Markerless 3D Motion Capture of Cheetahs in the Wild

University of Cape Town
Dept. of Electrical and Electronics Engineering

Presented by:
Liam James Clark

Prepared for:
Dr. Amir Patel

University of Cape Town

Submitted to the Department of Electrical and Electronics Engineering at the University of Cape Town in partial fulfilment of the academic requirements for a Master of Science degree in Engineering.

July 7, 2020

The copyright of this thesis vests in the author. No quotation from it or information derived from it is to be published without full acknowledgement of the source. The thesis is to be used for private study or non-commercial research purposes only.

Published by the University of Cape Town (UCT) in terms of the non-exclusive license granted to UCT by the author.

University of Cape Town

Acknowledgments

A large amount of time and effort has gone into making this project a success, and without the help and advice of others, this would not have been possible.

Firstly, I would like to thank my supervisor, Dr. Amir Patel, for the many hours put into helping me with this project. His enthusiasm towards everything technological and cheetah related as well as his guidance throughout the project and my studies at UCT have been invaluable.

I would like to thank Prof. Fred Nicolls for being welcoming and always making an effort to help and share his expertise, especially towards the computer vision and optimisation aspects of the project.

I would also like to thank Sarah Lawson for all the support during the project and taking the time and effort to read through the various iterations of my report; I really appreciate it.

Thank you to my friends in the Mechatronics Lab who have made my time in the lab so enjoyable. I will certainly miss the many interesting chats around the coffee machine and the various side projects.

Finally, to my family, thank you for making it possible for me to study and complete my MSc. I am incredibly grateful for the support which you have shown towards any interests of mine.

Declaration

I know the meaning of plagiarism and declare that all the work in the document, save for that which is properly acknowledged, is my own. This thesis/dissertation has been submitted to the Turnitin module and I confirm that my supervisor has seen my report and any concerns revealed by such have been resolved with my supervisor.

Signature: Signed by candidate
Liam James Clark

Date: January 7, 2021

Contents

1 Introduction **1**
- 1.1 Background . . . 1
- 1.2 Problem Statement . . . 3
- 1.3 Project Objectives . . . 5
- 1.4 Project Scope . . . 6
- 1.5 Project Outline . . . 6
- 1.6 Project Outcomes . . . 7
  - 1.6.1 Neural Network . . . 7
  - 1.6.2 Cameras . . . 7
  - 1.6.3 3D Reconstruction . . . 8

2 Literature Review **9**
- 2.1 3D Vision Based Motion Capture . . . 9
- 2.2 Convolutional Neural Networks and Pose Estimation . . . 11
- 2.3 Cameras . . . 14
  - 2.3.1 Pinhole Cameras . . . 14
  - 2.3.2 The Projective Camera . . . 14
  - 2.3.3 Lens Distortion . . . 19
  - 2.3.4 Calibration Methods . . . 21
  - 2.3.5 Direct Linear Transformation . . . 21
- 2.4 3D Triangulation . . . 25
- 2.5 Sparse Bundle Adjustment . . . 26
  - 2.5.1 Jacobian Sparsity . . . 27
  - 2.5.2 Robust Cost Functions . . . 28
- 2.6 The Kalman Filter . . . 30
  - 2.6.1 Process Model . . . 32
  - 2.6.2 Prediction and Update Steps . . . 33
  - 2.6.3 The Extended Kalman Filter . . . 34
  - 2.6.4 Kalman Smoothing . . . 35
- 2.7 Optimisation-based Estimation . . . 37

2.8 Summary . . . 38

3 Methodology **39**
3.1 Project Plan . . . 39
3.1.1 Problem Definition . . . 39
3.1.2 Project Requirements . . . 39
3.2 Design Approach . . . 40
3.3 Data . . . 44
3.3.1 Cheetah Videos . . . 44
3.3.2 DeepLabCut . . . 48
3.3.3 Simulated Data . . . 50
3.3.4 Cheetah Pose Model . . . 53
3.4 Calibration . . . 57
3.4.1 Intrinsic Parameters . . . 57
3.4.2 Extrinsic Parameters . . . 60

4 Sparse Bundle Adjustment **64**
4.1 Overview . . . 64
4.2 Design . . . 64
4.3 Simulated Data Results . . . 66
4.3.1 Statistics . . . 67
4.4 Real Data Results . . . 71
4.4.1 Cheetah Run . . . 73
4.4.2 Cheetah Dive . . . 74
4.5 Discussion . . . 75

5 Extended Kalman Filter **77**
5.1 Overview . . . 77
5.2 Design . . . 77
5.2.1 State Variables . . . 77
5.2.2 Process Model . . . 78
5.2.3 Process Covariance Matrix . . . 78
5.2.4 Measurement Function . . . 80
5.2.5 Measurement Covariance . . . 81
5.2.6 Outlier Rejection . . . 81
5.2.7 Prediction and Update Algorithm . . . 82
5.2.8 Initialisation of the EKF . . . 82
5.2.9 Kalman Smoother . . . 83
5.3 Simulated Data Results . . . 84
5.4 Real Results . . . 90
5.4.1 Cheetah Run . . . 91

5.4.2 Cheetah Dive 95
5.5 Discussion 98

6 Full Trajectory Optimisation **100**
6.1 Design 100
6.1.1 Parameters and Variables 100
6.1.2 Structuring the Optimisation Problem 101
6.1.3 Implementation 106
6.2 Simulated Data Results 107
6.3 Real Results 110
6.3.1 Cheetah Run 111
6.3.2 Cheetah Dive 114
6.4 Discussion 116

7 Conclusion **118**
7.1 Discussion 118
7.2 Future Work 120

List of Figures

1.1 A picture of the MIT Cheetah 3 [4] 2
1.2 A picture of Phantom, the cheetah, during a run at the Ann van Dyk Cheetah Centre in March, 2019 4
1.3 Dust occlusion during a cheetah run at the Ann van Dyk Cheetah Centre 5
1.4 Project outline 7

2.1 Through wall human pose estimation using radar [13] 10
2.2 Screenshot of Vicon's Nexus software used for human motion capture [16] 10
2.3 Overview of the process of using DeepLabCut for 3D reconstruction using triangulation [8] 12
2.4 Diagram of a pin-hole projection [33] 14
2.5 Diagram of a pin-hole camera projection model from the OpenCV software library [34] 15
2.6 Diagram of 3D point P being projected onto an image plane . 17
2.7 Top down view of Figure 2.6 17
2.8 A diagram showing two types of radial distortion applied to a chessboard pattern from the OpenCV library documentation [34] 19
2.9 Diagram of the epipolar geometry for point-line correspondence between two views [36] 24
2.10 Diagram of the minimisation problem for triangulation [36] . . 26
2.11 Diagram showing the Jacobian sparsity pattern for a SBA with three cameras and 4 3D points [36] 28
2.12 A comparative plot of cost functions 30
2.13 Flowchart of the Kalman filter algorithm 31
2.14 Plot of a single state Kalman filter estimating the mean of the state (blue line) and state covariance (yellow band) given a set of measurements of the state 32

2.15 Plot showing the state prediction and update steps where the measurement is defined in the space of the states, adapted from [48] . . . 33
2.16 Diagram showing the difference between prediction, filtering and smoothing . . . 36
2.17 Depiction of moving horizon estimation [55] . . . 37

3.1 A diagram showing the system components and flow of data through the system . . . 42
3.2 A diagram showing the WBS for camera calibration tasks . . . 43
3.3 A diagram showing the WBS for 3D reconstruction tasks . . . 44
3.4 Camera layout for recording the cheetahs at the cheetah sanctuary . . . 45
3.5 Overlaid frames from a video of a cheetah run . . . 45
3.6 Overlaid frames from a video of a cheetah dive . . . 45
3.7 A cheetah viewed from all 6 cameras . . . 46
3.8 DeepLabCut markers with likelihood ≥ 0.5 plotted over the frame they were obtained from . . . 48
3.9 Histogram of the cheetah data training error from DLC . . . 49
3.10 Histogram of the cheetah data test error from DLC . . . 50
3.11 Screenshot of a python app for viewing pose data showing select frames from a simulated cheetah run including 3D positions and the 2D camera projections . . . 51
3.12 Plot showing the effect of 100 measurements of the point $(0, 0)$ with varying degrees of noise (in orange) and outliers (in blue) 53
3.13 Cheetah model with markers corresponding to Table 3.7 . . . 54
3.14 A screenshot taken while extracting calibration frames from a GoPro Hero 7 using the frame extraction tool . . . 58
3.15 A plot of a set of checkerboard locations used for calibration . 58
3.16 The undistorted plot of Figure 3.15 . . . 60
3.17 Perspective view of the 3D calibration board points from 2D pairwise correspondences (1-2 red, 2-3 green, 3-4 yellow, 4-5 dark blue, 5-6 light blue) where camera 1 is in the bottom left of the image . . . 61
3.18 Top-down view of the 3D calibration board points from 2D pairwise correspondences (1-2 red, 2-3 green, 3-4 yellow, 4-5 dark blue, 5-6 light blue) where camera 1 is in the bottom left of the image . . . 62
3.19 Histogram of the calibration reprojeciton errors before and after SBA . . . 63

4.1 Cauchy cost function for the measurement errors 65
4.2 Histogram of the 3D position errors 69
4.3 Plot of the ground truth 3D point cloud of the simulated cheetah trajectory 70
4.4 Plot of the initial triangulated 3D point cloud of the simulated cheetah trajectory for test 5 70
4.5 Plot of the estimated 3D point cloud of the simulated cheetah trajectory for test 5 after SBA 71
4.6 Plot of the 3D point cloud of the cheetah trajectory from pairwise triangulation 72
4.7 Plot of the 3D point cloud of the cheetah trajectory after SBA 72
4.8 Plot of the SBA trajectory estimation for the cheetah run . . . 73
4.9 Errors in the SBA trajectory estimation for the cheetah run . 73
4.10 Plot of the SBA trajectory estimation for the cheetah dive . . 74
4.11 Errors in the SBA trajectory estimation for the cheetah dive . 74
4.12 Histogram of the 2D reprojection errors for test 5 75

5.1 Histogram of the 3D position errors 86
5.2 Comparison of the states from the EKF (before the Kalman smoother was run) and the states from the simulation for test 5 denoted by the *sim* subscript in the legend 87
5.3 Comparison of the states from the EKF (before the Kalman smoother was run) and the states from the simulation for test 5 denoted by the *sim* subscript in the legend 88
5.4 Comparison of the states from the EKF (after the Kalman smoother was run) and the states from the simulation for test 5 denoted by the *sim* subscript in the legend 89
5.5 Comparison of the states from the EKF (after the Kalman smoother was run) and the states from the simulation for test 5 denoted by the *sim* subscript in the legend 90
5.6 Plot of the EKF trajectory estimation for the cheetah run . . 91
5.7 Errors in the EKF trajectory estimation for the cheetah run . 91
5.8 Plot of the Kalman smoother trajectory estimation for the cheetah run 92
5.9 Errors in the Kalman smoother trajectory estimation for the cheetah run 92
5.10 States estimated from a real cheetah run using the EKF and Kalman smoother denoted by the *smooth* subscript in the legend 93
5.11 States estimated from a real cheetah run using the EKF and Kalman smoother denoted by the *smooth* subscript in the legend 94
5.12 Plot of the EKF trajectory estimation for the cheetah dive . . 95

5.13 Errors in the EKF trajectory estimation for the cheetah dive . 95
5.14 Plot of the Kalman smoother trajectory estimation for the cheetah dive 96
5.15 Errors in the Kalman smoother trajectory estimation for the cheetah dive 96
5.16 States estimated from a real cheetah run using the EKF and Kalman smoother denoted by the *smooth* subscript in the legend 97
5.17 States estimated from a real cheetah run using the EKF and Kalman smoother denoted by the *smooth* subscript in the legend 98

6.1 Redescending cost function for the normalised measurement errors 105
6.2 Histogram of the 3D position errors 108
6.3 Comparison of the states from the full trajectory estimation and the states from the simulation for test 5 109
6.4 Comparison of the states from the full trajectory estimation and the states from the simulation for test 5 110
6.5 Plot of the full optimisation trajectory estimation for the cheetah run 111
6.6 States estimated from a real cheetah run from the simultaneous trajectory estimation 112
6.7 States estimated from a real cheetah run from the simultaneous trajectory estimation 113
6.8 Plot of the full optimisation trajectory estimation for the cheetah dive 114
6.9 States estimated from a real cheetah run from the simultaneous trajectory estimation 115
6.10 States estimated from a real cheetah run from the simultaneous trajectory estimation 116

7.1 Visual comparison of the different trajectory estimation methods for the cheetah dive 120

List of Tables

3.1 A table showing the joint information obtained from DeepLabCut . . . 47
3.2 A example showing the type of data obtained from DeepLabCut 48
3.3 DeepLabCut training and test error statistics . . . 49
3.4 Noise and outlier parameters for the simulated datasets . . . 52
3.5 Pose parameters for the cheetah model . . . 54
3.6 Rotation matrices for the kinematic model of the cheetah . . . 55
3.7 Positions for the kinematic model of the cheetah . . . 57
3.8 GoPro Hero 7 settings used for filming of the cheetahs and calibration videos . . . 59
3.9 Mean and standard deviation of the reprojection errors before and after performing a SBA . . . 63

4.1 2D reprojection error's mean μ, standard deviation σ, median and median absolute deviation (MAD) before and after SBA . 67
4.2 3D position error's mean μ, standard deviation σ, median and median absolute deviation (MAD) before and after SBA . . . 68

5.1 Variance values used for the acceleration states in the process covariance matrix . . . 80
5.2 Modified EKF algorithm . . . 82
5.3 Variance values used for the acceleration states in the process covariance matrix . . . 83
5.4 3D position error's mean μ, standard deviation σ, median and median absolute deviation (MAD) from the EKF and Kalman Smoother . . . 85

6.1 Parameters used for the trajectory estimation . . . 101
6.2 Variables to be determined as part of the trajectory estimation 101
6.3 Bound constraints for the cheetah pose parameters . . . 103

6.4 3D position error's mean μ, standard deviation σ, median and median absolute deviation (MAD) from the trajectory optimisation 107

Chapter 1

Introduction

1.1 Background

The field of robotics is developing at a rapid pace. However, state-of-the-art legged robots are a far way off from navigating and interacting with the world in a manner which is as free as their animal and human counterparts.

Currently, in legged robotics, researchers often prefer to study humans and dogs due to the difficulty associated with studying other legged animals. The cheetah (*Acinonyx jubatus*) is a particularly compelling animal to study due to its various cursorial[1] adaptions. The cheetah is regarded as the fastest land animal [1] having been recorded running at speeds of up to $29m/s$. Cheetahs are also highly manoeuvrable, often exceeding lateral accelerations of $13m/s^2$ when hunting [2].

The MIT Cheetah, a quadruped robot, draws part of its mechanical design from the structure of animals, mimicking adaptations such as the locations of muscle within their leg structures. By positioning actuators close to the shoulders of the robot and using actuated links which mimicked tendons, Seok et al. [3] were able to reduce the inertial forces experienced by the legs of the robot under high accelerations. This reduced the stress on the legs while running.

[1] Cursorial animals are animals which are specifically adapted to running.

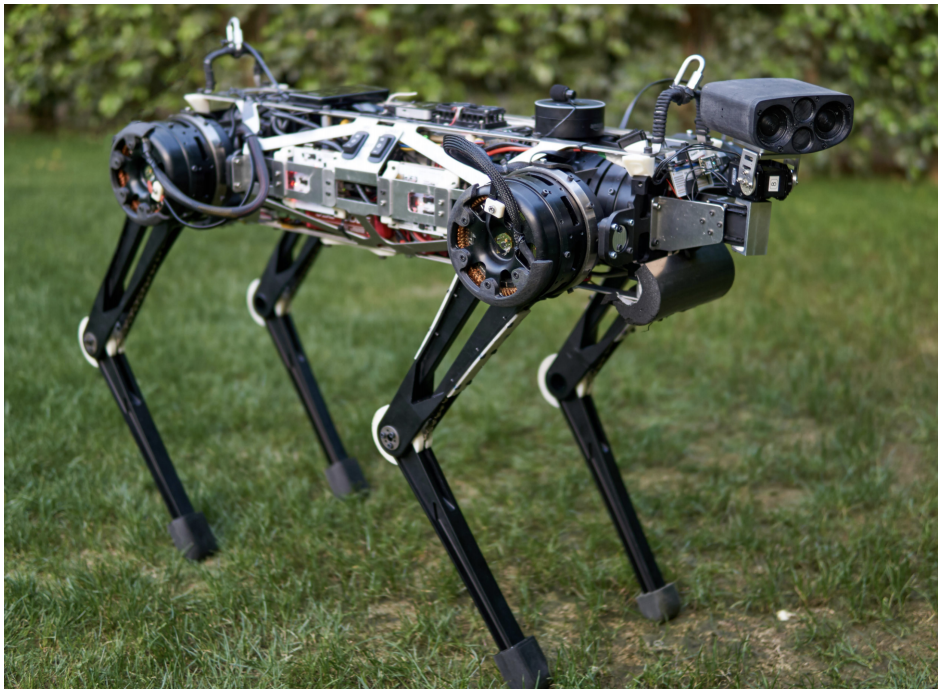

Figure 1.1: A picture of the MIT Cheetah 3 [4]

There is reason to believe that by studying the locomotion of animals like the cheetah, we will find other ways to improve upon the efficiency, agility, strength and speed of legged robots. One area which is not widely understood is why and how cheetahs use their tails when manoeuvring. There have been few studies looking at tails as a means to help with manoeuvrability in robotics even though studies have shown that the addition of an actuated tail can increase their acceleration capabilities [5] and improve stability [6].

In order to enable the biomimetic[2] design of robots, it is important to create tools to gather data from the behaviour of animals. Recently, due to the advancements in deep learning and convolutional neural networks, several software-based tools [7], [8], [9], [10] have become available, making it possible to study human and animal locomotion from video footage. These tools are mostly to study animal behaviour from a neuro-scientific perspective in a lab; however, this data could be beneficial in other areas, such as designing controllers for legged robots.

It is crucial to study wild animals in a natural environment and to avoid attaching equipment to them as this can cause the animals stress. During

[2]Biomimetic - related to methods which are derived from biology.

locomotion studies, this also provides data which is subjectively more natural, as equipment and strange environments can cause changes to the behaviour of the animal. Attempts to study the locomotion of cheetahs by attaching sensors and cameras to them using a harness [11] and by using a multi-camera setup to film them from multiple angles [8] have provided promising results; however, neither of these methods has allowed for robust, whole-body motion estimation.

1.2 Problem Statement

This project aims to develop and test a set of tools and methods for 3D motion capture of cheetahs, specifically aimed at tracking locations on the body of the animal. Cheetahs were chosen as the subjects for this study due to their incredible speed and agility. In order to study the natural locomotion of the cheetahs, it was essential to develop a non-intrusive system. The scope of the project was limited to developing a markerless motion capture system which could estimate the 3D pose of a cheetah from multi-view video footage.

An important design aspect of this system was that it was compatible with commercially available cameras and could leverage already existing video footage, specifically, videos of cheetahs captured during previous projects. In addition to this existing footage, additional footage was captured at the Ann van Dyk Cheetah Centre in the North-West province, South Africa, as well as at Cheetah Outreach in Cape Town, South Africa at the beginning of this project. All of the footage was recorded using GoPro cameras.

