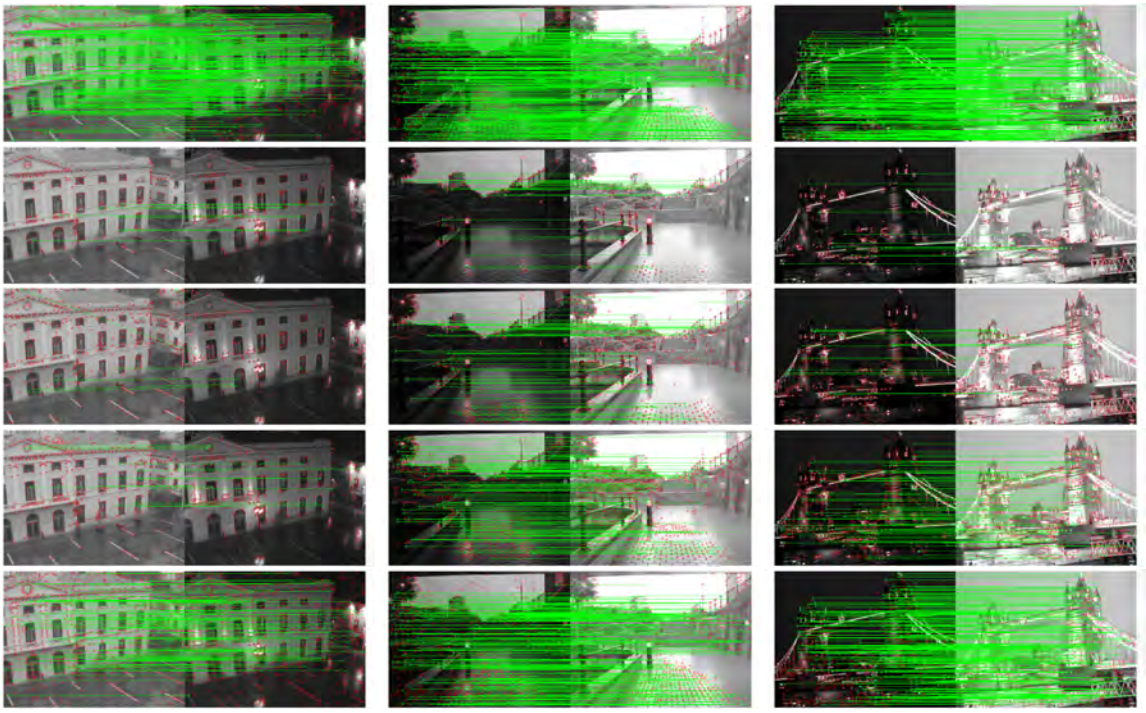

Figure 47: HPatches Illumination Qualitative Results: Rows from top to bottom: Pre-trained *DetDesc (Round 2)*, ORB, SIFT, FAST-FREAK, SuperPoint. The green lines show correct feature matches. The CNN model tends to produce more correct matches than the others. On average for HPatches illumination scenes, the number of correct matches for pre-trained *DetDesc (Round 2)*, ORB, SIFT, FAST-FREAK and SuperPoint are 460.68, 45.01, 115.46, 152.54 and 252.55 respectively.