Figure 1.2: A picture of Phantom, the cheetah, during a run at the Ann van Dyk Cheetah Centre in March, 2019

While commercial systems exist for marker-based motion estimation, obtaining accurate position measurements without the use of markers from video footage introduces many complexities. Without markers, positions on a subject which we want to track need to be estimated. This results in a certain amount of uncertainty in each estimation. Additionally, these estimations may be completely wrong sometimes. With varying lighting conditions, occlusion from grass, dust and the subject itself, it is expected that many position measurement estimates will be hard to determine. It is therefore essential to develop techniques which can cope with the uncertainty in determining these measurements to build a robust, markerless motion capture solution.

Figure 1.3: Dust occlusion during a cheetah run at the Ann van Dyk Cheetah Centre

1.3 Project Objectives

This project aims to explain the theory behind and describe steps taken to develop and evaluate a 3D markerless motion capture system. The high-level objectives of the project are to:

1. Discuss the various components and processes required to make up a multi-camera, 3D, markerless motion capture system;
2. Investigate the theory behind components which make up such a system;
3. Create a plan for building and developing a markerless motion capture system;
4. Document the development and implementation of several markerless motion capture techniques;
5. Evaluating the system using appropriate metrics using simulated data;
6. Show how the designed system performs on real-world data;

1.4 Project Scope

The outcomes of the project are largely based on the implementation and evaluation of different methods which will be used to estimate the 3D pose of a cheetah using data obtained from a neural network. The project's scope was constrained as follows:

1. An existing tool for extracting the 2D position estimates of points on the cheetah would be used (DeepLabCut);
2. A simulated cheetah run would be used to evaluate various techniques for 3D motion capture quantitatively;
3. Various issues, such as outlier measurements and noise, would be simulated to evaluate the 3D motion capture methods;
4. Two real multi-view videos would be used to evaluate the performance of the different motion capture methods visually;
5. Multi-view videos which were to be used were obtained using a set of six GoPros arranged in a grid;
6. A simulated camera arrangement of the six GoPros would be used in the simulated tests.

1.5 Project Outline

The project consists of three main sections, shown by Figure 1.4. The first chapter provides an overview of the project and describes the scope and aims of what was to be done. Chapter two gives an overview of the related literature and theory. The third chapter provides the project requirements and details the approach taken in designing and evaluating the various methods for 3D markerless motion capture. Chapter three also describes the setup which was required for the motion capture techniques. This includes describing how the 2D estimates were obtained from DeepLabCut, how the cheetah was modelled, how the cameras were modelled and calibrated, and showing the accuracy of the data from DeepLabCut. Chapters four, five and six each focus on a method for motion capture. Every chapter details the design, implementation and evaluation of the method, and ends off with a summary of the results. Chapter seven concludes the thesis with a discussion and suggests future work to be performed.

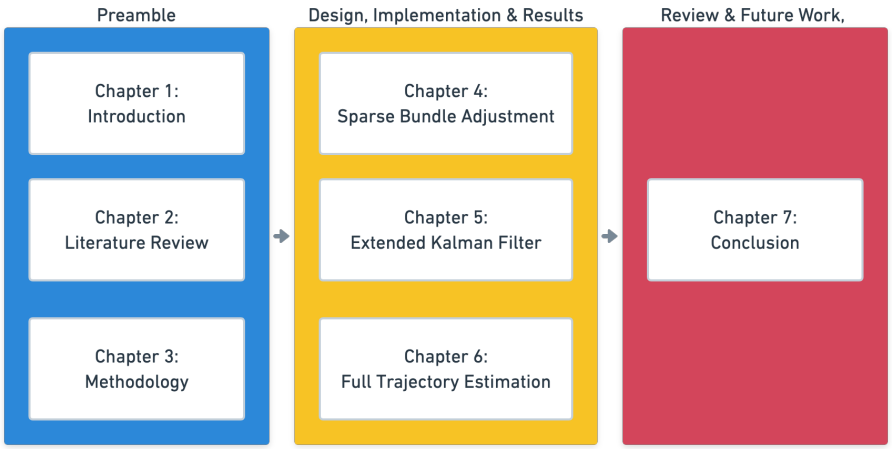

Figure 1.4: Project outline

1.6 Project Outcomes

Given the objectives and scope above, the expected outcomes could be grouped into several sections:

1.6.1 Neural Network

In order to obtain 2D estimates of joints on the cheetah, it was decided that a neural network-based tool be used. DeepLabCut, which is a tool for estimating the pixel positions of an animal's joints from video footage, need to be configured for cheetahs. The outcomes for 2D estimation were to:

1. Select a set of positions, known as *markers*, on the cheetah which we wanted to estimate;
2. Label data corresponding to the selected markers;
3. Train DeepLabCut's neural network;
4. Process a set of videos to be used in the project;
5. Obtain accuracy metrics for the neural network.

1.6.2 Cameras

This section pertains to the modelling and calibration of cameras in order to be able to use multi-video footage for 3D reconstruction. Several outcomes

were needed to achieve this. These were to:

1. Calibrate a camera's intrinsic parameters using a calibration object;
2. Calibrate extrinsic camera parameters for multiple cameras using a calibration object;
3. Compensate for lens distortion;
4. Store configurations of these parameters for use in reconstruction.

1.6.3 3D Reconstruction

The main purpose of the project was to capture 3D motion data of a cheetah. In order to evaluate the performance of various techniques, a simulation was used. These methods were then tested on real data. The outcomes for 3D estimation were to:

1. Create a kinematic model of the cheetah;
2. Simulate a cheetah run, known as a *trajectory*, using the above model;
3. Simulate effects such as noise and outlier measurements to determine their effects on the trajectory estimation for each motion capture technique;
4. Evaluate the performance of several 3D motion capture techniques based on their ability to reconstruct these trajectories;
5. Show how these methods perform on the selected set of real data.

Chapter 2

Literature Review

The following chapter provides an overview of the literature around motion capture, techniques for markerless pose estimation, camera modelling, approaches to 3D reconstruction, and model-based approaches to state estimation. An overview of the theory relevant to subsequent chapters in this thesis is also provided.

2.1 3D Vision Based Motion Capture

Motion capture is typically regarded as the process of recording the kinematics of a subject or object. A survey by Moeslund et al. [12] describes two main categories for types of motion capture. *Active* motion capture techniques make use of devices or equipment which are attached to a subject to measure generated motion signals. Examples of active techniques include attaching sensors such as accelerometers, gyroscopes and GPS trackers to a subject to estimate its motion. *Passive* motion capture techniques make use of the environment to measure the motion of a subject without having to attach devices to the subject. Examples of passive motion capture techniques include using cameras to record footage of a subject and the measurement of electromagnetic waves using radar [13], shown in Figure 2.1. The use of markers on subjects is usually seen as only mildly invasive and is, therefore, usually classified as a passive technique. A later survey in 2006 [14] notes the increase in popularity of computer vision-based approaches to motion capture due to their non-invasive nature.

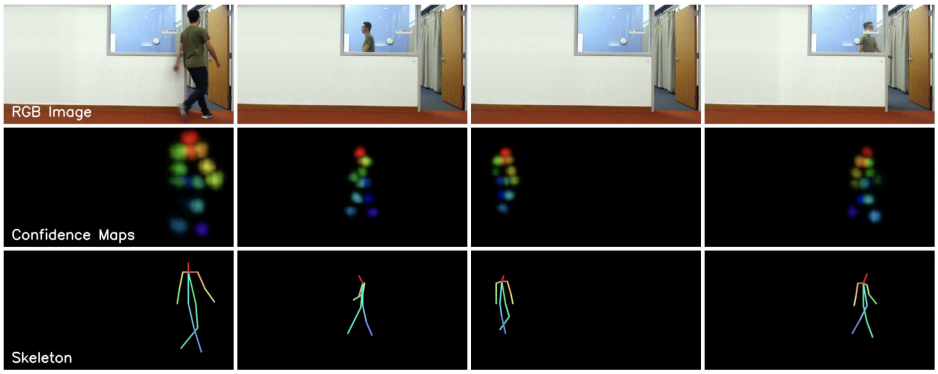

Figure 2.1: Through wall human pose estimation using radar [13]

Some commercial vision-based motion capture systems, such as those made by companies like Vicon, make use of infrared (IR) lights and multiple cameras which observe reflective markers on a subject. After a calibration procedure, it is possible to obtain 3D estimates of the marker positions. These systems can be prohibitively expensive, but measurements from these systems are often used as benchmarks in motion capture studies due to their sub-millimetre accuracy when locating markers in a 3D scene [15].

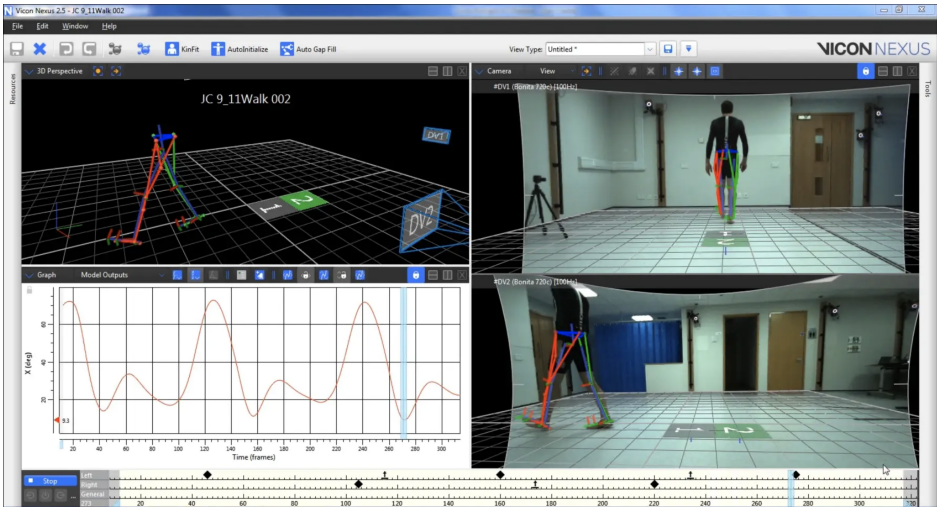

Figure 2.2: Screenshot of Vicon's Nexus software used for human motion capture [16]

In recent years, there has been increased interest in motion capture in consumer electronics due to the emergence of virtual reality (VR) and augmented

reality (AR). Modern game consoles such as Microsoft's Xbox 360 allowed for the use of external sensors, such as the Xbox Kinect, to track the motion of its users in order to allow them to interact with games via their body motions. The Xbox Kinect makes use of a colour camera as well as a depth sensor consisting of an IR projector and IR camera to determine the 3D pose of a human subject [17]. The Kinect projects an IR pattern which is seen by the IR camera, which enables a subject's 3D position to be estimated. The Kinect has become popular amongst researchers due to its high accuracy in estimating the depth of a subject from the camera [18]. Unfortunately, the Kinect is only rated to work up to ranges of 3.6 metres and requires an indoor environment to use as sunlight drowns out the IR light from the IR projector. Consumer stereo cameras have also become available, targeting areas such as autonomous robotics, AR and VR, specifically to be able to track subjects in 3D or estimate a depth map of a scene. Two examples of such cameras are Intel's RealSense cameras[1] and Stereolabs' ZED cameras[2]. These stereo cameras are capable of accurately capturing 3D depth data at distances up to 20m.

2.2 Convolutional Neural Networks and Pose Estimation

In recent years, there have been large improvements in computer-based pattern recognition in deep learning, specifically convolutional neural networks (CNNs) [19], [20], [21], [22]. It is now common to see CNNs performing well in various pattern recognition tasks such as character recognition, image classification, image segmentation and pose estimation, to name a few. There are several datasets which are now commonly used to train, benchmark and compare CNNs [23], [24], [25]. In some cases, given a large enough dataset, CNNs outperform humans.

In the fields of computer vision and deep learning, *pose estimation* describes the task of reconstructing the pose of a subject in the form of a skeletal model or geometric mesh given visual information such as images or videos [26]. Unfortunately, pose estimation is a term also associated with the process of estimating a camera's position and orientation using the data it has captured. In this thesis, it will be made clear as to which type of pose estimation is being referred to.

[1] `https://www.intelrealsense.com/stereo-depth/`

[2] `https://www.stereolabs.com/zed/`

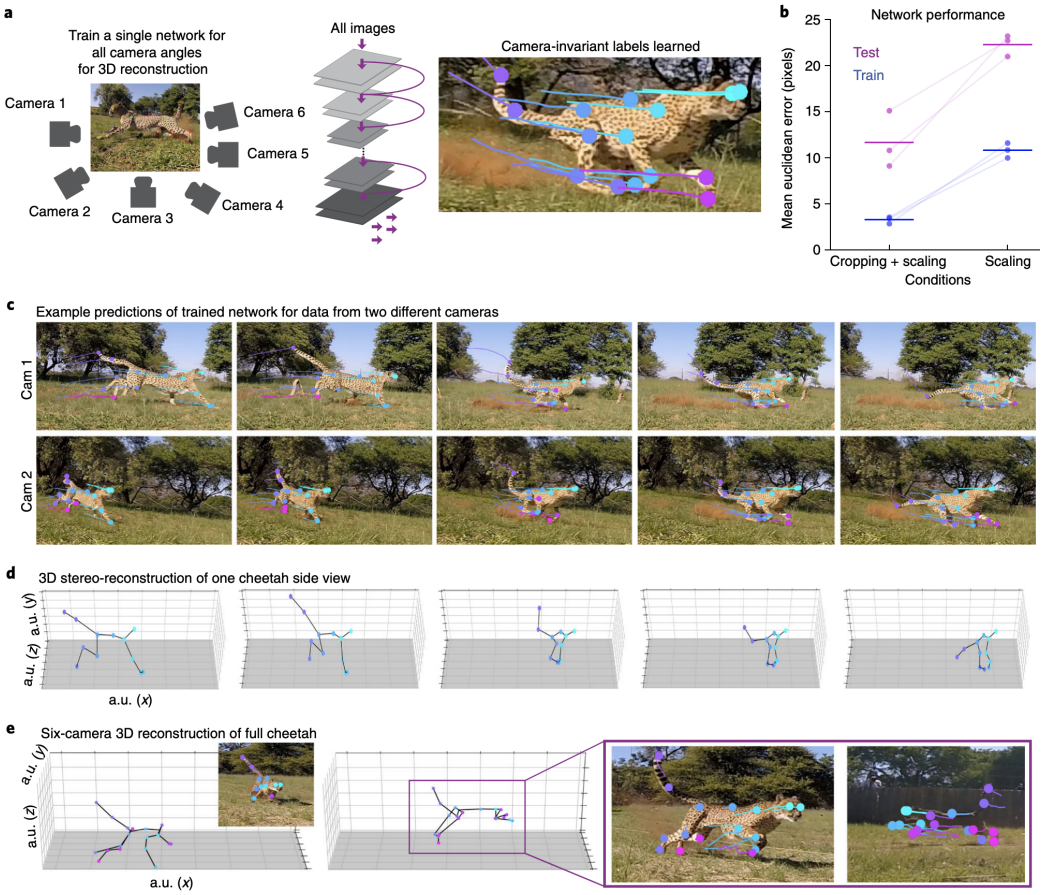

Figure 2.3: Overview of the process of using DeepLabCut for 3D reconstruction using triangulation [8]

Pose models typically approximate the pose of a subject as a set of joints and links, creating a stick-figure pose representation [26]. Figure 2.3 shows various stages of a pipeline for using deep learning to perform a 3D trajectory estimation for a simple joint and link based pose model from multi-view video footage. More complex pose models, such as Facebook's DensePose [27], can approximate the geometry of a subject and even extract textures from the visual information which are then mapped to the geometry.

Pose estimation software such as [7], [26] make use of CNNs to estimate the pose of humans by identifying locations of body parts within an image. Notably, [7] made use of pairwise correspondence between possible body part locations to group body part hypotheses from the CNNs into valid pose configurations from which a pose could be determined. When processing

video footage, these tools work on a frame by frame basis and do not take into account temporal information between frames.

The majority of pose estimation research has been performed on human subjects as there are several large, high-quality datasets [28], [29] which already exist for both 2D and 3D pose estimation. This makes it easier for researchers to start experimenting with the data, as well as provides them with a way to benchmark various pose estimation techniques using the same data. Unfortunately, there are far fewer datasets available which contain animal poses, especially for animals like cheetahs.

CNNs typically require a large amount of data to use for training in order to produce good results. There are several tools aimed at performing pose estimation on animals such as DeepLabCut [8], LEAP [9] and DeepPoseKit [10]. These tools typically work in a cycle of labelling images, training the CNN and evaluating the performance of the newly trained network. If the results are not satisfactory after training, more data is labelled, and the cycle is repeated until the results are good enough to use the network for the task at hand. In order to reduce the amount of labelled data that is needed, they employ techniques like transfer learning and data augmentation.

Transfer learning is the re-purposing of a network to be used for a different task than that for which it was trained. This is done to reduce the time needed to train a new network and to reduce the data needed to train the network. When a CNN is trained on a large dataset, it learns generalisations (called features) of the data which it is trained on [30]. For instance, if one trains a CNN to classify types of birds, internal parts of the network may learn to identify simple patterns like lines and corners as well as compositions of these patterns such which look like feathers and beaks.

By using a pre-trained network as the starting point, the filters within the CNN can be reused and do not need to be learnt from scratch. This means far less training time and data is required to achieve satisfactory results. This is provided that the filters that were initially learnt are applicable to the new task. Pre-training CNNs on the large ImageNet dataset [24] has become common practice due to the generalised features learnt during pre-training boosting the performance of networks on other tasks [31].

Data augmentation is another technique used to improve the performance of CNNs by artificially expanding a dataset. The usual techniques employed to do this for an image dataset are the scaling, translating, rotating and cropping of the images [32].

2.3 Cameras

2.3.1 Pinhole Cameras

In essence, a camera is a device which forms an image by capturing light projected from the 3D world it sees onto a 2D surface. Early forms of cameras operated by allowing light into a dark box or room through a small aperture called a pinhole. Rays of light shine through the hole, projecting an image of the world outside onto a surface inside.

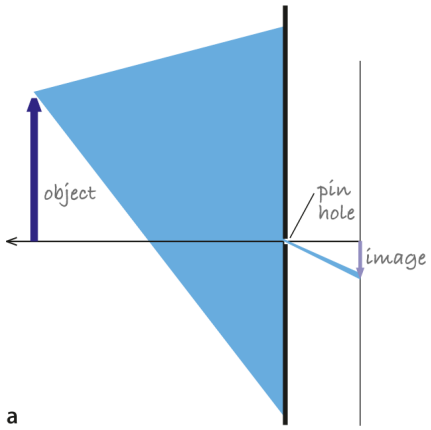

Figure 2.4: Diagram of a pin-hole projection [33]

The resulting projected image appears upside down and dimly lit. The image is dimly lit since only a small amount of radiant power can pass through the aperture. The radiant power is equal to the product of the luminance of the outside scene (measured in Wm^{-2}) and the area of the pinhole. Modern cameras overcome this problem by using lenses to focus light from larger apertures onto the image [33]; however, this results in a loss of focus at specific distances from the camera causing a blurring effect. Scenes observed by pinhole cameras are always in focus as long as their apertures are small enough. An ideal pinhole camera is said to have an infinitely small aperture; this means all projected rays pass through a single point. For an ideal pinhole camera, this point is known as the centre of projection.

2.3.2 The Projective Camera

In the field of computer vision, the projective camera model is commonly used to provide a geometric relationship between 3D points in the world and their

corresponding projected 2D image points. A linear transformation directly maps 3D points in the world onto an image plane in front of the camera. As shown in Figure 2.5, the projective camera model is closely aligned with the ideal pinhole camera; however, instead of having rays project onto an image plane behind the centre of projection, rays are instead projected through an image plane in front of the centre of projection.