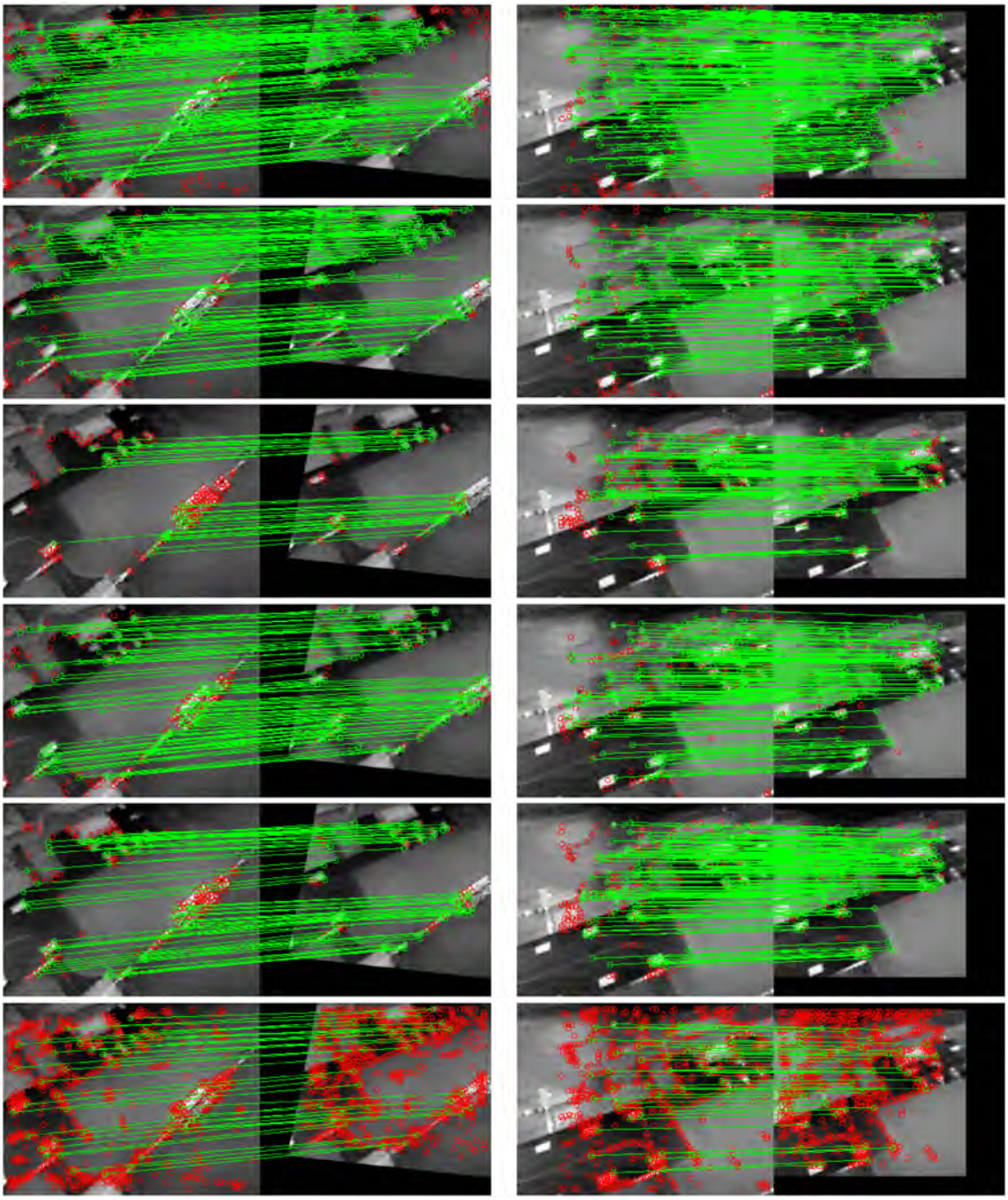

Figure 48: Aerial Event Frame Qualitative Results: Rows from top to bottom: Pre-trained *DetDesc (Round 2)*, pre-trained *DetDescEvents (Round 2)*, ORB, SIFT, FAST-FREAK, SuperPoint. The green lines show correct feature matches. The CNN models tend to produce more correct matches than ORB, SIFT and FAST-FREAK, while pre-trained *DetDesc (Round 2)* produces slightly more correct matches than pre-trained *DetDescEvents (Round 2)*. On average for this dataset, the number of correct matches for pre-trained *DetDesc (Round 2)*, pre-trained *DetDescEvents (Round 2)*, ORB, SIFT, FAST-FREAK and SuperPoint are 157.27, 145.66, 60.61, 103.73, 101.94 and 32.42, respectively.

Appendix C. Gyroscope and Accelerometer Bias Results

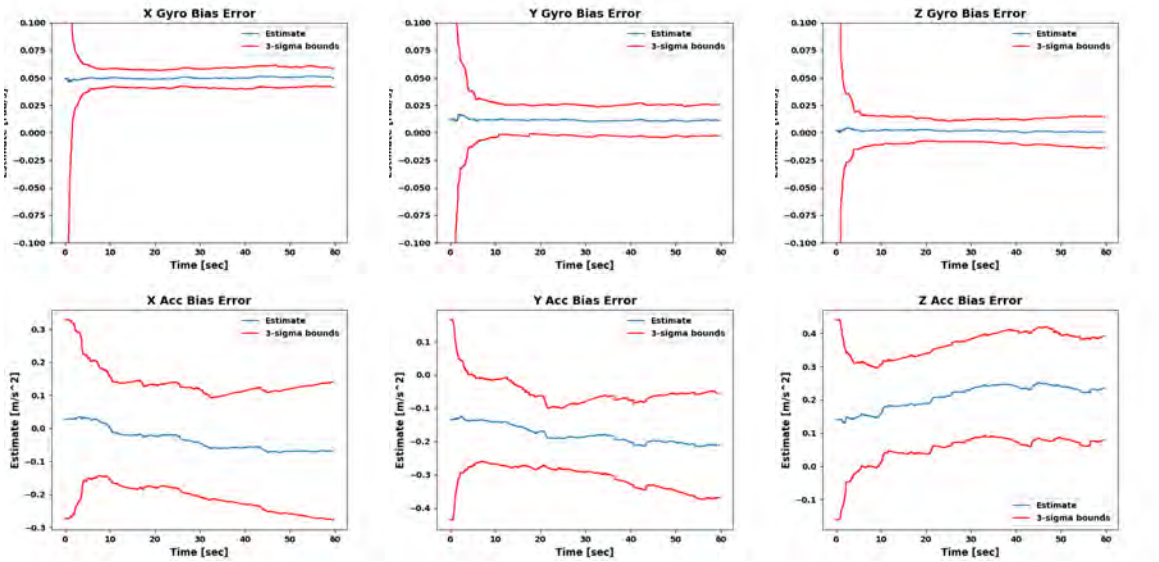

Figure 49: Boxes Dataset, MSCKF Bias Results with Grayscale Images and SIFT: Estimated gyroscope and accelerometer biases. The dataset does not contain truth values.

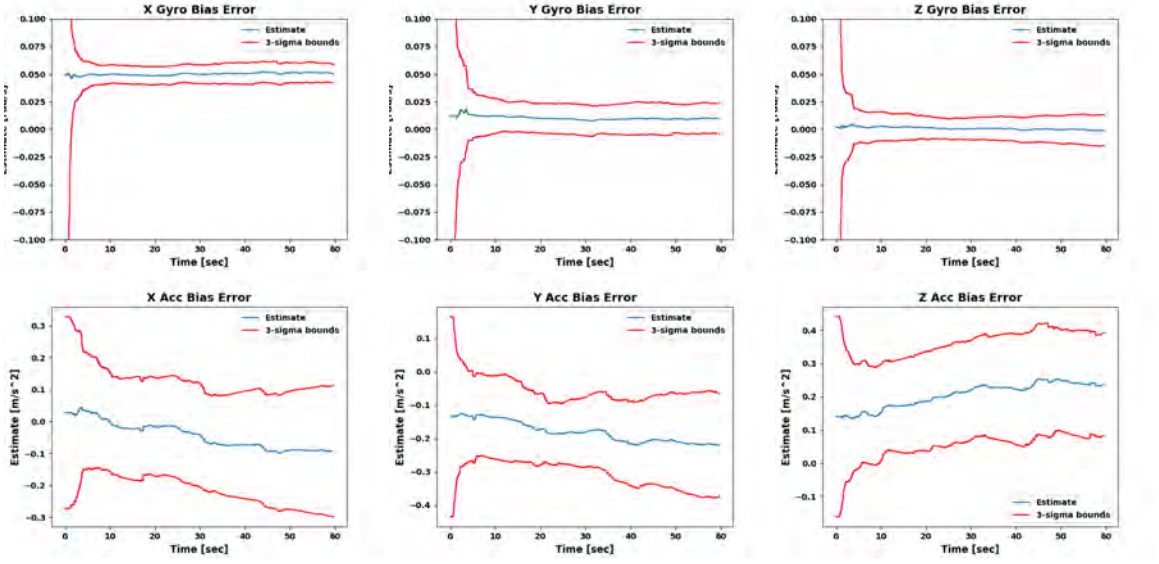

Figure 50: Boxes Dataset, MSCKF Bias Results with Grayscale Images and pre-trained *DetDesc (Round 2)*: Estimated gyroscope and accelerometer biases. The dataset does not contain truth values.

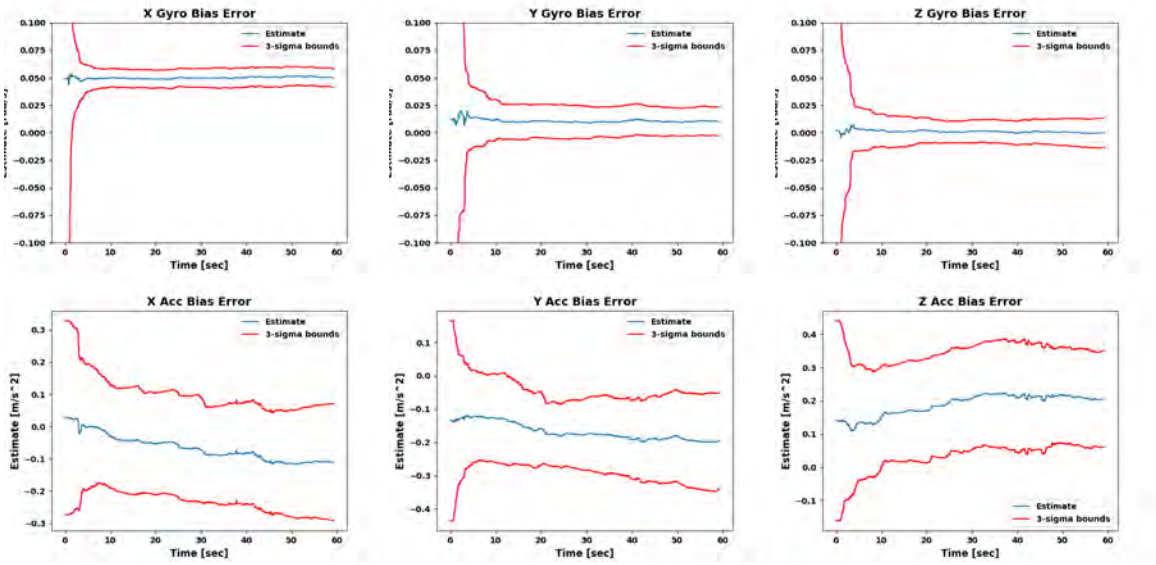

Figure 51: Boxes Dataset, MSCKF Bias Results with Event Frames and SIFT: Estimated gyroscope and accelerometer biases. The dataset does not contain truth values.

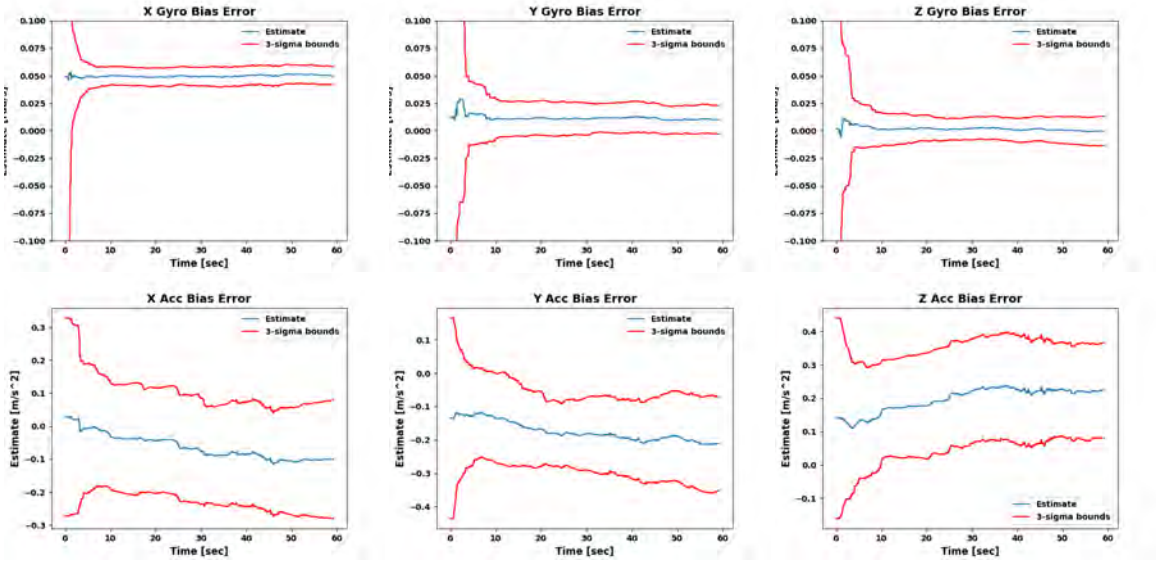

Figure 52: Boxes Dataset, MSCKF Bias Results with Event Frames and pre-trained *DetDesc (Round 2)*: Estimated gyroscope and accelerometer biases. The dataset does not contain truth values.

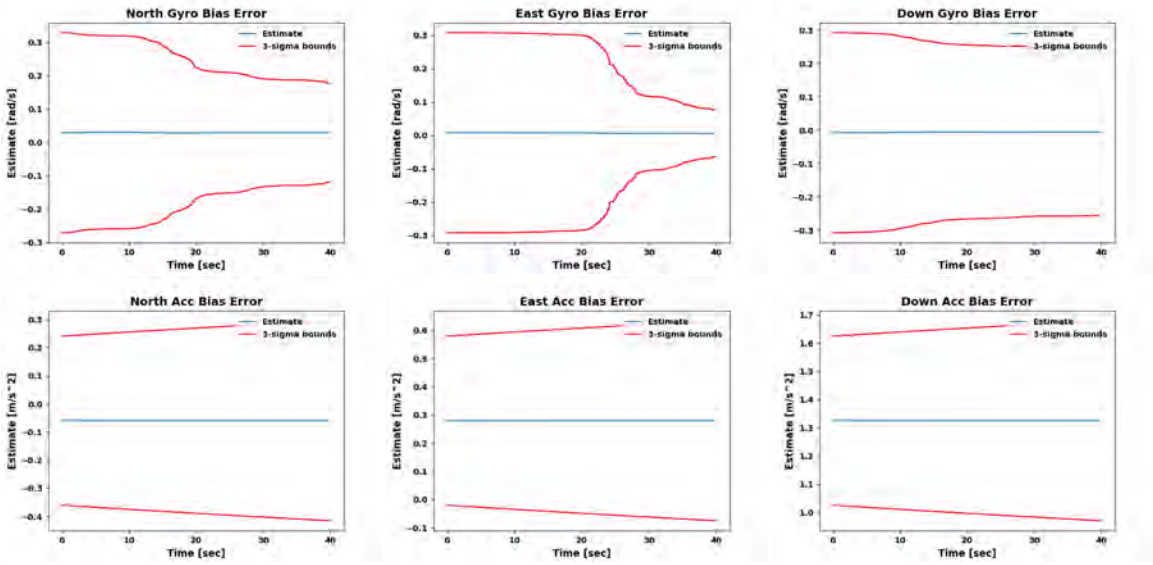

Figure 53: Camp Atterbury Dataset, MSCKF Bias Results with Grayscale Images and SIFT: Estimated gyroscope and accelerometer biases. The dataset does not contain truth values.