The centre of the camera's reference frame is aligned with the centre of projection with Z-axis passing through the centre of the image plane. The Z-axis is referred to as the *optical axis* of the camera and the centre of the image plane is referred to as the *principle point*. The Z coordinate of a point in the camera's reference frame denotes the *depth* of the point.

In order to project a 3D point onto the image plane, a line is found which passes through the centre of projection as well as the observed 3D point. The intersection of this line and the image plane provides us with the coordinate of the projected point on the image plane.

Digital images are divided into discrete elements known as pixels. In computer vision, it is common to use the letters u and v to represent the horizontal and vertical coordinates of a pixel with $(u, v) = (0, 0)$ being the top left corner of an image.

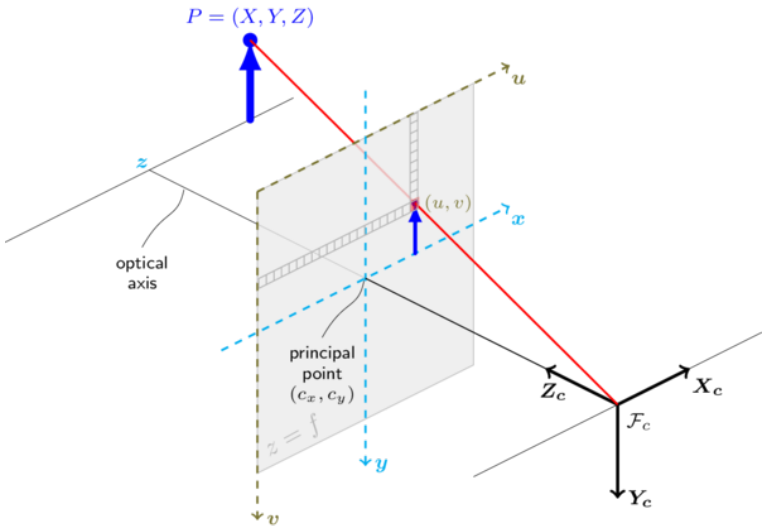

Figure 2.5: Diagram of a pin-hole camera projection model from the OpenCV software library [34]

The mapping of 3D points in a global coordinate frame into 2D image points

observed by a camera is performed through a projective transformation. The projective transformation is composed of several operations or steps.

Firstly, a coordinate transformation is applied to points in the global reference, making them relative to the camera's reference frame. A 3×3 rotation matrix $\mathbf{R}$ and 3×1 translation vector $\mathbf{t}$ are used to transform the points shown by Equation 2.1 where the subscript for coordinates denotes the frames which they are referenced in. $\mathbf{R}$ and $\mathbf{t}$ are said to define the *pose* of the camera.

$$\begin{bmatrix} X_c \\ Y_c \\ Z_c \end{bmatrix} = \mathbf{R} \begin{bmatrix} X_w \\ Y_w \\ Z_w \end{bmatrix} + \mathbf{t} \tag{2.1}$$

By using a homogeneous coordinate system, this transformation can be represented as a single linear transformation.

$$\begin{bmatrix} X_c \\ Y_c \\ Z_c \end{bmatrix} = \begin{bmatrix} r_{1,1} & r_{1,2} & r_{1,3} & t_1 \\ r_{2,1} & r_{2,2} & r_{2,3} & t_2 \\ r_{3,1} & r_{3,2} & r_{3,3} & t_3 \end{bmatrix} \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix} = \begin{bmatrix} \mathbf{R} & | & \mathbf{t} \end{bmatrix} \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix} \tag{2.2}$$

Given a point $\mathbf{P} = \begin{bmatrix} X_w & Y_w & Z_w \end{bmatrix}^{\mathbf{T}}$ in the camera's reference frame, the projected point $\mathbf{P'}$ which lies on the plane $z = f$ is expressed in 3D non-homogeneous camera coordinates by Equation 2.3.

$$\mathbf{P'} = f \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} \quad \text{where} \quad x' = \frac{x}{z} \quad \text{and} \quad y' = \frac{y}{z} \tag{2.3}$$

This projection is shown in Figure 2.6 below.

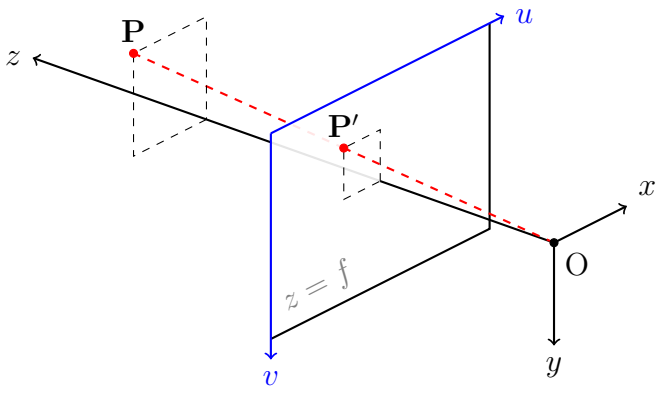

Figure 2.6: Diagram of 3D point P being projected onto an image plane

The top-down view in Figure 2.7 helps to show that in order to project a 3D point onto the image plane, the point simply needs to be scaled until its depth parameter is the same as the plane's. In a projective space, such as the image plane, all 3D points which share the same ratio between their coordinates are mapped to the same point.

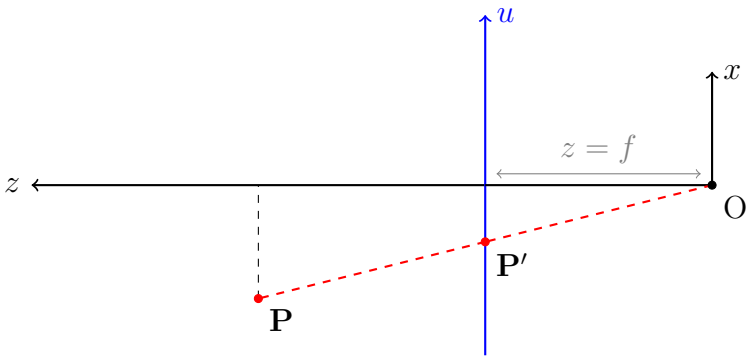

Figure 2.7: Top down view of Figure 2.6

The point $\mathbf{P}'$ on the image plane can now be mapped to pixel coordinates of the camera. This is done by scaling x' and y' by the number of pixels per unit length of the camera's coordinate system. We denote these scaling parameters as s_x and s_y.

$$x'' = s_x f x' \quad \text{and} \quad y'' = s_y f y' \tag{2.4}$$

It is common to see the scaling and focal length parameters grouped together as $f_x = fs_x$ and $f_y = fs_y$.

The final step in order to obtain the u and v pixel coordinates is to offset x'' and y'' from the principle point. The parameters c_x and c_y denote the pixel offset from the principal point. This results in Equation 2.5.

$$u = f_x x + c_x \quad \text{and} \quad v = f_y y + c_y \tag{2.5}$$

Sometimes, due to manufacturing defects, the pixels in a camera's sensor may be slightly skewed, resulting in the u and v axes not being orthogonal. To correct for this, a skew factor α can be added to Equation 2.5 giving Equation 2.6. The skew parameter is often omitted as the effects of skewing are negligible for most modern cameras.

$$u = f_x x + \alpha y + c_x \quad \text{and} \quad v = f_y y + c_y \tag{2.6}$$

The matrix form of the projection is shown in Equation 2.7. The matrix $\mathbf{K}$ is known as the *camera calibration matrix*.

$$\begin{aligned} \lambda \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} &= \begin{bmatrix} f_x & \alpha & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{1,1} & r_{1,2} & r_{1,3} & t_1 \\ r_{2,1} & r_{2,2} & r_{2,3} & t_2 \\ r_{3,1} & r_{3,2} & r_{3,3} & t_3 \end{bmatrix} \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix} \\ &= \mathbf{K} \begin{bmatrix} \mathbf{R} & | & \mathbf{t} \end{bmatrix} \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix} \quad \text{where} \quad \lambda \in \mathbb{R}; \lambda \neq 0 \end{aligned} \tag{2.7}$$

It is essential to note the scaling parameter λ. This is because any point that lies on a ray passing through the centre of projection will be projected to the same point on the image plane. This *scale ambiguity* is a result of information lost through the dimensionality reduction in the projection. The result is that the scale of objects observed by a camera cannot be determined from images alone; external information is needed from somewhere.

The camera calibration matrix and transformation matrix can be grouped into a single 3×4 matrix $\mathbf{P}$ known as the *projection* matrix or *camera* matrix.

$$\mathbf{P} = \mathbf{K} \begin{bmatrix} \mathbf{R} & | & \mathbf{t} \end{bmatrix} \tag{2.8}$$

2.3.3 Lens Distortion

The pinhole camera model described in the previous section provides a useful way of obtaining a mapping between 3D points in the world and the 2D pixel coordinates of a camera. However, this model does not account for the distortion effects caused by a camera's lens. Typically, modifications are made in mapping from image plane coordinates to pixel coordinates. Depending on the type of lens distortion, different mappings need to be used. Some examples of types of distortion are shown in Figure 2.8.

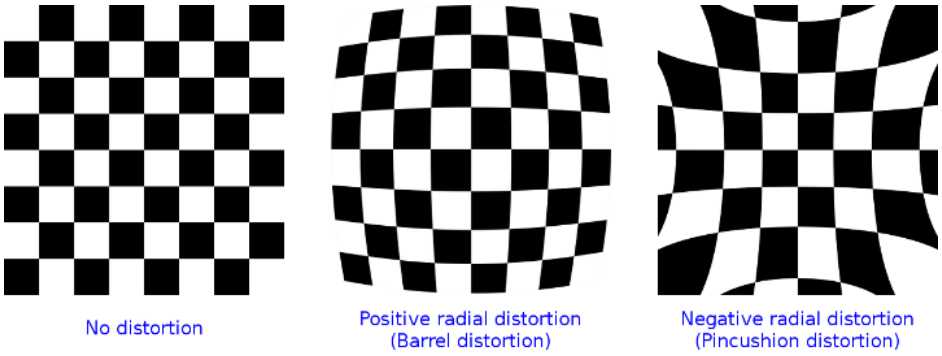

Figure 2.8: A diagram showing two types of radial distortion applied to a chessboard pattern from the OpenCV library documentation [34]

While there are various distortion models, two common types of lens distortion models which are part of the OpenCV library are described below. These assume that x' and y' from Equation 2.3 have been calculated. For each of the distortion models in the OpenCV library, the variable parameters for the model are grouped into an array $\mathbf{D}$. This allows for the intrinsic parameters of a camera to be completely defined by the camera's calibration matrix $\mathbf{K}$ and distortion parameters $\mathbf{D}$.

Standard Lens Distortion

OpenCV's default camera distortion model can account for three different types of distortion: *radial distortion*, *tangential distortion* and *thin-prism distortion*. In Equation 2.9, $k_1..k_6$ are the radial distortion coefficients, p_1 and p_2 are the tangential distortion coefficients, and $s_1..s_4$ are the thin prism

distortion coefficients. The use of all parameters is not required and by default in OpenCV, only $k_1..k_4$ are used while the other parameters are set to zero. For OpenCV's standard lens distortion model, x'' and y'' are not scaled (since the focal length is not taken into account at this stage but rather when mapping to pixel coordinates):

$$x'' = x' \frac{1 + k_1 r^2 + k_2 r^4 + k_3 r^6}{1 + k_4 r^2 + k_5 r^4 + k_6 r^6} + 2p_1 x' y' + p_2(r^2 + 2x'^2) + s_1 r^2 + s_2 r^4$$

$$y'' = y' \frac{1 + k_1 r^2 + k_2 r^4 + k_3 r^6}{1 + k_4 r^2 + k_5 r^4 + k_6 r^6} + p_1(r^2 + 2x'^2) + 2p_2 x' y' + s_3 r^2 + s_4 r^4 \quad (2.9)$$

$$\text{where} \quad r^2 = x'^2 + y'^2$$

The pixel coordinates, u and v, can be found using Equation 2.10.

$$u = f_x x'' + c_x \quad \text{and} \quad v = f_y y'' + c_y \quad (2.10)$$

The distortion parameters are grouped into an array $\mathbf{D}$.

$$\mathbf{D} = [k_1, k_2, k_3, k_4, k_5, k_6, p_1, p_2, s_1, s_2, s_3, s_4] \quad (2.11)$$

Fisheye Lens Distortion

The fisheye lens distortion model helps model the extreme radial distortion seen in cameras like GoPros. While OpenCV's default camera model can account for radial distortion, the fisheye model is specifically designed to cope with high degrees of radial distortion better than the standard model. In Equation 2.13, $k_1..k_4$ are the radial distortion coefficients. Distortion from fisheye lenses is accounted for as follows:

$$\theta = \arctan(r) \quad \text{where} \quad r^2 = x'^2 + y'^2 \quad (2.12)$$

$$\theta_d = \theta(1 + k_1\theta^2 + k_2\theta^4 + k_3\theta^6 + k_4\theta^8) \quad (2.13)$$

$$x'' = \frac{\theta_d}{r} x' \quad \text{and} \quad y'' = \frac{\theta_d}{r} y' \quad (2.14)$$

The pixel coordinates for the fisheye lens distortion model, u and v, can be found using Equation 2.15.

$$u = f_x(x'' + \alpha y'') + c_x \quad \text{and} \quad v = f_y y'' + c_y \quad (2.15)$$

The distortion parameters for the fisheye lens distortion model are grouped into an array $\mathbf{D}$.

$$\mathbf{D} = [k_1, k_2, k_3, k_4] \quad (2.16)$$

2.3.4 Calibration Methods

Calibration is the process of obtaining the various parameters of a camera model to match the characteristics of a real camera closely. The calibration process is typically split into two parts. *Intrinsic* calibration determines the camera calibration matrix $\mathbf{K}$ and other parameters internal to the camera such as the distortion coefficients $\mathbf{D}$. *Extrinsic* calibration is used to determine the camera's pose, defined by $\mathbf{R}$ and $\mathbf{t}$.

Typically a patterned, planar calibration object is used to calibrate cameras. Patterns are used to make it possible for specific landmarks on the calibration object to be identified through image processing techniques. Chessboard patterning is commonly used as it allows for a grid pattern of corners to be detected using a filter like the Harris detector [35]. These calibration objects can be used to define known 3D point positions in a world frame or to define sets of 3D points where the relative position of the points is known. Several analytical and numerical techniques have been developed to allow for intrinsic and extrinsic camera parameters to be determined.

Hartley and Zisserman's book on multi-view geometry in computer vision [36] describe methods for determining projective transformations and calibrating cameras using 2D to 2D and 3D to 2D point correspondences in great detail. The following sections on camera calibration draw primarily from this book. Several other books such as [33] and [37] provide good explanations of these topics as well and were also used as references for the following sections.

2.3.5 Direct Linear Transformation

The direct linear transformation (DLT) is an algorithm for determining a projective transformation or homography $\mathbf{H}$ which relates 2D to 2D point correspondences, $\mathbf{x}'_i$ and $\mathbf{x}_i$, from two cameras through Equation 2.17. In a projective space, $\mathbb{P}^2$, the 2D points are represented by homogeneous 3-vectors. $\mathbf{H}$ is therefore a 3×3 matrix. These 3-vectors can be thought of as representing lines in a 3D Euclidean space which pass through the 3D point and a centre of projection.

$$\mathbf{x}'_i = \mathbf{H}\mathbf{x}_i \tag{2.17}$$

It is important to note that as the vectors $\mathbf{x}'_i$ and $\mathbf{H}\mathbf{x}_i$ are homogeneous, they are not necessarily equal in magnitude but share the same direction. By using this property, a linear solution for $\mathbf{H}$ can be obtained from Equation 2.18.

However, the solution for $\mathbf{H}$ is *scale invariant*[3].

$$\mathbf{x}_i' \times \mathbf{H}\mathbf{x}_i = \mathbf{0} \tag{2.18}$$

In order to obtain a unique solution up to scale, four-point correspondences are required. It is also crucial that no three points are collinear as this leads to a *degenerate transformation* and results in no unique solution for $\mathbf{H}$. When more than 4 point correspondences are used, due to noise in measurements, the solution will likely be over-determined. In these cases, the resulting error is minimised through techniques such as least-squares. Points are usually of the magnitude $\mathbf{x}_i = (1000, 1000, 1)^T$. In the over-determined case, normalisation is imperative to avoid round off errors due to the large difference in order of magnitude of terms that arise. The DLT algorithm forms the basis for several other calibration methods.

Determining the Intrinsic Parameters

Determining the camera matrix $\mathbf{P}$ is referred to as resectioning. There are two main ways that these parameters can be obtained, either through a set of 3D to 2D point projections or through a set of 2D to 2D point correspondences seen by two cameras. Given a 3D point $\mathbf{X}_i$, its corresponding 2D projection $\mathbf{x}_i$ is found through Equation 2.19.

$$\mathbf{x}_i = \mathbf{P}\mathbf{X}_i \tag{2.19}$$

Similarly to the DLT algorithm, using the fact that the direction of $\mathbf{x}_i$ and $\mathbf{P}\mathbf{X}_i$ are the same, Equation 2.20 is obtained. Note that the only difference between Equation 2.20 and Equation 2.18 is the dimensionality of the problem. $\mathbf{P}$ is a 3×4 matrix and $\mathbf{X_i}$ is a 4-vector.

$$\mathbf{x}_i \times \mathbf{P}\mathbf{X}_i = \mathbf{0} \tag{2.20}$$

In the same vein as the DLT algorithm, a unique solution can be found using $5\frac{1}{2}$ 3D to 2D point correspondences where only either the x or y coordinate is needed from the 6th point correspondence. Degenerate configurations, over-determined configurations and normalisation are taken care of in a similar manner to that of the DLT.

[3]A property resulting in mathematical laws or objects being unchanged after multiplication by some factor.

In order to obtain the camera parameters, *QR decomposition*[4] is used to decompose $\mathbf{P}$ into $\mathbf{K}$, $\mathbf{R}$ and $\mathbf{t}$. While the extrinsic parameters $\mathbf{R}$ and $\mathbf{t}$ are only valid for the current scene, the camera calibration matrix $\mathbf{K}$ can be reused.

Estimating distortion parameters $\mathbf{D}$ can be performed through an iterative process using geometric constraints from the calibration object. The most prevalent type of distortion in cameras is radial distortion. A common approach to compensate for this is to use the property that straight lines in a scene should project to straight lines in the projected image. By finding points which lie on straight lines in a scene, the error associated with the deviation of the projected points from a line can be minimised. Zhang [38] introduced a method for camera matrix and radial distortion calibration using a single, planar calibration object. Zhang's method is currently the standard method for calibrating intrinsic camera parameters in both the OpenCV library [34] and the MATLAB camera calibration toolbox [39].

Extrinsics and Multiple Cameras

Given a 2D point $\mathbf{x}$ in the view of one camera, constraints can be obtained for a point $\mathbf{x}'$ viewed by a second camera. The relationship between the two points is described by a 3×3 matrix $\mathbf{F}$, known as the fundamental matrix. This matrix encapsulates epipolar geometric constraints between the two views through Equation 2.21, Equation 2.22 and Equation 2.23. *Epipolar points* or *epipoles*, denoted by $\mathbf{e}$ and $\mathbf{e}'$ in Figure 2.9, are the intersection points of a line joining the two centres of projection of the views and their image planes. This line joining the two centres of projection is called a baseline. *Epipolar planes* are any planes which intersect this baseline. *Epipolar lines*, denoted by $\mathbf{l}'$ in Figure 2.9, correspond to a projection of an epipolar plane onto an image plane.

$$\mathbf{x'^T F x} = \mathbf{0} \tag{2.21}$$

$$\mathbf{l}' = \mathbf{F}\mathbf{x} \quad \text{and} \quad \mathbf{l} = \mathbf{F}^T\mathbf{x}' \tag{2.22}$$

$$\mathbf{F}\mathbf{e} = \mathbf{0} \quad \text{and} \quad \mathbf{F}^T\mathbf{e}' = \mathbf{0} \tag{2.23}$$

[4]QR decomposition factorises a single matrix into an orthogonal matrix Q and upper triangular matrix R.