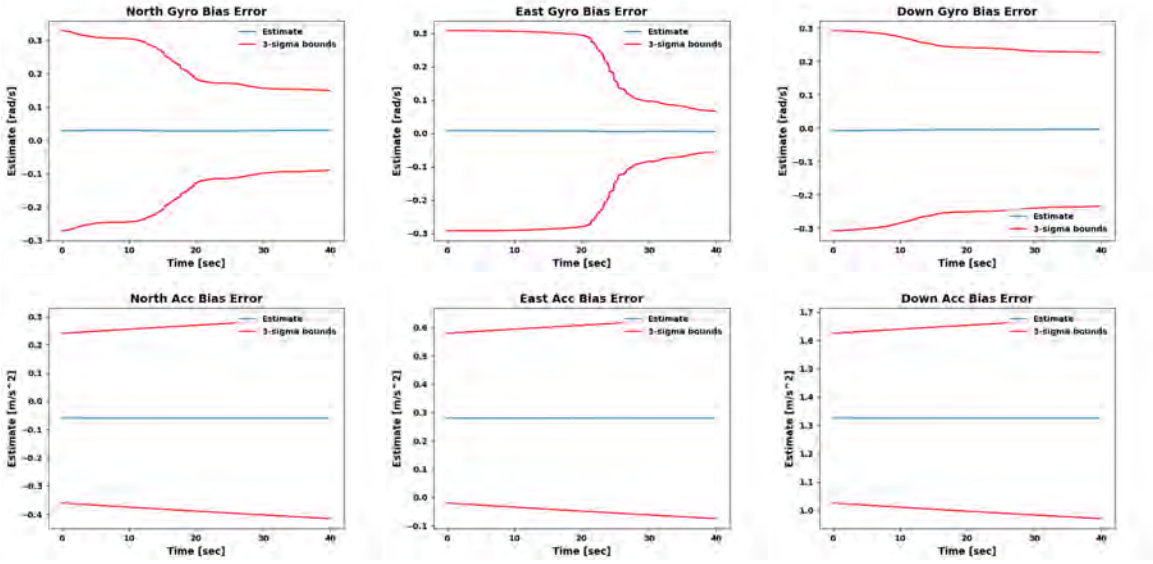

Figure 54: Camp Atterbury Dataset, MSCKF Bias Results with Grayscale Images and pre-trained *DetDesc (Round 2)*: Estimated gyroscope and accelerometer biases. The dataset does not contain truth values.

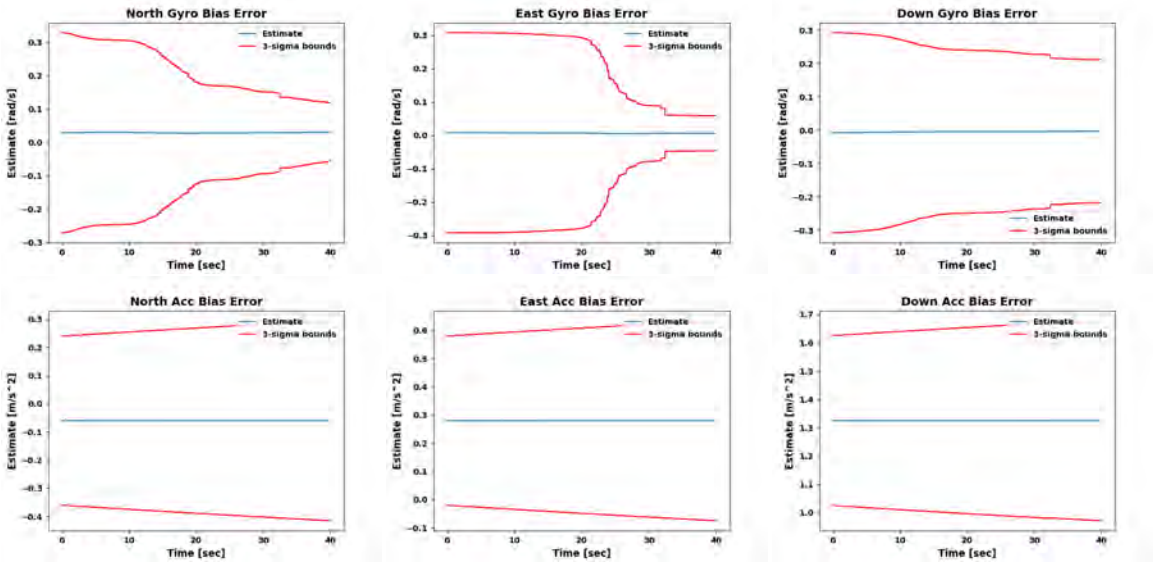

Figure 55: Camp Atterbury Dataset, MSCKF Bias Results with Grayscale Images and pre-trained *DetDescEvents (Round 2)*: Estimated gyroscope and accelerometer biases. The dataset does not contain truth values.

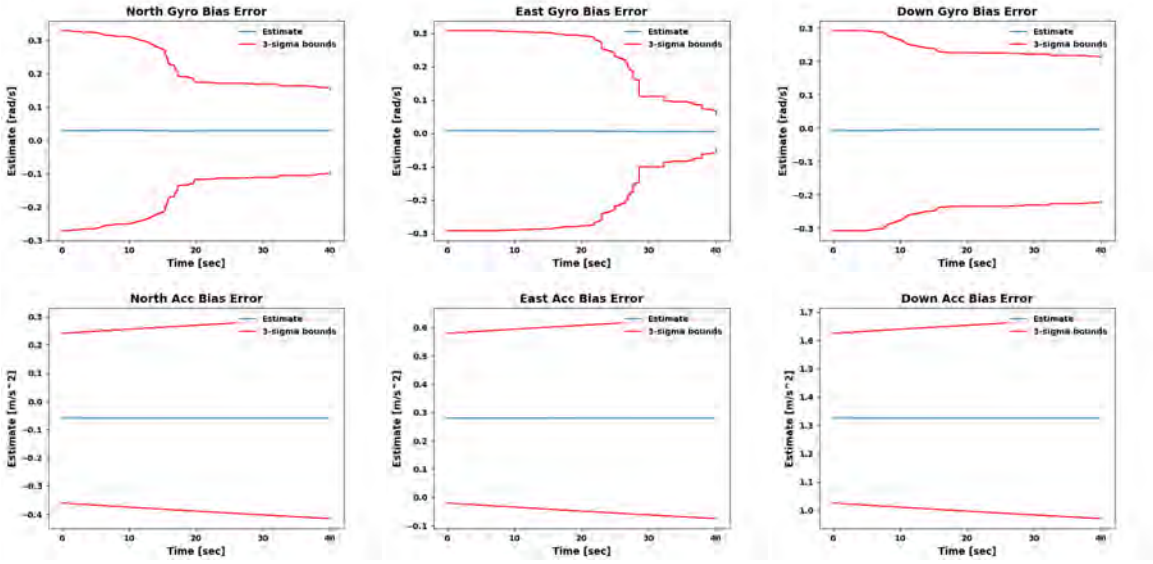

Figure 56: Camp Atterbury Dataset, MSCKF Bias Results with Event Frames and SIFT: Estimated gyroscope and accelerometer biases. The dataset does not contain truth values.

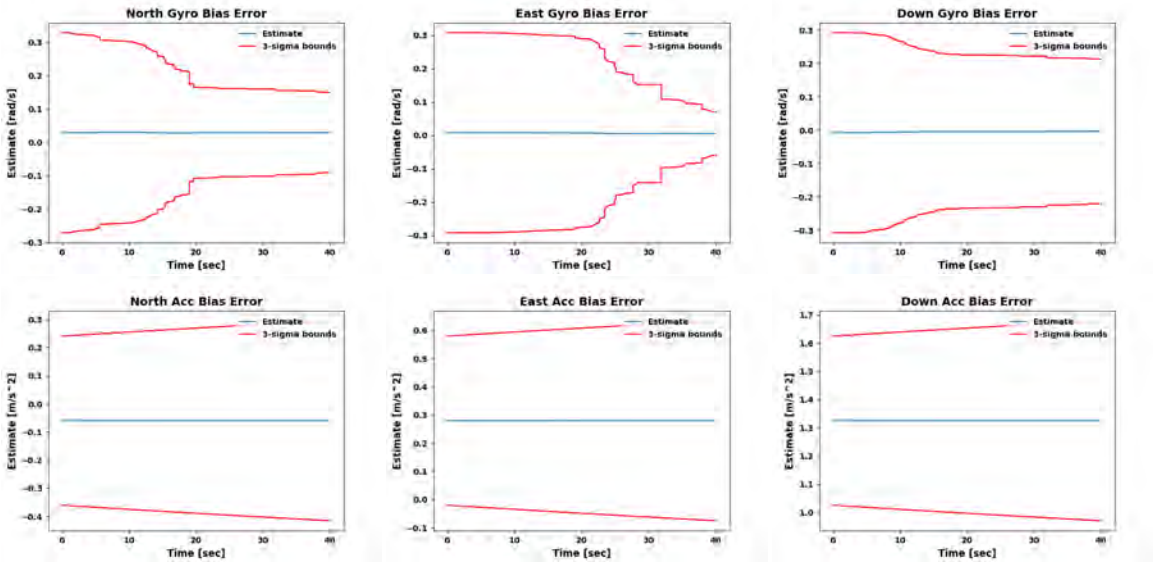

Figure 57: Camp Atterbury Dataset, MSCKF Bias Results with Event Frames and pre-trained *DetDesc (Round 2)*: Estimated gyroscope and accelerometer biases. The dataset does not contain truth values.

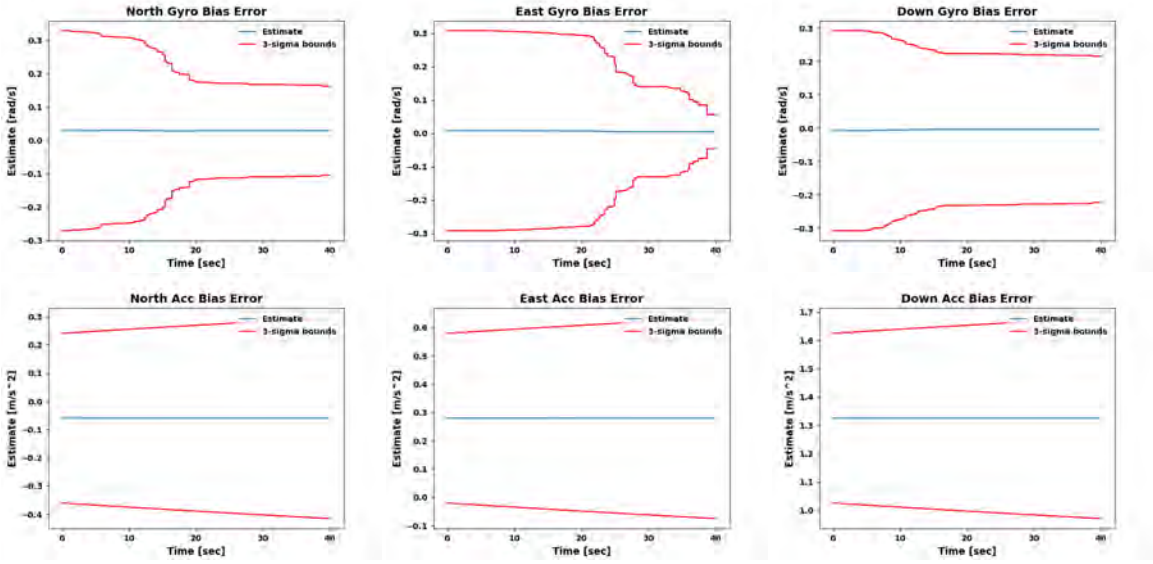

Figure 58: Camp Atterbury Dataset, MSCKF Bias Results with Event Frames and pre-trained *DetDescEvents (Round 2)*: Estimated gyroscope and accelerometer biases. The dataset does not contain truth values.

Bibliography

1. Elias Mueggler, Henri Rebecq, Guillermo Gallego, Tobi Delbrück, and Davide Scaramuzza. The Event-Camera Dataset and Simulator: Event-based data for pose estimation, visual odometry, and SLAM. *Computing Research Repository (CoRR)*, abs/1610.08336, 2016.

2. Davide Scaramuzza and Friedrich Fraundorfer. Visual odometry part I: The first 30 years and fundamentals. *Robotics and Automation Magazine*, pages 80—-92, Dec 2011.

3. Friedrich Fraundorfer and Davide Scaramuzza. Visual odometry part 2: Matching, robustness, optimization, and applications. *Robotics and Automation Magazine*, pages 78—-90, Jun 2012.