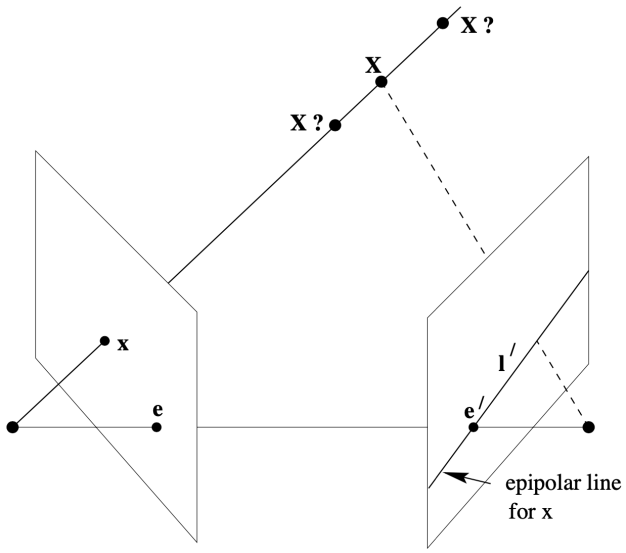

Figure 2.9: Diagram of the epipolar geometry for point-line correspondence between two views [36]

A special form of the fundamental matrix known as the essential matrix, $\mathbf{E}$, can be used to encapsulate the epipolar geometry between two *normalised cameras*. A normalised camera has a calibration matrix $\mathbf{K}$ which is identity. The projection matrix of a camera can be converted into a normalised camera by left multiplying by the inverse of the camera matrix $\mathbf{K}$.

$$\mathbf{P}_{normalised} = \mathbf{K}^{-1}\mathbf{P} = \begin{bmatrix}\mathbf{R} & | & \mathbf{t}\end{bmatrix} \tag{2.24}$$

Similarly, a 2D point $\mathbf{x}$ can be converted to a normalised point $\hat{\mathbf{x}}$ by left multiplication of the inverse of the camera matrix $\mathbf{K}$.

$$\hat{\mathbf{x}} = \mathbf{K}^{-1}\mathbf{x} = \begin{bmatrix}\mathbf{R} & | & \mathbf{t}\end{bmatrix}\mathbf{X} \tag{2.25}$$

For a pair of normalised camera projection matrices, $\mathbf{P} = \begin{bmatrix}\mathbf{I} & | & \mathbf{0}\end{bmatrix}$ and $\mathbf{P}' = \begin{bmatrix}\mathbf{R} & | & \mathbf{t}\end{bmatrix}$, the equation shown in Equation 2.26 defines the correspondence for normalised image coordinates. It can then be shown that Equation 2.27 is obtained.

$$\hat{\mathbf{x}}'\mathbf{E}\hat{\mathbf{x}} = \mathbf{0} \tag{2.26}$$

$$\mathbf{E} = \mathbf{K}'^{T}\mathbf{F}\mathbf{K} = [\mathbf{t}]_{\times}\mathbf{R} \tag{2.27}$$

A matrix decomposition can then be performed, splitting $\mathbf{E}$ into a skew-symmetric matrix [5] $[\mathbf{t}]_\times$ and a rotation matrix $\mathbf{R}$. There are four possible solutions corresponding to possible reconstructions; however, only one of the four corresponds to a point which lies in front of the image plane in both views.

There are several ways to compute the fundamental matrix $\mathbf{F}$. A minimum of 7 point correspondences is needed to compute a linear solution for $\mathbf{F}$. This minimum solution exploits the fact that $\mathbf{F}$ is singular. A simpler method, known as the 8-point-algorithm [40], can be used to compute $\mathbf{F}$ using least-squares on a set of linear equations. Normalisation through scaling and translating of image points is used to improve the conditioning of the problem. There are also non-linear algorithms which are capable of computing $\mathbf{F}$ using a minimal set of 5 points such as [41] and [42].

The gold-standard algorithm for computing $\mathbf{F}$ involves computing an initial maximum-likelihood estimate $\hat{\mathbf{F}}$ and choosing our camera projection matrices $\mathbf{P} = \begin{bmatrix} \mathbf{I} & | & 0 \end{bmatrix}$ and $\mathbf{P}' = \begin{bmatrix} [\mathbf{e}']_\times & | & \mathbf{e}' \end{bmatrix}$ where $\mathbf{e}'$ is obtained from $\hat{\mathbf{F}}$. A 3D estimate $\hat{\mathbf{X}}_i$ is then obtained for each point correspondence $\mathbf{x}_i$ and $\mathbf{x'}_i$ through triangulation techniques which are discussed in the next section. From this, an estimate of the point correspondences is calculated from the projection equations $\mathbf{x}_i = \mathbf{P}\hat{\mathbf{X}}_i$ and $\mathbf{x'}_i = \mathbf{P}'\hat{\mathbf{X}}_i$. The geometric distance between the image points and their estimates is then minimised, shown by Equation 2.28. Typically the Levenberg-Marquardt algorithm is used to perform the minimisation.

$$\min_{\hat{\mathbf{F}}, \hat{\mathbf{X}}} \quad \sum_i \|\mathbf{x}_i - \hat{\mathbf{x}}_i\|^2 + \|\mathbf{x'}_i - \hat{\mathbf{x}'}_i\|^2 \tag{2.28}$$

When using calibrated cameras, by converting to normalised coordinates, the essential matrix $\mathbf{E}$ can be computed using the above techniques and the extrinsic parameters $\mathbf{R}$ and $\mathbf{t}$ can be obtained.

2.4 3D Triangulation

The process of triangulation tries to estimate a 3D point $\hat{\mathbf{X}}$ from two corresponding 2D image points $\mathbf{x}$ and $\mathbf{x}'$. There are several methods to compute $\hat{\mathbf{X}}$; however; under the assumption of Gaussian measurement noise, a provably optimal solution can be obtained using a method developed by Hartley

[5] This notation is used to represent cross products as matrix multiplications (where the matrix is skew-symmetric and has a null-space corresponding to the vector enclosed by the square brackets).

and Sturm [43]. The optimal method estimates $\hat{\mathbf{X}}$ by using its projections, $\hat{\mathbf{x}}$ and $\hat{\mathbf{x}'}$.

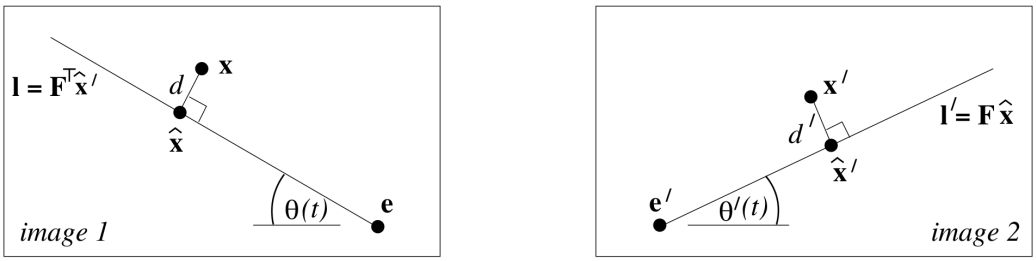

Figure 2.10: Diagram of the minimisation problem for triangulation [36]

From the estimated point $\hat{\mathbf{x}}$ in the 1st view, an epipolar line $\mathbf{l}' = \mathbf{F}\hat{\mathbf{x}}$ can be obtained in the 2nd. Similarly for $\hat{\mathbf{x}}'$ in the 2nd view, a line $\mathbf{l} = \mathbf{F}^T\hat{\mathbf{x}}'$ is obtained in the 1st view. By minimising the perpendicular distance from $\mathbf{x}$ to $\mathbf{l}$ and $\mathbf{x}'$ to $\mathbf{l}'$ a maximum likelihood estimation of $\hat{\mathbf{X}}$ can obtained through linear methods as $\hat{\mathbf{x}}'^T\mathbf{F}\hat{\mathbf{x}} = 0$ is satisfied. The epipolar lines can be expressed as a function of a single parameter t where $\mathbf{l}(t)$ and $\mathbf{l}'(t)$ are related through $\mathbf{F}$. Thus the minimisation only needs to be performed over a single parameter t. This minimisation can be obtained through iterative methods such as the Levenberg-Marquardt algorithm, however; it is possible to reformulate the problem to find the global minimum using a non-iterative method [43].

2.5 Sparse Bundle Adjustment

Sparse bundle adjustment (SBA) is a technique which allows for the simultaneous optimisation of camera poses ($\mathbf{R}$ and $\mathbf{t}$) and 3D point locations. Sometimes, the camera calibration matrix $\mathbf{K}$ and distortion parameters $\mathbf{D}$ are also optimised. The intended result is to obtain an optimally reconstructed scene as well as the optimal camera parameter estimates [44]. An SBA is often performed as the last step in 3D reconstruction problems to obtain optimal results. [45]

SBA, similarly to the 3D reconstruction problem in the previous section, aims to minimise the reprojection error of estimated 3D points, usually through iterative non-linear least-squares methods. However, instead of only varying parameters related to the points, the camera parameters may also be varied in the minimisation process.

Bundle adjustment refers to the adjustment of "bundles" of lines projected

from the 3D points which meet at the centre of projection for each camera. The reason this problem is known as a *sparse* bundle adjustment is due to the way the Jacobian matrix is populated.

2.5.1 Jacobian Sparsity

Letting the ith 3D point be denoted by, $\mathbf{X}_i$, and a set of parameters for the jth camera be denoted as, $\mathbf{P}^j$, the 3D points and camera parameters can be grouped into a vector of parameters, $\theta = [\mathbf{P^1}, ..., \mathbf{P}^n, \mathbf{X}_1, ..., \mathbf{X}_m]^\mathbf{T}$ which can be varied in the optimisation. Each 2D point estimate for a point i viewed by a camera j is denoted as $\hat{\mathbf{x}}_i^j$, where each coordinate of $\hat{\mathbf{x}}_i^j$ is a function of the parameters. The Jacobian for this problem is computed as per Equation 2.29.

$$\mathbf{J} = \left[\frac{\partial \hat{\mathbf{x}}}{\partial \theta}\right] \tag{2.29}$$

For a 2D point $\hat{\mathbf{x}}_i^j$, which is a projection of the ith 3D point in the jth camera view, we see that the only parameters in θ that can have non-zero partial derivatives are $\mathbf{P}^j$ and $\mathbf{X}_i$ in θ.

$$\frac{\partial \hat{\mathbf{x}}_i^j}{\partial \mathbf{P}^k} = 0 \quad \text{where} \quad j \neq k \tag{2.30}$$

$$\frac{\partial \hat{\mathbf{x}}_i^j}{\partial \mathbf{X}_l} = 0 \quad \text{where} \quad i \neq l \tag{2.31}$$

This results in a sparsely populated Jacobian. Figure 2.11 is an example of a sparsity pattern showing blocks in the Jacobian matrix with non-zero values (which are white) for a problem where 4 points are observed by three cameras.

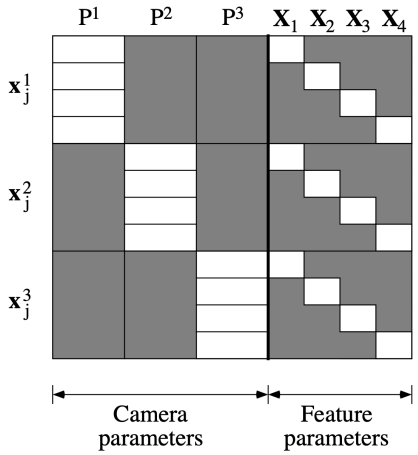

Figure 2.11: Diagram showing the Jacobian sparsity pattern for a SBA with three cameras and 4 3D points [36]

By exploiting the sparse property of the Jacobian, significant computational speed improvements can be gained during the optimisation routine [45].

2.5.2 Robust Cost Functions

Outliers are a common problem when using optimisation techniques to estimate 3D points from 2D point correspondences where least-squares methods are often used to minimise the reprojected 2D point error. Outliers, in this sense, are corresponding 2D points which have high reprojection errors. When under the assumption that measurement noise is Gaussian, least-squares methods typically result in a maximum likelihood estimation for the 3D point estimates. However, when outliers are present, the solution can be skewed towards the outliers. A solution to this is to reduce or remove the effect that outliers have on the cost function by using robust cost functions. For the explanations that follow, a cost C was defined as a function of some error e.

Starting with a least-squares cost, Equation 2.32, it is seen that large errors are heavily penalised. As noted above, outliers have a high cost.

$$C(e) = e^2 \tag{2.32}$$

A simple way to reduce the effect of outliers is to use a function where the cost is proportional to the error instead of having a quadratic relationship. An L1 cost, Equation 2.33, can be used to reduce the effect of outliers; however,

a downside is that the function is not differentiable at $e = 0$. A parameter b can be used to scale the cost.

$$C(e) = 2b|e| \tag{2.33}$$

A Huber cost function, Equation 2.34, has quadratic cost for small errors and a linear cost for larger values. The Huber cost function is piece-wise continuous and the point at which the piece-wise functions change from a quadratic to linear cost can be determined using a parameter b.

$$C(e) = \begin{cases} e^2 & \text{for} \quad |e| < b \\ 2b|e| - b^2 & \text{for} \quad |e| \geq b \end{cases} \tag{2.34}$$

To further reduce the effects of outliers, one may wish to use a Cauchy cost function, Equation 2.35, which approximates a quadratic cost for small errors. Larger errors are penalised far less due to the logarithmic relationship. The parameter b is used to determine the point at which the cost-error relationship changes from being quadratic to logarithmic. The Cauchy cost function is continuous.

$$C(e) = 2b \log\left(1 + \frac{e^2}{b^2}\right) \tag{2.35}$$

A redescending cost function, Equation 2.36, can be used to remove the influence of outliers entirely by no longer increasing the cost after a certain threshold. The redescending cost function also has quadratic and linear regions and is piece-wise continuous. Parameters a and b can be used to determine the range for these regions, and c is used to determine the threshold after which the cost no longer increases.

$$C(e) = \begin{cases} \frac{e^2}{2} & \text{for} \quad |e| < a \\ a|e| - \frac{a^2}{2} & \text{for} \quad |e| \geq a, \quad |e| < b \\ ab - \frac{a^2}{2} + \frac{a(c-b)^2}{2}\left(1 - \left(\frac{c-|e|}{c-b}\right)^2\right) & \text{for} \quad |e| \geq b, \quad |e| < c \\ ab - \frac{a^2}{2} + \frac{a(c-b)^2}{2} & \text{for} \quad |e| \geq c \end{cases} \tag{2.36}$$

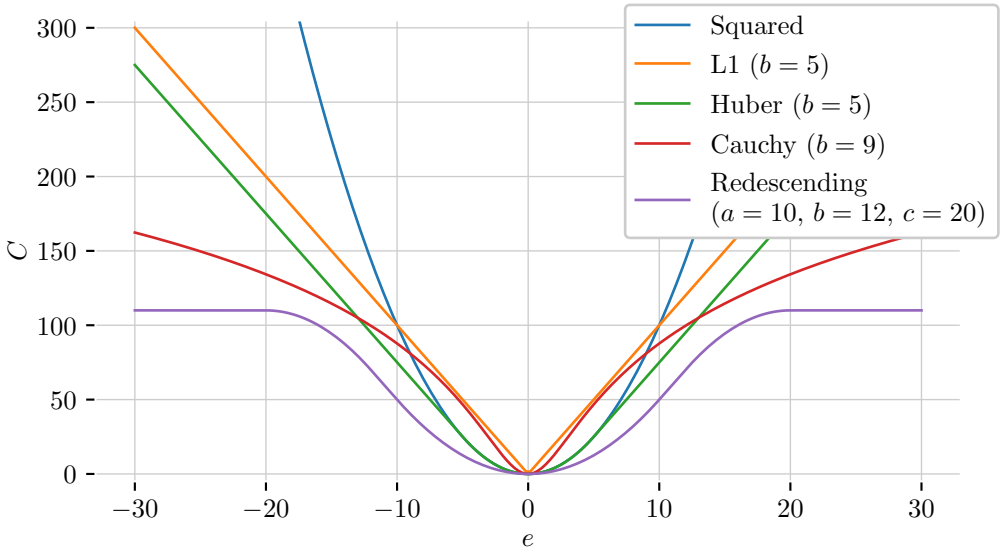

Figure 2.12: A comparative plot of cost functions

2.6 The Kalman Filter

The Kalman filter, named after Rudolf Emil Kálmán [46], is an optimal linear estimation algorithm used in areas such as control systems, signal processing, time-series analysis, trajectory optimisation and navigation to name a few. Kalman filters allow one to produce accurate predictions of a system's state through the observation of measurements containing statistical noise or errors. In this regard, Kalman filters are also used in sensor fusion. The main breakthrough that the Kalman filter is known for is that it does not rely on a history of observations or estimates and that it allows for the efficient computation of the estimated system states.

The discrete Kalman filter algorithm is comprised of two main processes known as the *prediction* and *update* steps. These steps are usually performed recursively at regular time intervals, shown by Figure 2.13.

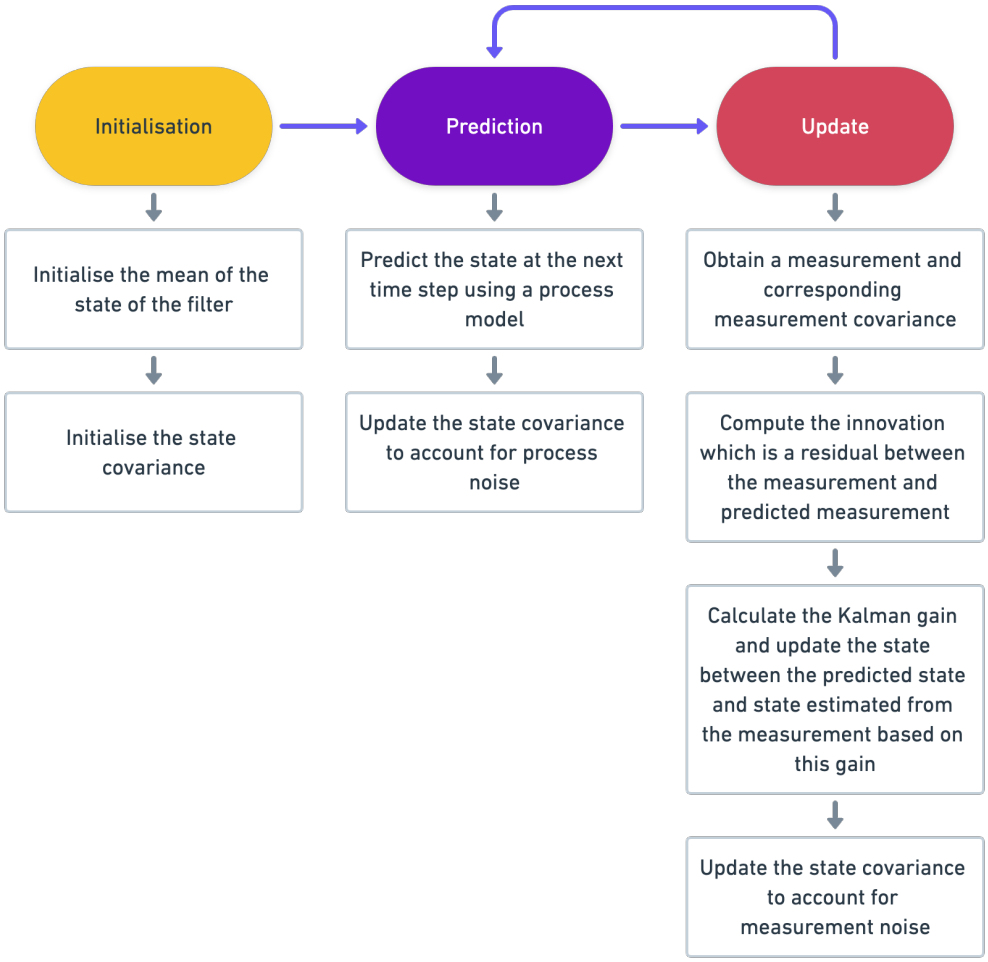

Figure 2.13: Flowchart of the Kalman filter algorithm

Kalman filters are used as a tool to fuse measurements from sensors to predict the state of a system and are used extensively in computer vision for problems like object tracking. In order to use a Kalman filter, one needs to create a model of the system in terms of the Kalman filter equations. A derivation of the Kalman filter follows, drawing from [47] and [48]. R. Labbe's Jupyter Notebooks [48] provide an invaluable resource for learning about the Kalman filter from an implementation perspective.