4. David G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110, Nov 2004.

5. Herbert Bay, Andreas Ess, Tinne Tuytelaars, and Luc Van Gool. Speeded-up robust features (SURF). *Computer Vision and Image Understanding*, 110(3):346–359, 2008.

6. E. Rublee, V. Rabaud, K. Konolige, and G. Bradski. ORB: An efficient alternative to SIFT or SURF. In *International Conference on Computer Vision*, pages 2564–2571, Nov 2011.

7. Deepak Viswanathan. Features from accelerated segment test (FAST), 2011.

8. C. Harris and M. Stephens. A combined corner and edge detector. *Procedings of the Alvey Vision Conference*, 1988.

9. Michael Calonder, Vincent Lepetit, Christoph Strecha, and Pascal Fua. BRIEF: Binary robust independent elementary features. In *Eur. Conf. Comput. Vis.*, volume 6314, pages 778–792, Sep 2010.

10. Yannick Verdie, Kwang Moo Yi, Pascal Fua, and Vincent Lepetit. TILDE: A temporally invariant learned detector. *Computing Research Repository (CoRR)*, abs/1411.4568, 2014.

11. Daniel DeTone, Tomasz Malisiewicz, and Andrew Rabinovich. SuperPoint: Self-supervised interest point detection and description. *Computing Research Repository (CoRR)*, abs/1712.07629, 2017.

12. Xufeng Han, T. Leung, Y. Jia, R. Sukthankar, and A. C. Berg. MatchNet: Unifying feature and metric learning for patch-based matching. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3279–3286, Jun 2015.

13. Sergey Zagoruyko and Nikos Komodakis. Learning to compare image patches via convolutional neural networks. *Computing Research Repository (CoRR)*, abs/1504.03641, 2015.

14. E. Simo-Serra, E. Trulls, L. Ferraz, I. Kokkinos, P. Fua, and F. Moreno-Noguer. Discriminative learning of deep convolutional feature point descriptors. In *IEEE International Conference on Computer Vision (ICCV)*, pages 118–126, Dec 2015.

15. Kwang Moo Yi, Eduard Trulls, Vincent Lepetit, and Pascal Fua. LIFT: Learned invariant feature transform. *Computing Research Repository (CoRR)*, abs/1603.09114, 2016.

16. Henri Rebecq, René Ranftl, Vladlen Koltun, and Davide Scaramuzza. High speed and high dynamic range video with an event camera. *Computing Research Repository (CoRR)*, abs/1906.07165, 2019.

17. Henri Rebecq, René Ranftl, Vladlen Koltun, and Davide Scaramuzza. Events-to-video: Bringing modern computer vision to event cameras. *IEEE Conf. Comput. Vis. Pattern Recog. (CVPR)*, 2019.

18. A. I. Mourikis and S. I. Roumeliotis. A multi-state constraint Kalman filter for vision-aided inertial navigation. In *IEEE International Conference on Robotics and Automation*, pages 3565–3572, Apr 2007.

19. Ke Sun, Kartik Mohta, Bernd Pfrommer, Michael Watterson, Sikang Liu, Yash Mulgaonkar, Camillo J. Taylor, and Vijay Kumar. Robust stereo visual inertial odometry for fast autonomous flight. *Computing Research Repository (CoRR)*, abs/1712.00036, 2017.

20. D Nister, O Naroditsky, and J Bergen. Visual odometry. *Computer Vision and Pattern Recognition (CVPR)*, 2004.

21. Hans Moravec. Obstacle avoidance and navigation in the real world by a seeing robot rover. Technical Report CMU-RI-TR-80-03, Carnegie Mellon University, Pittsburgh, PA, Sep 1980.

22. Mark Maimone, Yang Cheng, and Larry Matthies. Two years of visual odometry on the mars exploration rovers. *Journal of Field Robotics*, 24(3):169–186, 2007.

23. Richard Hartley and Andrew Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, 2 edition, 2004.

24. Fumio Hamano. Derivative of rotation matrix direct matrix derivation of well known formula. *Computing Research Repository (CoRR)*, abs/1311.6010, 2013.

25. Jianbo Shi and Carlo Tomasi. Good features to track. *IEEE conference on Computer Vision and Pattern Recognition*, 1994.

26. S. Leutenegger, M. Chli, and R. Y. Siegwart. BRISK: Binary robust invariant scalable keypoints. In *International Conference on Computer Vision*, pages 2548–2555, Nov 2011.

27. I. Parra, M. ngel Sotelo, D. F. Llorca, C. Fernndez, A. Llamazares, N. Hernndez, and I. Garca. Visual odometry and map fusion for GPS navigation assistance. In *IEEE International Symposium on Industrial Electronics*, pages 832–837, Jun 2011.

28. Johannes Gräter, Alexander Wilczynski, and Martin Lauer. LIMO: Lidar-monocular visual odometry. *Computing Research Repository (CoRR)*, abs/1807.07524, 2018.

29. Ramon Gonzalez, Francisco Rodriguez, Jose Luis Guzman, Cedric Pradalier, and Roland Siegwart. Combined visual odometry and visual compass for off-road mobile robots localization. *Robotica*, 30(6):865878, 2012.

30. Navid Nourani-Vatani and Paulo Vinicius Koerich Borges. Correlation-based visual odometry for ground vehicles. *Journal of Field Robotics*, 28(5):742–768, 2011.

31. Guillermo Gallego, Tobi Delbrück, Garrick Orchard, Chiara Bartolozzi, Brian Taba, Andrea Censi, Stefan Leutenegger, Andrew J. Davison, Jörg Conradt, Kostas Daniilidis, and Davide Scaramuzza. Event-based vision: A survey. *Computing Research Repository (CoRR)*, abs/1904.08405, 2019.

32. Davide Scaramuzza. Tutorial on event-based cameras. *IROS 2015: Proc. of the 2nd Workshop on Alternative Sensing for Robot Perception*, 2015.

33. Shih-Chii Liu, Tobi Delbruck, Giacomo Indiveri, Adrian Whatley, and Rodney Douglas. *Event-Based Neuromorphic Systems*. John Wiley & Sons, Ltd, 2014.

34. C. Posch. Bio-inspired vision. *Journal of Instrumentation*, 7(1), 2012.

35. E. Mueggler, B. Huber, and D. Scaramuzza. Event-based, 6-DOF pose tracking for high-speed maneuvers. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2761–2768, Sep 2014.

36. G. Gallego, J. E. A. Lund, E. Mueggler, H. Rebecq, T. Delbruck, and D. Scaramuzza. Event-based, 6-DOF camera tracking from photometric depth maps. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 40(10):2402–2412, Oct 2018.

37. B. Kueng, E. Mueggler, G. Gallego, and D. Scaramuzza. Low-latency visual odometry using event-based feature tracks. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 16–23, Oct 2016.

38. P. Lichtsteiner, C. Posch, and T. Delbruck. A 128×128 120 dB 15μs latency asynchronous temporal contrast vision sensor. *IEEE Journal of Solid-State Circuits*, 43(2):566–576, Feb 2008.

39. Hanme Kim, Stefan Leutenegger, and Andrew Davison. Real-time 3D reconstruction and 6-DOF tracking with an event camera. In *Computer Vision - ECCV*, volume 9910, pages 349–364, Sep 2016.

40. Christian Reinbacher, Gottfried Graber, and Thomas Pock. Real-time intensity-image reconstruction for event cameras using manifold regularisation. *Computing Research Repository (CoRR)*, abs/1607.06283, 2016.

41. Hanme Kim, Ankur Handa, Ryad B. Benosman, Sio-Hoi Ieng, and Andrew J. Davison. Simultaneous mosaicing and tracking with an event camera. In *BMVC*, 2014.

42. Henri Rebecq, Guillermo Gallego, Elias Mueggler, and Davide Scaramuzza. EMVS: Event-based multi-view stereo3D reconstruction with an event camera in real-time. *International Journal of Computer Vision*, Nov 2017.

43. P. Bardow, A. J. Davison, and S. Leutenegger. Simultaneous optical flow and intensity estimation from an event camera. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 884–892, Jun 2016.

44. S. Barua, Y. Miyatani, and A. Veeraraghavan. Direct face detection and video reconstruction from event cameras. In *IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 1–9, Mar 2016.

45. C. Posch, D. Matolin, and R. Wohlgenannt. A QVGA 143dB dynamic range asynchronous address-event PWM dynamic image sensor with lossless pixel-level video compression. In *IEEE International Solid-State Circuits Conference - (ISSCC)*, pages 400–401, Feb 2010.

46. T. Delbruck, V. Villanueva, and L. Longinotti. Integration of dynamic vision sensor with inertial measurement unit for electronically stabilized event-based vision. In *IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 2636–2639, Jun 2014.

47. Henri Rebecq, Timo Horstschaefer, and Davide Scaramuzza. Real-time visual-inertial odometry for event cameras using keyframe-based nonlinear optimization. In *BMVC*, 2017.

48. Han-Chao Liu, Fang-Lue Zhang, David Marshall, Luping Shi, and Shi-Min Hu. High-speed video generation with an event camera. *The Visual Computer*, 33, May 2017.

49. H. Rebecq, T. Horstschaefer, G. Gallego, and D. Scaramuzza. EVO: A geometric approach to event-based 6-DOF parallel tracking and mapping in real time. *IEEE Robotics and Automation Letters*, 2(2):593–600, Apr 2017.

50. Elias Mueggler, Guillermo Gallego, and Davide Scaramuzza. Continuous-time trajectory estimation for event-based vision sensors. In *Robotics: Science and Systems*, 2015.

51. A. Censi and D. Scaramuzza. Low-latency event-based visual odometry. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 703–710, May 2014.

52. A. Z. Zhu, N. Atanasov, and K. Daniilidis. Event-based visual inertial odometry. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5816–5824, Jul 2017.

53. Antoni Rosinol Vidal, Henri Rebecq, Timo Horstschaefer, and Davide Scaramuzza. Hybrid, frame and event based visual inertial odometry for robust, autonomous navigation of quadrotors. *Computing Research Repository (CoRR)*, abs/1709.06310, 2017.

54. Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

55. Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. *An introduction to statistical learning: with applications in R*. Springer, 2013.

56. Franois Chollet. *Deep learning with Python*. Manning, Nov 2017.

57. Joseph A. Curro II. Navigation with artificial neural networks (2018). *Theses and Dissertations*, 1948. `https://scholar.afit.edu/etd/1948`.

58. Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *International Conference on Learning Representations*, Dec 2014.

59. Yann Dauphin, Harm Vries, Junyoung Chung, and Y. Bengio. RMSProp and equilibrated adaptive learning rates for non-convex optimization. *arXiv*, 35, Feb 2015.

60. Mihai Dusmanu, Ignacio Rocco, Tomás Pajdla, Marc Pollefeys, Josef Sivic, Akihiko Torii, and Torsten Sattler. D2-Net: A trainable CNN for joint detection and description of local features. *Computing Research Repository (CoRR)*, abs/1905.03561, 2019.

61. Christopher Bongsoo Choy, JunYoung Gwak, Silvio Savarese, and Manmohan Chandraker. Universal correspondence network. *Computing Research Repository (CoRR)*, abs/1606.03558, 2016.

62. Jian Jiao, Jichao Jiao, Yaokai Mo, Weilun Liu, and Zhongliang Deng. MagicVO: End-to-end monocular visual odometry through deep bi-directional recurrent convolutional neural network. *Computing Research Repository (CoRR)*, abs/1811.10964, 2018.

63. Ruihao Li, Sen Wang, Zhiqiang Long, and Dongbing Gu. UnDeepVO: Monocular visual odometry through unsupervised deep learning. *Computing Research Repository (CoRR)*, abs/1709.06841, 2017.

64. Benjamin Ummenhofer, Huizhong Zhou, Jonas Uhrig, Nikolaus Mayer, Eddy Ilg, Alexey Dosovitskiy, and Thomas Brox. DeMoN: Depth and motion network for learning monocular stereo. *Computing Research Repository (CoRR)*, abs/1612.02401, 2016.

65. Sen Wang, Ronald Clark, Hongkai Wen, and Niki Trigoni. End-to-end, sequence-to-sequence probabilistic visual odometry through deep neural networks. *The International Journal of Robotics Research*, 37(4-5):513–542, 2018.

66. Guido Van Rossum and Fred L. Drake. *Python 3 reference manual*. CreateSpace, Scotts Valley, CA, 2009.

67. Martín Abadi et al. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. `tensorflow.org`.

68. François Chollet et al. Keras. `https://keras.io`, 2015.

69. Tsung-Yi Lin, Michael Maire, Serge J. Belongie, Lubomir D. Bourdev, Ross B. Girshick, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. Microsoft COCO: Common objects in context. *Computing Research Repository (CoRR)*, abs/1405.0312, 2014.

70. Vassileios Balntas, Karel Lenc, Andrea Vedaldi, and Krystian Mikolajczyk. HPatches: A benchmark and evaluation of handcrafted and learned local descriptors. *Computing Research Repository (CoRR)*, abs/1704.05939, 2017.