The plot below in Figure 2.14 demonstrates a single state Kalman filter estimating the *state* and the *state covariance*, which is effectively how accurate the filter believes its state estimate is. In the plot, you can see that the filter's state estimate is always situated in between a *state prediction* and a

measurement. How much the filter favours the measurement or the predicted estimate is determined by the *Kalman gain*. It is also worth noting how the filter is initialised to have a large state covariance, but this converges very quickly once measurements are available.

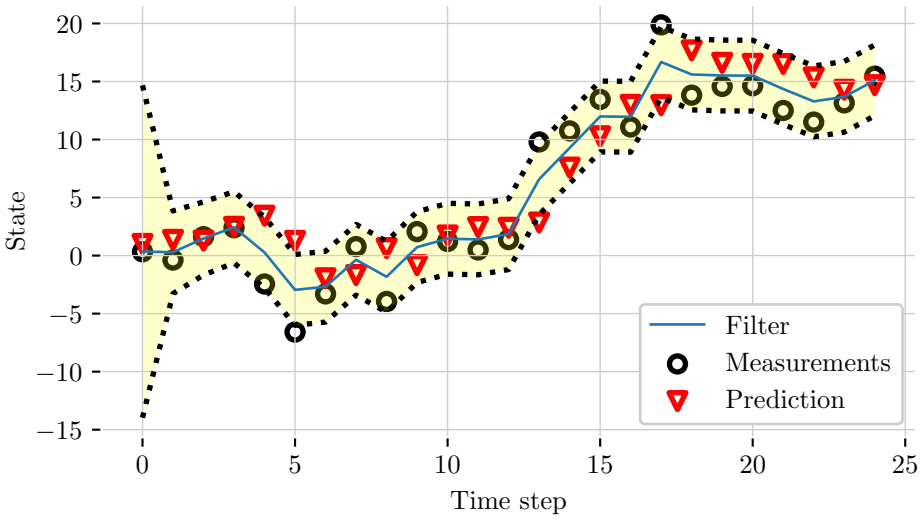

Figure 2.14: Plot of a single state Kalman filter estimating the mean of the state (blue line) and state covariance (yellow band) given a set of measurements of the state

2.6.1 Process Model

The standard Kalman filter assumes that the system dynamics we wish to model are linear. First, a *state-space* representation of the system is usually derived in the form of Equation 2.37 where $\mathbf{x}$ is state vector of the process being modelled. $\mathbf{A}$ is the system dynamics matrix and $\mathbf{B}$ is the control input model which defines a linear relationship between a control input $\mathbf{u}$ and the change in system state. $\mathbf{w}$ accounts for noise in the system.

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u} + \mathbf{w} \tag{2.37}$$

A recursive or discrete form of the state-space system, Equation 2.37, can be written as Equation 2.38 where $\mathbf{x}_k$ is the *state vector* of a process and $\mathbf{F}_k$ is the *state transition model*. A measurement $\mathbf{z}_k$ is related to the state $\mathbf{x}_k$ via a measurement model $\mathbf{H}_k$, which maps the state from state space to measurement space.

$$\mathbf{x}_k = \mathbf{F}_k\mathbf{x}_{k-1} + \mathbf{B}_k\mathbf{u}_k + \mathbf{w}_k \tag{2.38}$$

$$\mathbf{z}_k = \mathbf{H}_k \mathbf{x}_k + \mathbf{v}_k \tag{2.39}$$

The *process noise* $\mathbf{w}_k$ and *measurement noise* $\mathbf{v}_k$ are random variables with a multivariate Gaussian probability distribution having covariance $\mathbf{Q}_k$ and $\mathbf{R}_k$ respectively.

$$\mathbf{w}_k \sim \mathcal{N}(0, \mathbf{Q}_k^2) \tag{2.40}$$

$$\mathbf{v}_k \sim \mathcal{N}(0, \mathbf{R}_k^2) \tag{2.41}$$

2.6.2 Prediction and Update Steps

A Kalman filter's state is determined by a state estimate $\hat{\mathbf{x}}$ and corresponding state covariance $\mathbf{P}$. In this section, the notation $\hat{\mathbf{x}}_{n|m}$ will be used to represent an estimate of $\mathbf{x}$ at time n given measurements up until time m where $m \leq n$.

In the prediction step, an a priori estimate of the state $\hat{\mathbf{x}}_{k|k-1}$ and state covariance $\mathbf{P}_{k|k-1}$ are obtained using the a posteriori estimates from the previous time step. In the update step, a measurement is made and a posteriori estimates of the state, $\hat{\mathbf{x}}_{k|k}$, and state covariance, $\mathbf{P}_{k|k}$, are obtained for the current time step. The a posteriori estimates from the update step are then used in the prediction step at the next time step. The prediction and update steps are performed cyclically.

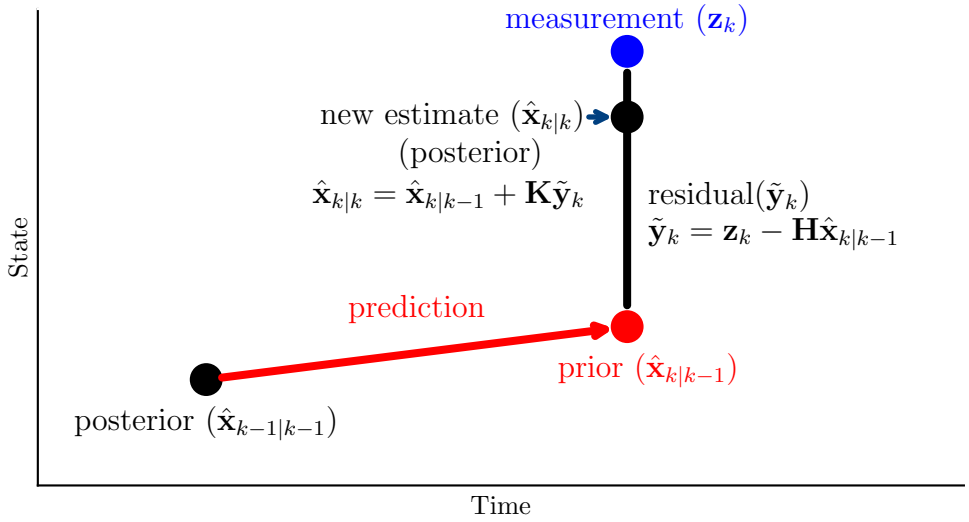

Figure 2.15: Plot showing the state prediction and update steps where the measurement is defined in the space of the states, adapted from [48]

Prediction

The *a priori* estimates of the state and state covariance at time k are calculated using the a posteriori estimate at time $k-1$ using Equation 2.42 and Equation 2.43.

$$\hat{\mathbf{x}}_{k|k-1} = \mathbf{F}_k \hat{\mathbf{x}}_{k-1|k-1} + \mathbf{B}_k \mathbf{u}_k \tag{2.42}$$

$$\mathbf{P}_{k|k-1} = \mathbf{F}_k \mathbf{P}_{k-1|k-1} \mathbf{F}_k^T + \mathbf{Q}_k \tag{2.43}$$

Update

The update step can be split into four stages. Firstly the innovation $\tilde{\mathbf{y}}_k$ is calculated using Equation 2.44, which is a residual between a measurement, $\mathbf{z}_k$, and an estimated measurement based on the a priori state. Similarly, the innovation covariance, $\mathbf{S}_k$ is calculated using Equation 2.45.

$$\tilde{\mathbf{y}}_k = \mathbf{z}_k - \mathbf{H}_k \hat{\mathbf{x}}_{k|k-1} \tag{2.44}$$

$$\mathbf{S}_k = \mathbf{H}_k \mathbf{P}_{k|k-1} \mathbf{H}_k^T + \mathbf{R}_k \tag{2.45}$$

The next stage is to calculate the Kalman gain, $\mathbf{K}_k$, based on the a priori state covariance and innovation covariance according to Equation 2.46.

$$\mathbf{K}_k = \mathbf{P}_{k|k-1} \mathbf{H}_k^T \mathbf{S}_k^{-1} \tag{2.46}$$

Using the Kalman gain, the a posteriori state estimate and state covariance can be computed.

$$\hat{\mathbf{x}}_{k|k} = \hat{\mathbf{x}}_{k|k-1} + \mathbf{K}_k \tilde{\mathbf{y}}_k \tag{2.47}$$

$$\mathbf{P}_{k|k} = (\mathbf{I} - \mathbf{K}_k \mathbf{H}_k) \mathbf{P}_{k|k-1} \tag{2.48}$$

The post-update residual can be computed for the measurement according to Equation 2.49.

$$\tilde{\mathbf{y}}_{k|k} = \mathbf{z}_k - \mathbf{H}_k \hat{\mathbf{x}}_{k|k} \tag{2.49}$$

2.6.3 The Extended Kalman Filter

As noted in the previous section, the Kalman filter is an optimal linear estimator; however, most real-world systems are non-linear in the sense that their process and/or measurement models do not map linearly to the system states. The extended Kalman filter (EKF) adapts the Kalman filter by using linear approximations about the filter's current state and state covariance. Other methods such as the unscented Kalman filter (UKF) [49] and

particle filter [50] are also used as non-linear estimators. Unlike the Kalman filter, the EKF is not a linear estimator unless the state transition and the measurement model are linear.

The EKF makes changes to the state transition and measurement models representing them instead by differentiable functions.

$$\mathbf{x}_k = \mathbf{f}(\mathbf{x}_{k-1}, \mathbf{u}_k) + \mathbf{w}_k \tag{2.50}$$

$$\mathbf{z}_k = \mathbf{h}(\mathbf{x}_k) + \mathbf{v}_k \tag{2.51}$$

The state prediction and innovation equations are changed to reflect the use of the non-linear functions.

$$\hat{\mathbf{x}}_{k|k-1} = \mathbf{f}(\hat{\mathbf{x}}_{k-1|k-1}, \mathbf{u}_k) \tag{2.52}$$

$$\tilde{\mathbf{y}}_k = \mathbf{z}_k - \mathbf{h}_k(\hat{\mathbf{x}}_{k|k-1}) \tag{2.53}$$

The state transition and measurement model matrices are defined as the Jacobians evaluated at the current state.

$$\mathbf{F}_k = \frac{\partial \mathbf{f}}{\partial \mathbf{x}}\Big|_{\hat{\mathbf{x}}_{k-1|k-1}, \mathbf{u}_k} \tag{2.54}$$

$$\mathbf{H}_k = \frac{\partial \mathbf{h}}{\partial \mathbf{x}}\Big|_{\hat{\mathbf{x}}_{k|k-1}} \tag{2.55}$$

2.6.4 Kalman Smoothing

Kalman smoothing is an algorithm which uses measurements, $\mathbf{z}_1 \dots \mathbf{z}_n$, to compute a maximum likelihood state estimate measurement, $\hat{\mathbf{x}}_{k|n}$, where $k < n$. The Kalman smoother is a two-pass algorithm consisting of a forward and backward pass. There are several Kalman smoother algorithms [51], [52], [53] which store data from a forward pass and use this data in a backwards pass to compute optimal state estimates. The forward pass consists of a standard Kalman filter algorithm sequentially estimating states from the measurements $\mathbf{z}_1 \dots \mathbf{z}_n$.

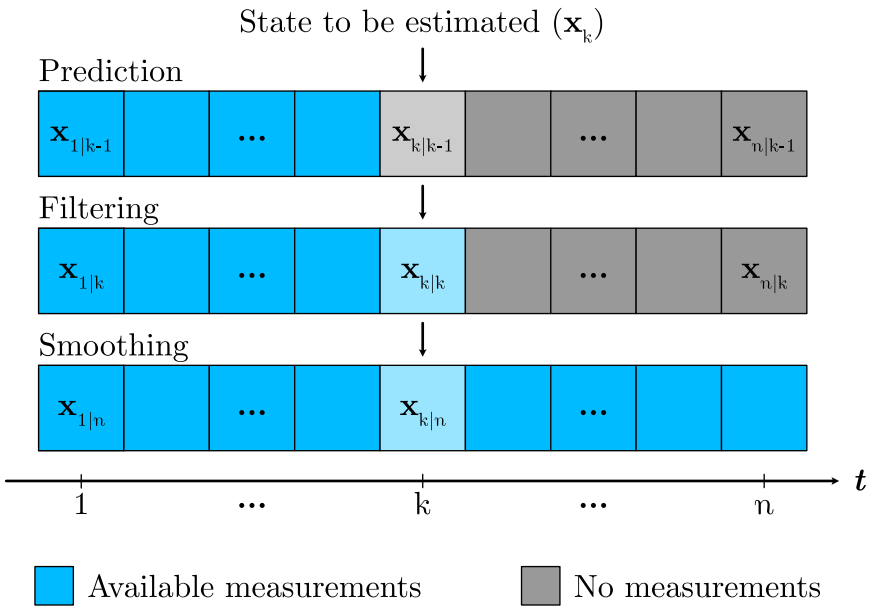

Figure 2.16: Diagram showing the difference between prediction, filtering and smoothing

The Rauch-Tung-Striebel Kalman smoother [51] is particularly simple to implement and works by storing the a priori and a posteriori state estimates, $\hat{\mathbf{x}}_{k|k}$ and $\hat{\mathbf{x}}_{k|k-1}$, and a priori and a posteriori covariances, $\mathbf{P}_{k|k}$ and $\mathbf{P}_{k|k-1}$, at each time instance in the forward pass. In the backwards pass, a matrix $\mathbf{A}$ is constructed from the a priori and a posteriori covariance matrices, $\mathbf{P}_{k|k}$ and $\mathbf{P}_{k+1|k}$.

$$\mathbf{A}_k = \mathbf{P}_{k|k}\mathbf{F}_{k+1}^T\mathbf{P}_{k+1|k}^{-1} \tag{2.56}$$

Using $\mathbf{A}$, the smoothed state estimates and covariances are obtained through Equation 2.57 and Equation 2.58.

$$\hat{\mathbf{x}}_{k|n} = \hat{\mathbf{x}}_{k|k} + \mathbf{A}_k(\hat{\mathbf{x}}_{k+1|n} - \hat{\mathbf{x}}_{k+1|k}) \tag{2.57}$$

$$\mathbf{P}_{k|n} = \mathbf{P}_{k|k} + \mathbf{A}_k(\mathbf{P}_{k+1|n} - \mathbf{P}_{k+1|k})\mathbf{A}_k^T \tag{2.58}$$

2.7 Optimisation-based Estimation

While the EKF provides a computationally inexpensive method for non-linear state estimation, it lacks the ability to place constraints on the states of a system. Moving horizon estimation (MHE) provides a method for non-linear state estimation by using a fixed "window" of historical measurements to compute a maximum a posteriori (MAP) estimate of the state. Rao et al. [54] show that since MHE is optimisation based, inequality constraints on the state variables can be imposed. Rao et al. [54] also note that there is increased computational complexity when compared to the EKF.

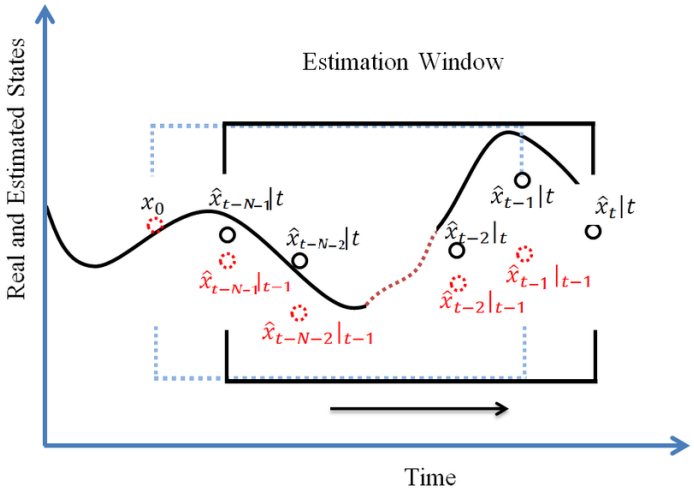

Figure 2.17: Depiction of moving horizon estimation [55]

In the case of human pose estimation using inertial measurement units (IMUs), Kok et al. [56] proposed an optimisation-based method to incorporate valuable information from biomechanical constraints. This method also involves computing the MAP estimate for a set of measurements given a dynamic model of a human; however, unlike MHE, this was computed over the entire set of measurements.

Xinjilefu et al. [57] use optimisation as their approach for dynamic state estimation (of Boston Dynamics' Atlas robot), which was solved using quadratic programming. They stated that one of the main advantages of using the optimisation approach instead of a Kalman filter was the ability to naturally constrain states using inequality and equality constraints.

Mueller et al. [58] make use of a gradient descent based optimisation approach to perform kinematic skeletal fitting where they estimate the 3D

skeletal pose of a hand from video footage. They attempt to minimise a cost function consisting of four "error" terms, which are the errors from fitting 2D joint positions from video, fitting of a 3D hand model, joint angle constraints, and a temporal smoothness term.

2.8 Summary

From the literature discussed above, there are several key takeaways. While various motion capture techniques exist, we see that markerless motion capture is increasing in popularity due to its non-invasive nature and the recently increased accuracy afforded to it by the advancements in CNNs. It has been shown that CNNs allow for 2D pose estimation in outdoor environments which poses a problem to techniques which make use of IR markers, projectors or cameras.

In order to use a camera-based approach, we see that it is necessary to model the camera's characteristics. It is shown that there are two main sets of parameters which describe a camera, namely the intrinsic and extrinsic parameters. Several tools are available which can be used to calibrate cameras through the use of a calibration object. Two examples of which are OpenCV and the MATLAB camera calibration toolbox. As a final stage of extrinsic camera calibration, SBA is commonly used to obtain the optimal camera parameters when multiple cameras are used.

It was seen that sparse bundle adjustment is the de facto method for reconstruction of 3D points from 2D point correspondences (when not using a model). If outliers are present when performing an SBA, the results of the optimisation may be skewed. Robust cost functions can be used to prevent or limit outliers from skewing the estimated parameters and 3D point estimates.

Model-based techniques like the Kalman filter provide maximum likelihood state estimates for the model in the presence of noise. Due to the non-linear measurement functions which are used in 3D reconstruction, the use of an EKF is needed. As an additional step, a Kalman smoother can be used to obtain optimal estimates for a full trajectory. Unfortunately, Kalman filters also suffer from outliers. Another downside of using a Kalman filter is that it is not easy to enforce constraints on the states of a model. In this case that it is necessary to constrain the states of the model, optimisation based techniques, including MHE and MAP estimation, may be used allowing for the states to be bounded by inequality constraints.

Chapter 3

Methodology

3.1 Project Plan

In exploring the various methods for 3D reconstruction, the planning, development and evaluation stages make up essential parts of the project. This section aims to provide a detailed understanding of the problem which this project aims to solve and in doing so, sets out the requirements, constraints and scope of the project.

3.1.1 Problem Definition

This project aims to implement and evaluate various methods which will facilitate the extraction of 3D motion data from multi-camera video footage of cheetahs. Specifically, we aim to construct a 3D rigid-body skeletal approximation of a cheetah from 2D marker positions obtained from video footage. DeepLabCut [8] will be used to obtain these 2D marker position estimates. The implementations need to account for errors in the data, including inaccurate and missing measurements, to ensure a robust 3D reconstruction of the cheetah's locomotion. While cheetahs will be the subjects studied during this project, the methods developed should apply to the study of locomotion for other subjects.

3.1.2 Project Requirements

Requirements for this project originated from the initial project brief and have evolved throughout the project. Project requirements were adapted to explore different approaches to the 3D reconstruction problem and to limit

the project scope. The resulting list of requirements was:

1. Implement and evaluate several software-based methods to reconstruct the 3D locomotion of cheetahs from multi-camera video footage from stationary cameras. This includes the ability to:
   (a) calibrate intrinsic camera parameters using a calibration object,
   (b) calibrate external camera parameters for multiple cameras in a scene using a calibration object,
   (c) reconstruct 3D point estimates for corresponding 2D points seen by two or more cameras,
   (d) within reason, robustly handle incorrect and missing measurements
2. Evaluate different 3D point, model and motion estimation methods. This includes:
   (a) choosing appropriate metrics for 3D point estimation and motion estimation,
   (b) choosing a specific set of data to use for comparison throughout the project,
   (c) establishing ground-truth measurements as a baseline for comparison,
3. Aid in the analysis of the data by:
   (a) appropriately recording and exporting the 3D position and locomotion data for use in tools such as Microsoft Excel, Python and MATLAB,
   (b) visualising 3D position and locomotion data through the use of graphs, charts and interactive graphics,

3.2 Design Approach

The approach taken in planning and developing the system was to divide the project into separate tasks. Goals were also then associated with each task. It was evident from the planning that the two main components of the system were calibration and 3D estimation.

For the calibration, it was necessary to calibrate both the intrinsic and extrinsic parameters of the cameras. A camera's intrinsic parameters only need to be obtained once as these do not change unless factors affecting the resolution and field of view change such as changing lenses or mechanically changing the focus of the camera. Extrinsic parameters, which describe the position and orientation of the cameras, need to be calculated whenever cameras are moved. Each set of extrinsic parameters is unique to a stationary camera arrangement. From here onwards, these stationary camera arrangements will be referred to as scenes.

Various methods presented themselves for the task of 3D estimation, all based around some form of optimisation. Initially, deep learning approaches to this task were also considered; however, they were decided against due to the large amounts of data required. In the end, three approaches were decided upon for the task of estimation:

1. a 3D point reconstruction using a sparse bundle adjustment (SBA) routine,
2. a model-based trajectory estimation using an extended Kalman filter (EKF) and Kalman smoother, and,
3. a model-based, simultaneous full-trajectory estimation using a non-linear interior-point optimiser.

All three estimation methods were responsible for performing the same task of estimating the 3D point estimates of locations on the cheetah. They also worked off the same inputs of 2D marker locations from DeepLabCut.

Before describing the system components further, an overview of the system is provided in Figure 3.1.

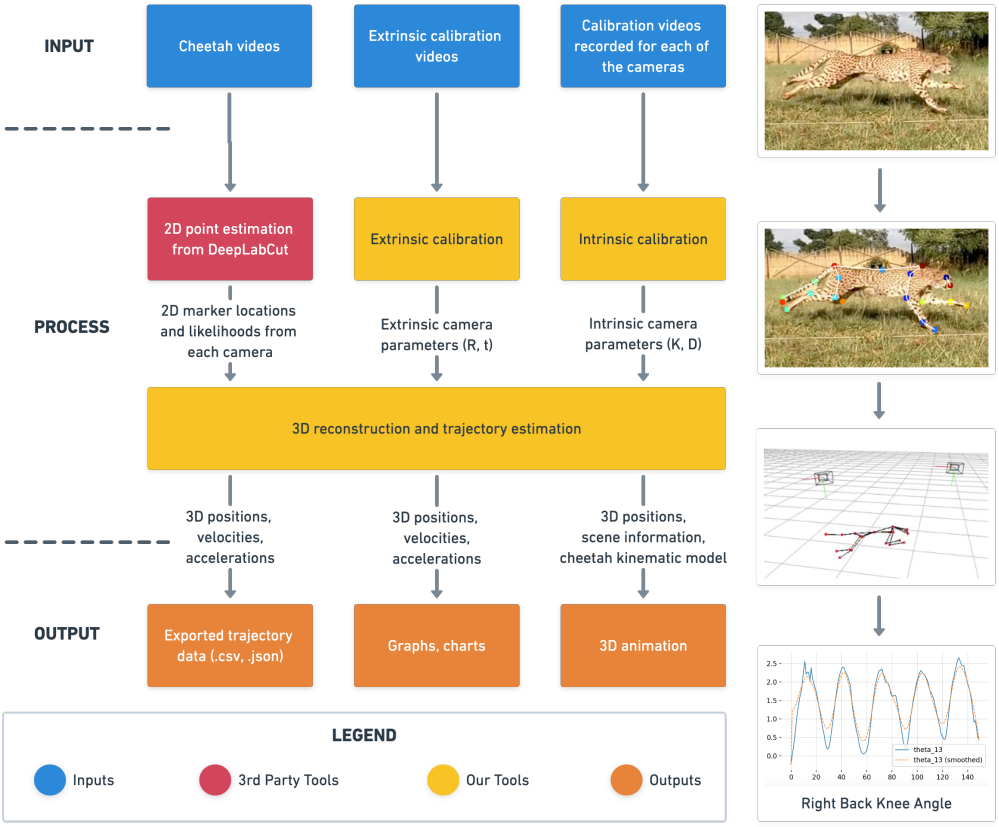

Figure 3.1: A diagram showing the system components and flow of data through the system

Work breakdown structures (WBS) were created to illustrate the tasks required for the calibration and 3D reconstruction components of the system. The diagrams provided high-level descriptions of task outcomes resulting in an outline of the workflow for each section.

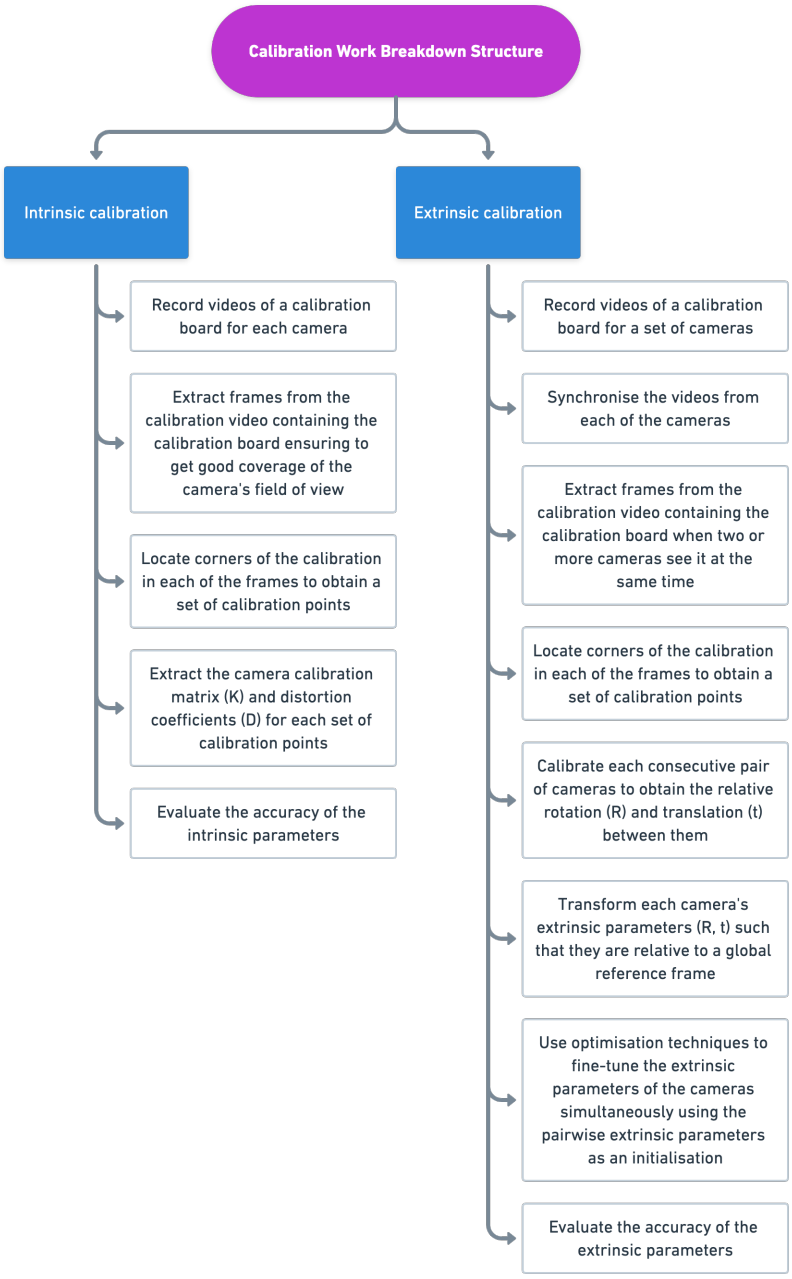

Figure 3.2: A diagram showing the WBS for camera calibration tasks

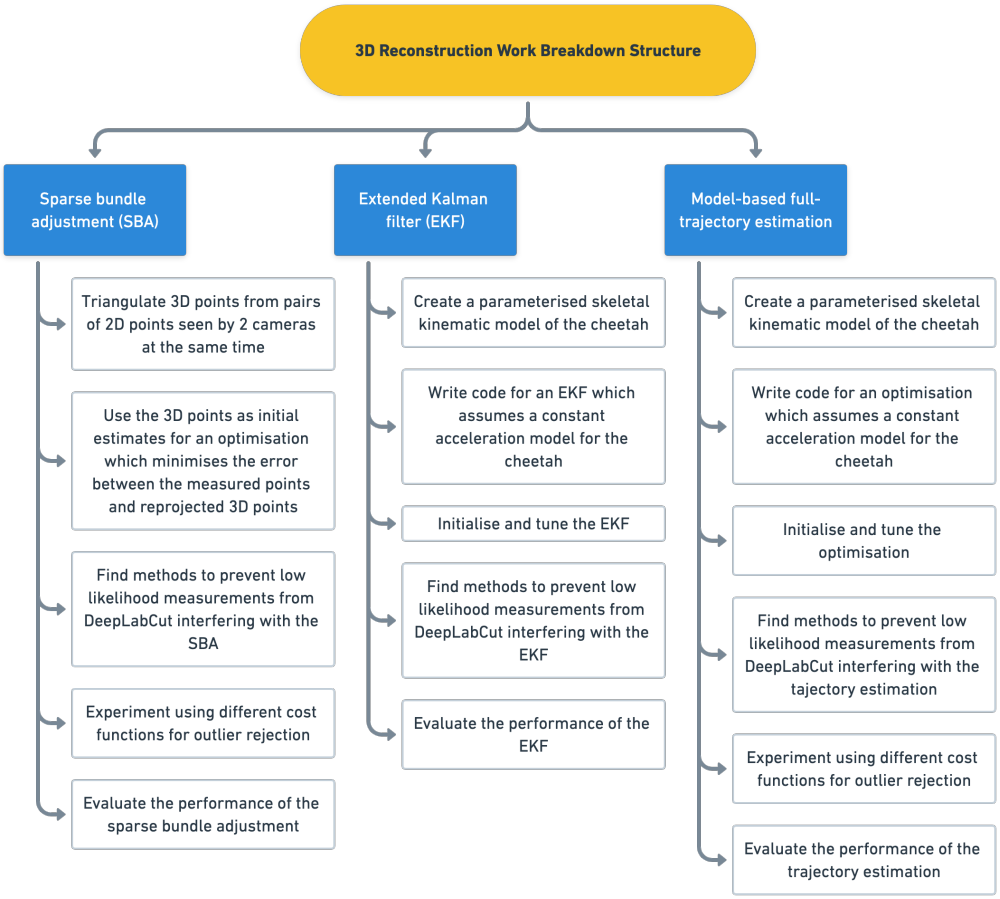

Figure 3.3: A diagram showing the WBS for 3D reconstruction tasks

3.3 Data

Various videos and sets of data used in the project are available at `this link`. Additionally, a video describing an overview of the project is also available at the same link.

3.3.1 Cheetah Videos

Two multi-view videos of cheetahs were selected for this project. The footage was captured from 6 GoPro Hero 7 Black cameras. Videos from each camera were recorded at a resolution of 2704×1520 at 120 frames per second (fps). Each video was captured on the GoPro's wide field of view (FOV) setting

providing a FOV of 118.2° in the horizontal direction and 69.5° in the vertical direction.

At the cheetah sanctuaries, cheetahs are exercised by enticing them to chase a lure. This lure is attached to a motorised winch system which pulls the lure around a track. In order to film the cheetahs, three cameras were placed on either side of the track facing inwards such that the cheetah would be visible in all the camera frames at roughly the same time.

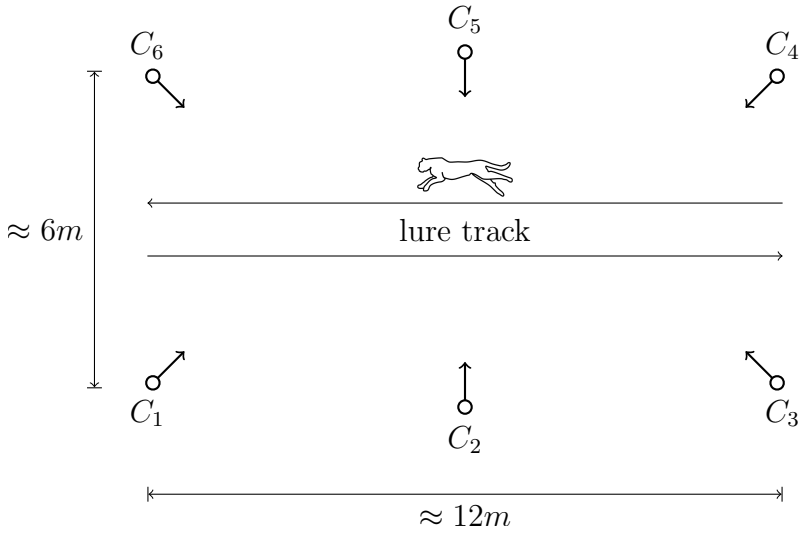

Figure 3.4: Camera layout for recording the cheetahs at the cheetah sanctuary

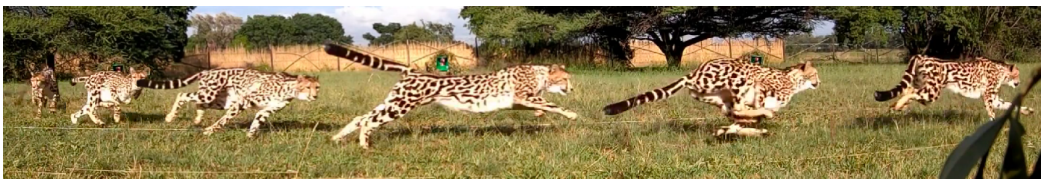

Figure 3.5: Overlaid frames from a video of a cheetah run

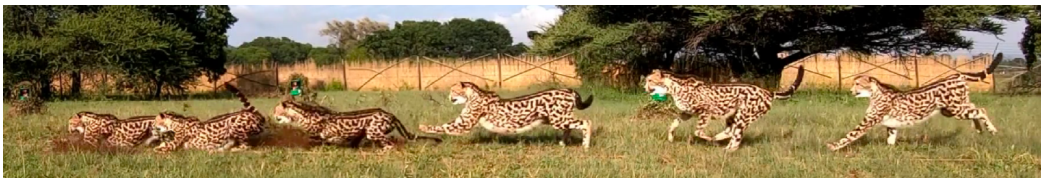

Figure 3.6: Overlaid frames from a video of a cheetah dive

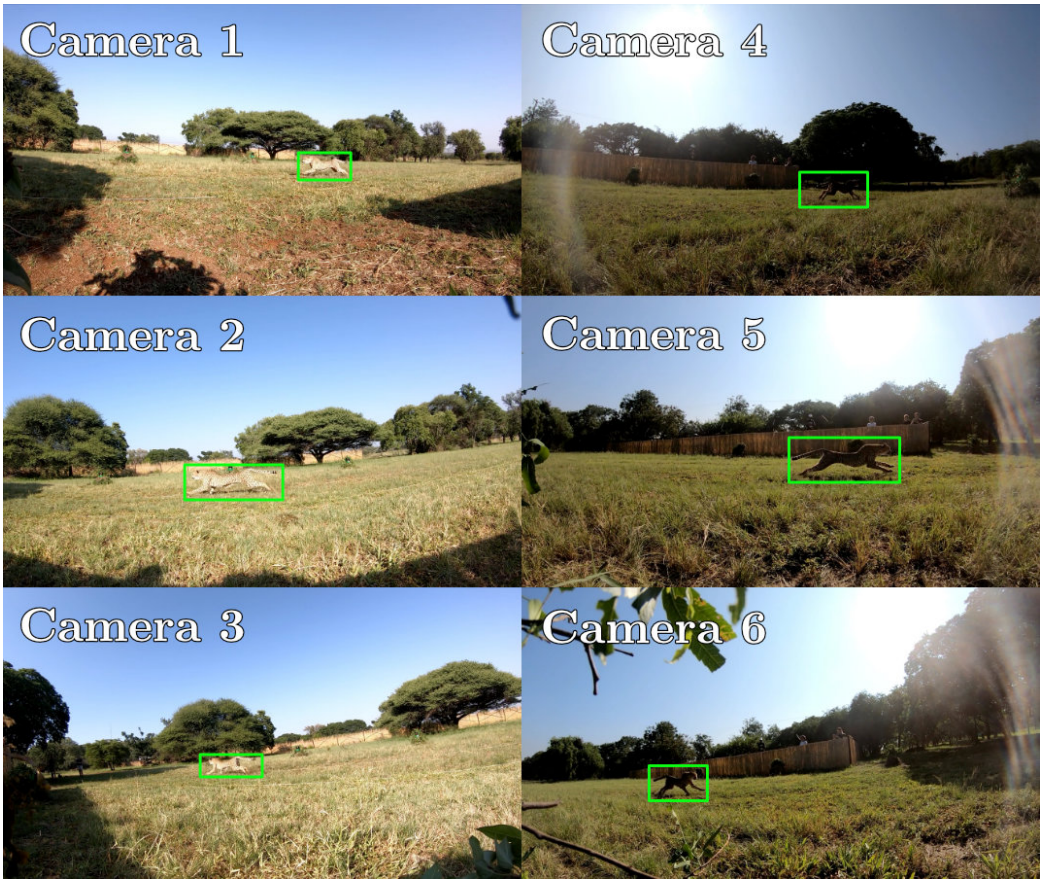

Figure 3.7: A cheetah viewed from all 6 cameras

The data used for all of the motion estimation techniques in this project relied on 2D points estimates of positions on the cheetah. These positions are referred to as markers or labels. Table 3.1 lists and describes the markers that were obtained from DeepLabCut, which were used for motion estimation.