71. G. Bradski. The OpenCV library. *Dr. Dobb's Journal of Software Tools*, 2000.

72. Hannes Sommer, Igor Gilitschenski, Michael Bloesch, Stephan Weiss, Roland Siegwart, and Juan I. Nieto. Why and how to avoid the flipped quaternion multiplication. *Computing Research Repository (CoRR)*, abs/1801.07478, 2018.

73. C. Troiani, A. Martinelli, C. Laugier, and D. Scaramuzza. 2-point-based outlier rejection for camera-IMU systems with applications to micro aerial vehicles. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 5530–5536, May 2014.

74. Shaojie Shen, Yash Mulgaonkar, Nathan Michael, and Vijay Kumar. *Initialization-free monocular visual-inertial state estimation with application to autonomous MAVs*. Springer International Publishing, 2016.

75. Bill Triggs, Philip F. McLauchlan, Richard I. Hartley, and Andrew W. Fitzgibbon. Bundle adjustment - a modern synthesis. In *Workshop on Vision Algorithms*, 1999.

76. J. M. M. Montiel, Javier Civera, and Andrew J. Davison. Unified inverse depth parametrization for monocular SLAM. In *Robotics: Science and Systems*, 2006.

77. Joel A. Hesch, Dimitrios G. Kottas, Sean L. Bowman, and Stergios I. Roumeliotis. Observability-constrained vision-aided inertial navigation. *Multiple Autonomous Robotic Systems Laboratory, TR-2012-001*, 2012.

78. BatchGeo LLC. [online software], 2017. `https://batchgeo.com`.

79. Zichao Zhang and Davide Scaramuzza. A tutorial on quantitative trajectory evaluation for visual(-inertial) odometry. In *IEEE/RSJ Int. Conf. Intell. Robot. Syst. (IROS)*, 2018.

Acronyms

AFIT Air Force Institute of Technology. 1

ANN artificial neural network. 5, 15, 16, 17, 18, 20, 83

ANT Autonomy and Navigation Technology. iv, 1, 3, 52

API application programming interface. 22

BRIEF binary robust independent elementary features. 9

BRISK binary robust invariant scalable keypoints. 9

CMOS complementary metal-oxide semiconductor. 12

CNN convolutional neural network. iv, 2, 3, 18, 19, 22, 23, 25, 26, 28, 31, 33, 34, 42, 54, 83, 84, 85

DAVIS dynamic and active-pixel vision sensor. 22, 52

DCM direct cosine matrix. 8, 37, 39

DOF degrees-of-freedom. iv, 2, 6, 38, 50, 52

DVS dynamic vision sensor. 11

EKF extended Kalman filter. 3, 50

EVIO event-based visual-inertial odometry. iv, 3, 15, 22, 54, 83, 84, 85

FAST features from accelerated segment test. 9

fps frames-per-second. 1, 84, 86

GPS Global Positioning System. 1, 5, 10

HDR high dynamic range. 11, 13

IMU inertial measurement unit. iv, 2, 5, 10, 15, 21, 22, 36, 37, 38, 39, 40, 41, 42, 50, 52, 65, 69, 70, 83

INS inertial navigation system. 15, 83

JPL Jet Propulsion Laboratory. 37

LIDAR light detection and ranging. 10

LIFT learned invariant feature transform. 20

LSE least squares estimate. 44, 47, 50

MLP multilayer perceptron. vii, 16

MSCKF multi-state constraint Kalman filter. iv, 3, 22, 36, 42, 46, 50, 54, 55, 64, 68, 70, 72, 83, 84

NED North-East-Down. 52

OC-EKF observability constrained EKF. 50

ORB oriented FAST and rotated BRIEF. 9

RANSAC random sample consensus. 36, 42

RCNN recurrent-convolutional neural network. 22, 54, 55

ReLU rectified linear unit. 16, 26

RMSE root-mean-square error. 69, 70, 72

RPG Robotics and Perception Group. 3, 22, 52, 55

SE(3) Special Euclidean. 9

SGD stochastic gradient descent. 16

SIFT scale invariant feature transform. 8, 9, 42, 60, 61, 68, 69, 70, 83, 84

SURF speeded up robust features. 9

SVD singular value decomposition. 6, 45

UAV unmanned aerial vehicle. iv, vii, 3, 22, 23, 24, 25, 52, 55, 56, 65, 70, 83

UCN universal correspondence network. 20

VIO visual-inertial odometry. 50

VO visual odometry. iv, 1, 2, 3, 5, 8, 10, 11, 14, 20, 21, 83

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704–0188

1. REPORT DATE *(DD–MM–YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From — To)*
26–03–2020	Master's Thesis	Sept 2018 — Mar 2020

4. TITLE AND SUBTITLE

Event-Based Visual-Inertial Odometry
Using Smart Features

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Friedel, Zachary P, 1st Lt, USAF

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Air Force Institute of Technology
Graduate School of Engineering and Management (AFIT/EN)
2950 Hobson Way
WPAFB OH 45433-7765

8. PERFORMING ORGANIZATION REPORT NUMBER

AFIT-ENG-MS-20-M-021

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Intentionally Left Blank

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT

DISTRIBUTION STATEMENT A:
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

Event-based cameras are a novel type of visual sensor that operate under a unique paradigm, providing asynchronous data on the log-level changes in light intensity for individual pixels. This hardware-level approach to change detection allows these cameras to achieve ultra-wide dynamic range and high temporal resolution. Furthermore, the advent of convolution neural networks (CNNs) has led to state-of-the-art navigation solutions that now rival or even surpass human engineered algorithms. The advantages offered by event cameras and CNNs make them excellent tools for visual odometry. A visual odometry pipeline was implemented with a front-end network for generating event-frames that were fed into a multi-state constraint Kalman filter (MSCKF) back-end, which utilized features and descriptors generated by a CNN. This pipeline was tested on a public dataset and data collected from an ANT Center UAV flight test.

15. SUBJECT TERMS

artificial neural network (ANN), convolutional neural network (CNN), deep learning, dynamic vision sensor (DVS), event-based cameras, event-based visual-inertial odometry (EVIO), Extended Kalman Filter (EKF), machine learning, multi-state constraint Kalman filter (MSCKF), neuromorphic engineering, visual odometry (VO)

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Dr. Robert C. Leishman, AFIT/ENG
U	U	U	UU	133	19b. TELEPHONE NUMBER *(include area code)* (937) 255-3636 x4755; robert.leishman@afit.edu