Index	Marker Name	Description
0	nose	The tip of the nose
1	l_eye	Centre of the left eye
2	r_eye	Centre of the right eye
3	neck_base	Base of the neck, just above shoulders
4	spine	Centre of the spine
5	tail_base	Base of the tail.
6	tail1	Half way between the base and tip of the tail
7	tail2	The tip of the tail
8	l_shoulder	Top of the bulge caused by the shoulder blade
9	l_front_knee	Centre of rotation of the knee
10	l_front_ankle	Centre of rotation of the ankle
11	l_paw	Centre of the paw
12	r_shoulder	Top of the bulge caused by the shoulder blade
13	r_front_knee	Centre of rotation of the knee
14	r_front_ankle	Centre of rotation of the ankle
15	r_paw	Centre of the paw
16	l_hip	Centre of the visible bulge at the hip
17	l_back_knee	Centre of rotation of the knee
18	l_back_ankle	Centre of rotation of the ankle
19	l_back_paw	Centre of the paw
20	l_hip	Centre of the visible bulge at the hip
21	l_back_knee	Centre of rotation of the knee
22	l_back_ankle	Centre of rotation of the ankle
23	l_back_paw	Centre of the paw

Table 3.1: A table showing the joint information obtained from DeepLabCut

For a marker in any given image, DeepLabCut produced a pixel coordinate prediction, (u, v), accompanied by a likelihood value in the range $[0, 1]$ which represents DeepLabCut's faith in the prediction. For each video of the cheetahs, marker coordinates and likelihoods were obtained for each frame. The output from DeepLabCut was stored in both Comma Separated Value (CSV) and Hierarchical Data Format (HDF) files. An example of the markers being plotted on an image of a cheetah is shown below in Figure 3.8.

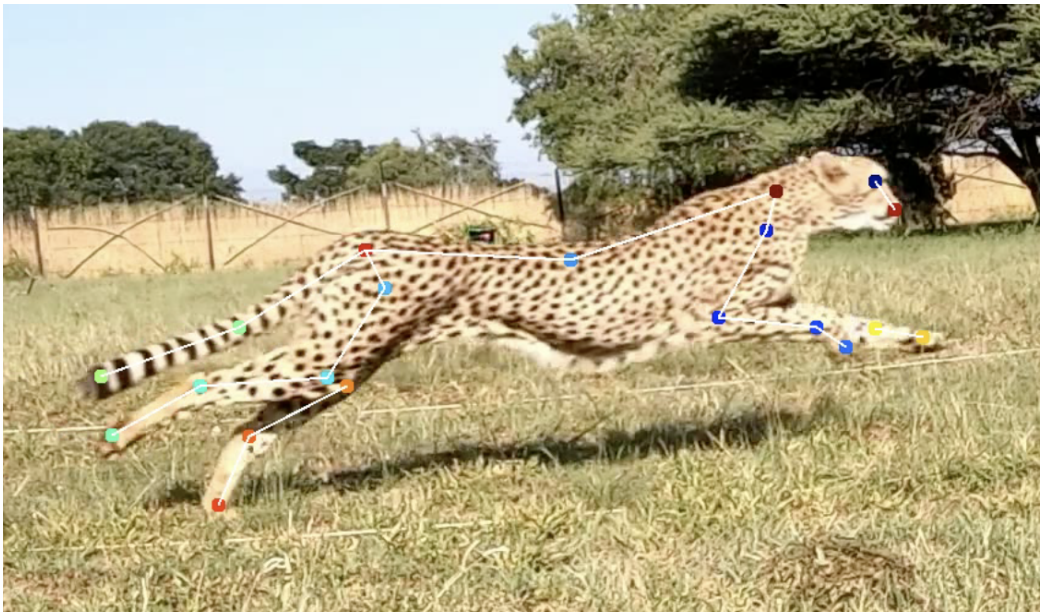

Figure 3.8: DeepLabCut markers with likelihood ≥ 0.5 plotted over the frame they were obtained from

Frame	Markers						⋯
	nose			l_eye			⋯
	u	v	likelihood	u	v	likelihood	⋯
1	250	300	0.85	245	305	0.95	
2	265	302	0.87	261	307	0.99	
⋮							

Table 3.2: A example showing the type of data obtained from DeepLabCut

In order to estimate the 3D position for a marker, 2D point correspondences were required from the different camera views. This required the captured videos to be in sync with one another. To synchronise the videos, a light was flashed in the view of all the cameras shortly after starting to record. The videos were synchronised to this flash and subsequently clipped into short videos for each cheetah run.

3.3.2 DeepLabCut

The data for DeepLabCut consisted of 8522 human labelled images of cheetahs, containing a label for each marker listed in Table 3.1. 95% of the data

was used for training, leaving 5% to be used for testing.

The images used for labelling spanned ten different cheetahs and were captured on GoPro Hero 7 cameras at a pixel resolution of 2704×1520 and GoPro Hero 5 Session cameras at a pixel resolution of 1920 × 1080. The network used for training was based on a pre-trained version of ResNet50 and was trained for a total of 1030000 iterations. DeepLabCut was also configured to use pairwise correspondence to improve the accuracy of the predictions. After training, the network was evaluated for predictions with a likelihood greater than 0.5.

Dataset	Mean (Pixels)	Median (Pixels)	Std. Deviation (Pixels)	Median Absolute Deviation (Pixels)
Train	0.47	1.12	59.88	4.60
Test	1.43	1.12	68.82	5.04

Table 3.3: DeepLabCut training and test error statistics

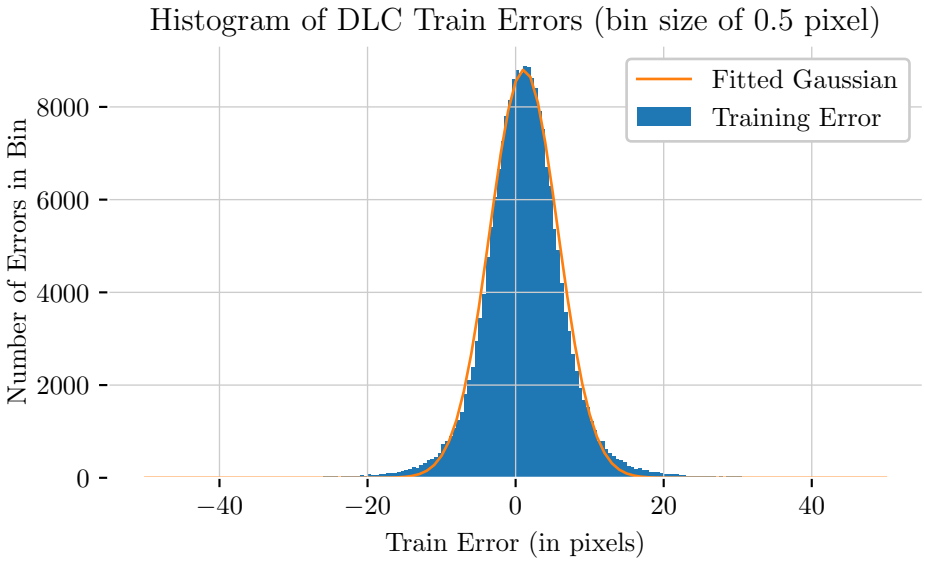

Figure 3.9: Histogram of the cheetah data training error from DLC

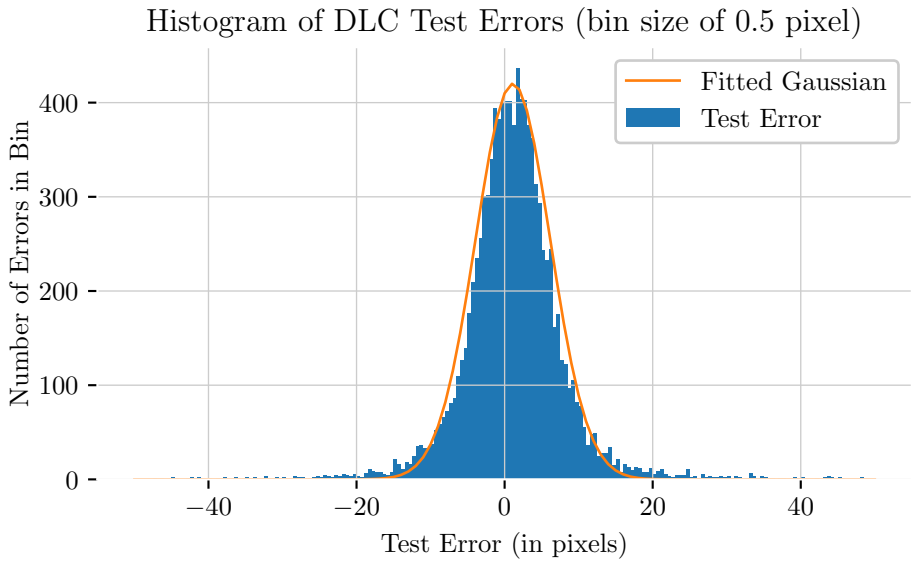

Figure 3.10: Histogram of the cheetah data test error from DLC

From Figure 3.9 and Figure 3.10, it can be seen that the error distribution resembles a Gaussian distribution; however, outliers skew the data (some of which have an error of over 2000 pixels) resulting in a high standard deviation, shown in Table 3.3, when compared to the fitted Gaussian. This made it challenging to select a statistical estimate for outliers without visualising the data.

A rule of thumb for a Gaussian distributed dataset is that outliers are determined as points which deviate from the mean by more than three standard deviations (99.7% of the data lying within this region). However, it is evident from the data that this would not have been a reasonable assumption when using the standard deviation of the dataset. In the case of the test errors from DeepLabCut, it was decided that the mean (1.12 pixels), and standard deviation (5.04 pixels) from the fitted Gaussian be used as metrics for accuracy. Therefore, outliers could roughly be determined as points deviating by more than 15 pixels from an expected value.

3.3.3 Simulated Data

Overview

In order to evaluate each of the trajectory estimation techniques, ground truth data was needed, which did not include errors from DeepLabCut, cam-

era calibration or human labelling. In order to do this, a cheetah run was simulated by defining a 3D pose, corresponding to marker positions, for a cheetah as a function of pose parameters. A 3D trajectory could then created by adjusting the pose parameters at each frame to create a realistic-looking cheetah run.

Using a real set of camera intrinsic and extrinsic parameters, the 3D points from the trajectory could then be projected into the view of each camera, thus giving us a set of 2D point correspondences with corresponding 3D ground truth positions.

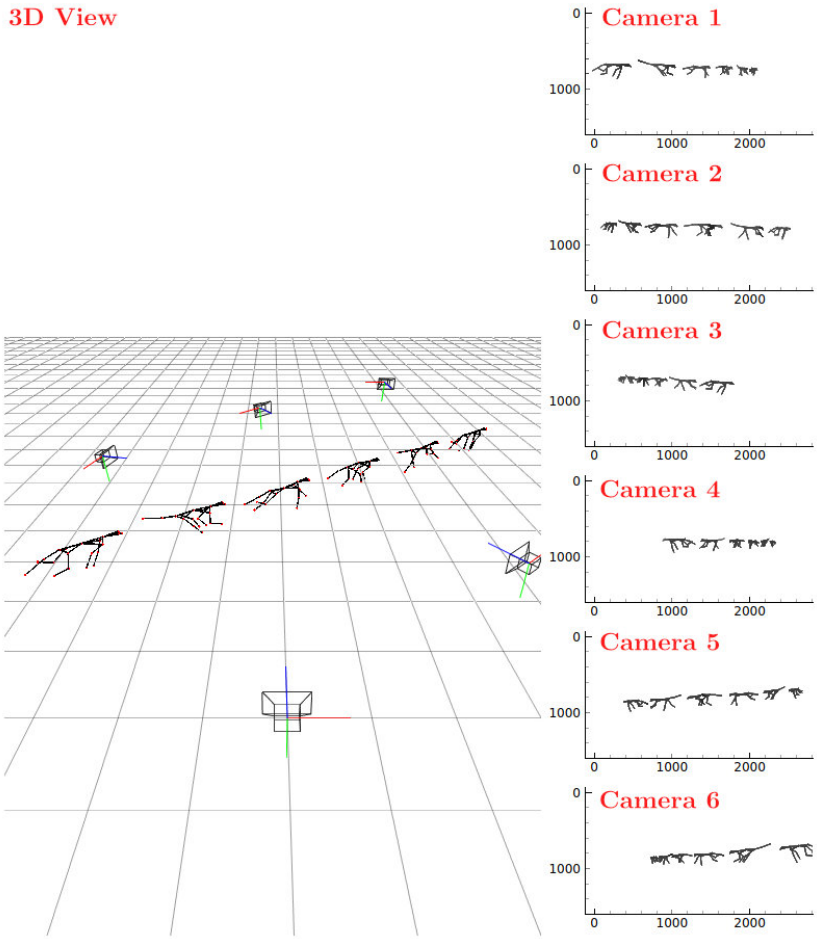

Figure 3.11: Screenshot of a python app for viewing pose data showing select frames from a simulated cheetah run including 3D positions and the 2D camera projections

Simulated Dataset for Evaluation

It was decided that several datasets be used for testing and evaluating the different trajectory estimation methods; specifically, datasets which allowed for the evaluation of these methods in the presence of noise and outliers in the measurement data (the 2D points seen by the cameras). Varying degrees of Gaussian noise was added to the ground truth 2D points. The noise n for each 2D coordinate was randomly drawn from a Gaussian distribution with 0 mean and standard deviation σ_n.

$$n \sim \mathcal{N}(0, \sigma_n^2) \tag{3.1}$$

Outliers were created by adding an "outlier value" o to a randomly selected subset of the ground truth 2D points. The probability of a 2D point being turned into an outlier was determined by p_o. The outlier value o for each 2D coordinate was randomly drawn from a Gaussian distribution with 0 mean and standard deviation σ_o.

$$o \sim \mathcal{N}(0, \sigma_o^2) \tag{3.2}$$

Each modified simulated 2D point coordinate c_{noisy} was obtained using Equation 3.3 and Equation 3.4, where c is one of the (u, v) pixel coordinates.

$$c_{noisy} = c + n \quad \text{for noise only} \tag{3.3}$$

$$c_{noisy} = c + n + o \quad \text{for outliers (including noise)} \tag{3.4}$$

Dataset	σ_n (pixels)	p_o (%)	σ_o (pixels)
1	0	0	0
2	0	2	100
3	0	5	100
4	5	0	100
5	5	2	100
6	5	5	100
7	10	0	100
8	10	2	100
9	10	5	100

Table 3.4: Noise and outlier parameters for the simulated datasets

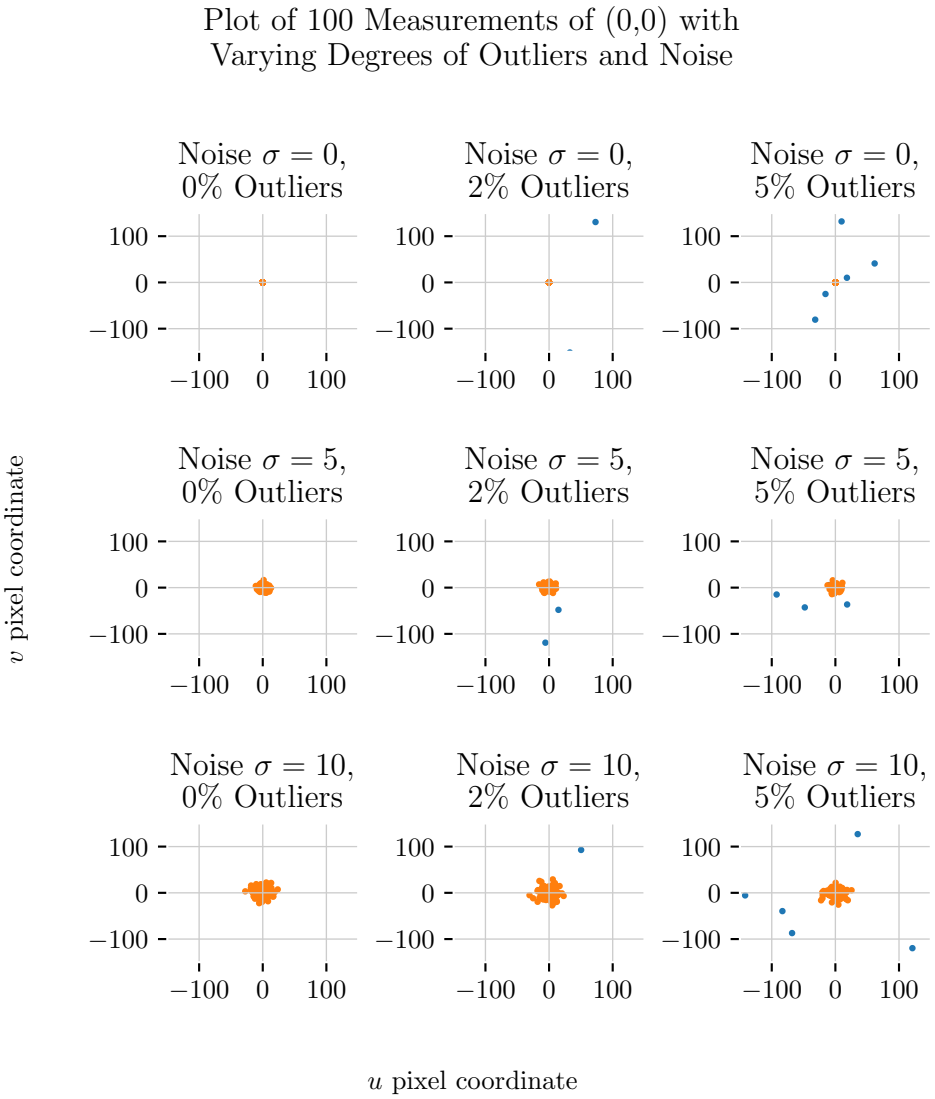

Figure 3.12: Plot showing the effect of 100 measurements of the point $(0, 0)$ with varying degrees of noise (in orange) and outliers (in blue)

3.3.4 Cheetah Pose Model

For the Extended Kalman Filter and the model-based trajectory estimation, the rigid body model of the cheetah shown in Figure 3.13 was used. This same model was also used for the generation of the simulated data.

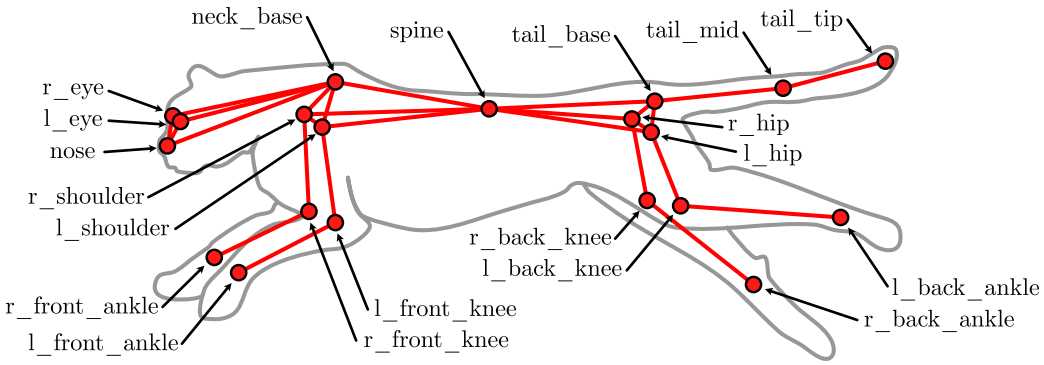

Figure 3.13: Cheetah model with markers corresponding to Table 3.7

A set of parameters, shown in Table 3.5, were used to determine the pose the cheetah model. The model was defined in terms of a global position and orientation of the cheetah and a set of angles describing the relative rotations between the rigid bodies in the model.

Pose parameters	Description
x, y, z	Head position in inertial frame
ϕ_1, θ_1, ψ_1	Head's roll, pitch and yaw in the inertial frame
ϕ_2, θ_2, ψ_2	Neck's roll, pitch and yaw relative to the head
θ_3	Front torso's pitch relative to the neck
θ_4, ψ_4	Back torso's pitch and yaw relative to the front torso
θ_5, ψ_5	Tail base's pitch and yaw relative to the back torso
θ_6, ψ_6	Middle of tail's pitch and yaw relative to the tail base
θ_7	Left shoulder's pitch relative to the front torso
θ_8	Left front knee's pitch relative to the left shoulder
θ_9	Right shoulder's pitch relative to the front torso
θ_{10}	Right front knee's pitch relative to the right shoulder
θ_{11}	Left hip's pitch relative to the back torso
θ_{12}	Left back knee's pitch relative to the left hip
θ_{13}	Right hip's pitch relative to the back torso
θ_{14}	Right back knee's pitch relative to the right hip

Table 3.5: Pose parameters for the cheetah model

Rotations matrices were defined to relate the orientations of the rigid bodies in the model; these rotation matrices were defined as coordinate rotations according to [59] defined by the Euler angles ϕ, θ and ψ. In order to avoid

singularities, the order of rotations was chosen such that the second rotation was one which would be less than 90° [59].

$$\mathbf{R}_x(\phi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\phi) & \sin(\phi) \\ 0 & -\sin(\phi) & \cos(\phi) \end{bmatrix} \tag{3.5}$$

$$\mathbf{R}_y(\theta) = \begin{bmatrix} \cos(\theta) & 0 & \sin(\theta) \\ 0 & 1 & 0 \\ -\sin(\theta) & 0 & \cos(\theta) \end{bmatrix} \tag{3.6}$$

$$\mathbf{R}_z(\psi) = \begin{bmatrix} \cos(\psi) & \sin(\psi) & 0 \\ -\sin(\psi) & \cos(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.7}$$

The inverse of a rotation matrix is given by its transpose, shown in Equation 3.9.

$$\mathbf{z}' = \mathbf{R}\mathbf{z} \tag{3.8}$$

$$\mathbf{z} = \mathbf{R}^T\mathbf{z}' = \mathbf{R}^{-1}\mathbf{z}' \quad \text{where } \mathbf{z} \text{ and } \mathbf{z}' \text{ are 3-vectors} \tag{3.9}$$

The following rotation matrices were defined for the cheetah:

Rotation matrices
$\mathbf{R}_1^i = \mathbf{R}_z(\psi_1)\mathbf{R}_x(\phi_1)\mathbf{R}_y(\theta_1)$
$\mathbf{R}_2^i = \mathbf{R}_z(\psi_2)\mathbf{R}_x(\phi_2)\mathbf{R}_y(\theta_2)\mathbf{R}_1^i$
$\mathbf{R}_3^i = \mathbf{R}_y(\theta_3)\mathbf{R}_2^i$
$\mathbf{R}_4^i = \mathbf{R}_z(\psi_4)\mathbf{R}_y(\theta_4)\mathbf{R}_3^i$
$\mathbf{R}_5^i = \mathbf{R}_z(\psi_5)\mathbf{R}_y(\theta_5)\mathbf{R}_4^i$
$\mathbf{R}_6^i = \mathbf{R}_z(\psi_6)\mathbf{R}_y(\theta_6)\mathbf{R}_5^i$
$\mathbf{R}_7^i = \mathbf{R}_y(\theta_7)\mathbf{R}_3^i$
$\mathbf{R}_8^i = \mathbf{R}_y(\theta_8)\mathbf{R}_7^i$
$\mathbf{R}_9^i = \mathbf{R}_y(\theta_9)\mathbf{R}_3^i$
$\mathbf{R}_{10}^i = \mathbf{R}_y(\theta_{10})\mathbf{R}_9^i$
$\mathbf{R}_{11}^i = \mathbf{R}_y(\theta_{11})\mathbf{R}_4^i$
$\mathbf{R}_{12}^i = \mathbf{R}_y(\theta_{12})\mathbf{R}_{11}^i$
$\mathbf{R}_{13}^i = \mathbf{R}_y(\theta_{13})\mathbf{R}_4^i$
$\mathbf{R}_{14}^i = \mathbf{R}_y(\theta_{14})\mathbf{R}_{13}^i$

Table 3.6: Rotation matrices for the kinematic model of the cheetah

The positions in the model were mostly based on the markers from the DeepLabCut; however, the paws were excluded as the grass occluded them in most of the videos. A "head" position, located directly between the eyes of the cheetah, was added to define the cheetah's position in the inertial frame, which is a frame fixed relative to the cameras. The positions of the markers in the inertial frame are defined below in Table 3.7. Using the pose parameters in Table 3.5 to define the rotation matrices in Table 3.6, a complete set of positions for the markers of the cheetah can be obtained.

Measurements of the limbs of the cheetah were obtained from the Ann van Dyk Cheetah Centre, and these were used to determine the positions of markers in Table 3.7. These measurements do vary between cheetahs; however, the measurements below were fixed for the cheetahs we observed. We found the measurements to be reasonable approximations for the cheetahs in the videos which were used.

Positions for the kinematic model of the cheetah
$\mathbf{p}_{head} = \begin{bmatrix} x & y & z \end{bmatrix}^T$
$\mathbf{p}_{l_eye} = \mathbf{p}_{head} + \mathbf{R}_i^1 \begin{bmatrix} 0 & 0.03 & 0 \end{bmatrix}^T$
$\mathbf{p}_{r_eye} = \mathbf{p}_{head} + \mathbf{R}_i^1 \begin{bmatrix} 0 & -0.03 & 0 \end{bmatrix}^T$
$\mathbf{p}_{nose} = \mathbf{p}_{head} + \mathbf{R}_i^1 \begin{bmatrix} 0.055 & 0 & -0.055 \end{bmatrix}^T$
$\mathbf{p}_{neck_base} = \mathbf{p}_{head} + \mathbf{R}_i^2 \begin{bmatrix} -0.28 & 0 & 0 \end{bmatrix}^T$
$\mathbf{p}_{spine} = \mathbf{p}_{neck_base} + \mathbf{R}_i^3 \begin{bmatrix} -0.37 & 0 & 0 \end{bmatrix}^T$
$\mathbf{p}_{tail_base} = \mathbf{p}_{spine} + \mathbf{R}_i^4 \begin{bmatrix} -0.37 & 0 & 0 \end{bmatrix}^T$
$\mathbf{p}_{tail_mid} = \mathbf{p}_{tail_base} + \mathbf{R}_i^5 \begin{bmatrix} -0.28 & 0 & 0 \end{bmatrix}^T$
$\mathbf{p}_{tail_tip} = \mathbf{p}_{tail_mid} + \mathbf{R}_i^6 \begin{bmatrix} -0.36 & 0 & 0 \end{bmatrix}^T$
$\mathbf{p}_{l_shoulder} = \mathbf{p}_{neck_base} + \mathbf{R}_i^3 \begin{bmatrix} -0.04 & 0.08 & -0.10 \end{bmatrix}^T$
$\mathbf{p}_{l_front_knee} = \mathbf{p}_{l_shoulder} + \mathbf{R}_i^7 \begin{bmatrix} -0 & 0 & -0.24 \end{bmatrix}^T$
$\mathbf{p}_{l_front_ankle} = \mathbf{p}_{l_front_knee} + \mathbf{R}_i^8 \begin{bmatrix} -0 & 0 & -0.28 \end{bmatrix}^T$
$\mathbf{p}_{r_shoulder} = \mathbf{p}_{neck_base} + \mathbf{R}_i^3 \begin{bmatrix} -0.04 & -0.08 & -0.10 \end{bmatrix}^T$
$\mathbf{p}_{r_front_knee} = \mathbf{p}_{r_shoulder} + \mathbf{R}_i^9 \begin{bmatrix} -0 & 0 & -0.24 \end{bmatrix}^T$
$\mathbf{p}_{r_front_ankle} = \mathbf{p}_{r_front_knee} + \mathbf{R}_i^{10} \begin{bmatrix} -0 & 0 & -0.28 \end{bmatrix}^T$
$\mathbf{p}_{l_hip} = \mathbf{p}_{tail_base} + \mathbf{R}_i^4 \begin{bmatrix} 0.12 & 0.08 & -0.06 \end{bmatrix}^T$
$\mathbf{p}_{l_back_knee} = \mathbf{p}_{l_hip} + \mathbf{R}_i^{11} \begin{bmatrix} -0 & 0 & -0.32 \end{bmatrix}^T$
$\mathbf{p}_{l_back_ankle} = \mathbf{p}_{l_back_knee} + \mathbf{R}_i^{12} \begin{bmatrix} -0 & 0 & -0.25 \end{bmatrix}^T$
$\mathbf{p}_{r_hip} = \mathbf{p}_{tail_base} + \mathbf{R}_i^4 \begin{bmatrix} 0.12 & 0.08 & -0.06 \end{bmatrix}^T$
$\mathbf{p}_{r_back_knee} = \mathbf{p}_{r_hip} + \mathbf{R}_i^{13} \begin{bmatrix} -0 & 0 & -0.32 \end{bmatrix}^T$
$\mathbf{p}_{r_back_ankle} = \mathbf{p}_{r_back_knee} + \mathbf{R}_i^{14} \begin{bmatrix} -0 & 0 & -0.25 \end{bmatrix}^T$

Table 3.7: Positions for the kinematic model of the cheetah

3.4 Calibration

3.4.1 Intrinsic Parameters

For all the trajectory estimation techniques, a set of intrinsic and extrinsic camera parameters was needed. Intrinsic and extrinsic parameters were calibrated using planar calibration boards.

The first step of the calibration was to extract a set of frames from the calibration video. A frame extraction tool was written to allow for a user to select and save individual frames from a video. In order to get a proper calibration, frames were chosen which covered a large area of the camera's FOV, ensuring that the calibration board was captured over a set of varying

positions and orientations.

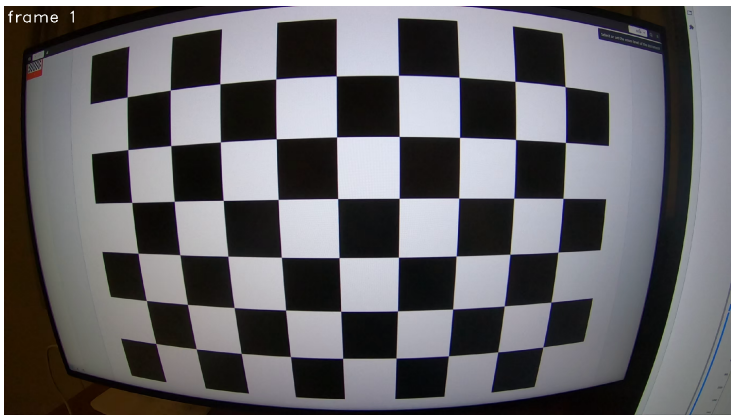

Figure 3.14: A screenshot taken while extracting calibration frames from a GoPro Hero 7 using the frame extraction tool

From the extracted frames, the positions of the internal corners of the calibration board were located using a corner detection function in OpenCV. Internally, OpenCV uses a Harris corner detector [35] to locate the corner positions of the calibration board. Figure 3.14 shows a calibration board with 9×6 corners with each square's edge length equal to 40mm. Figure 3.15 shows grids created from the corner points of 10 extracted frames. The locations of these points were saved to a file.

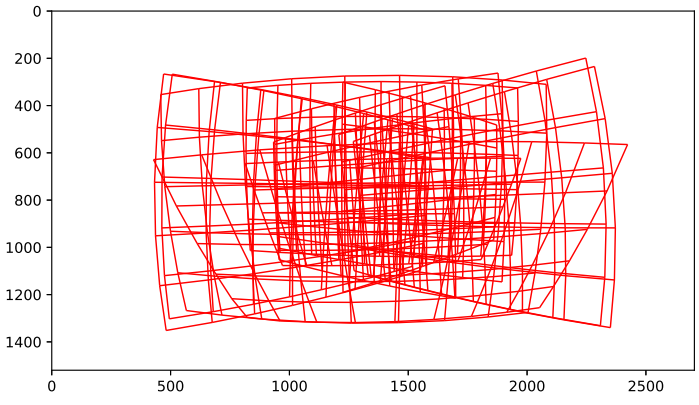

Figure 3.15: A plot of a set of checkerboard locations used for calibration

Using OpenCV's camera calibration function, which is an implementation of

Zhang's algorithm [38], the camera calibration matrix $\mathbf{K}$ and the distortion parameters $\mathbf{D}$, corresponding to Equation 2.16, were obtained for the GoPro Hero 7 cameras. The settings which were used during the filming of the cheetahs and calibration videos are shown in Table 3.8.

Camera Setting	Value
Resolution	2704×1520
Frame Rate	120 fps
Field of View (FOV)	wide
Stabilisation	disabled

Table 3.8: GoPro Hero 7 settings used for filming of the cheetahs and calibration videos

OpenCV's fisheye camera model was used since the cameras had a FOV of 118.2° in the horizontal direction and 69.5° in the vertical direction as well as large amounts of radial distortion. After calibration, the following parameters were obtained:

$$\mathbf{K} = \begin{bmatrix} f_x & \alpha & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1239.74 & 0 & 1345.67 \\ 0 & 1238.68 & 772.86 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.10}$$

$$\mathbf{D} = [k_1, k_2, k_3, k_4] = [0.03744, 0.04786, -0.03764, 0.01082] \tag{3.11}$$

The root-mean-square (RMS) reprojection error for this calibration was 0.48803 pixels.

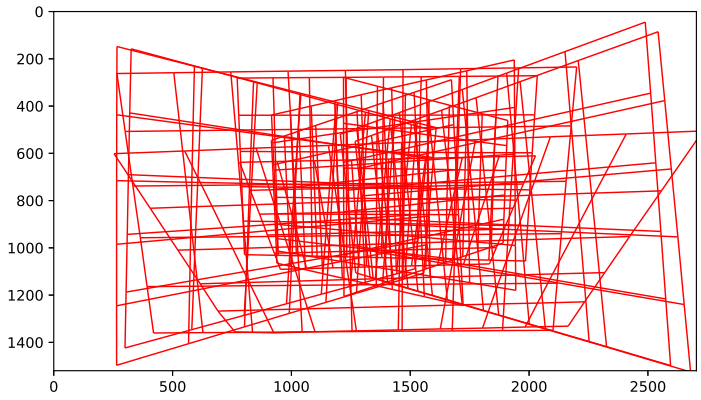

Figure 3.16: The undistorted plot of Figure 3.15

3.4.2 Extrinsic Parameters

Pairwise Extrinsic Parameters

In order to calibrate the extrinsic parameters, a calibration board was moved around in front of all the cameras ensuring that each successive pair of cameras (i.e. camera 1&2, 2&3, 3&4 etc.) obtained multiple corresponding images of the calibration board in different orientations and positions. Similarly to the intrinsic calibration, a 9×6 calibration board was used; however, this calibration board was larger than the one used for intrinsic calibration. The square's edge length on this calibration board was 88mm. This was done so that the corners of the calibration board could be identified at distances up to roughly 10m away from the camera.

By using 2D point correspondences of the corner points from the calibration board, the relative camera pose ($\mathbf{R}$ and $\mathbf{t}$) between each successive pair of cameras was calculated. For 6 cameras, 5 relative poses were obtained. Using the relative poses, a pose for each camera in the world frame was obtained. Denoting the ith relative pose as $\mathbf{R}_{i,i+1}$ and $\mathbf{t}_{i,i+1}$ between cameras i and $i+1$, the relative poses could be chained in order to relate them to the first camera which could then be related to the world coordinate frame. Let the pose of camera i relative to the world frame be denoted by $\mathbf{R}_i$ and $\mathbf{t}_i$.

$$\mathbf{R}_{i+1} = \mathbf{R}_{i,i+1}\mathbf{R}_i \tag{3.12}$$

$$\mathbf{t}_{i+1} = \mathbf{R}_{i,i+1}\mathbf{t}_i + \mathbf{t}_{i,i+1} \tag{3.13}$$

Since we only have 5 relative poses for 6 cameras, the pose of the first camera was provided such that the grid of cameras roughly lay on the xy plane of the world coordinate system so that the z-axis was roughly equal to the height above ground. Camera 1 was positioned at the origin of the global coordinate system.

$$\mathbf{R}_1 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix} \tag{3.14}$$

$$\mathbf{t}_1 = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \tag{3.15}$$

After calculating the initial set of extrinsic parameters, the 3D points corresponding to the pairwise point correspondences were plotted. Figure 3.18 and Figure 3.17 show the plotted 3D points and cameras.

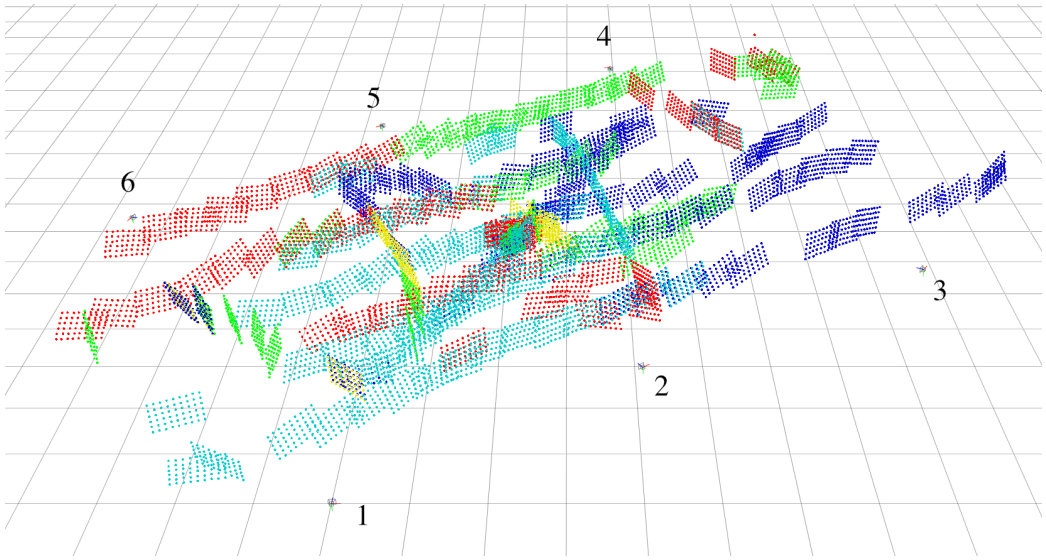

Figure 3.17: Perspective view of the 3D calibration board points from 2D pairwise correspondences (1-2 red, 2-3 green, 3-4 yellow, 4-5 dark blue, 5-6 light blue) where camera 1 is in the bottom left of the image

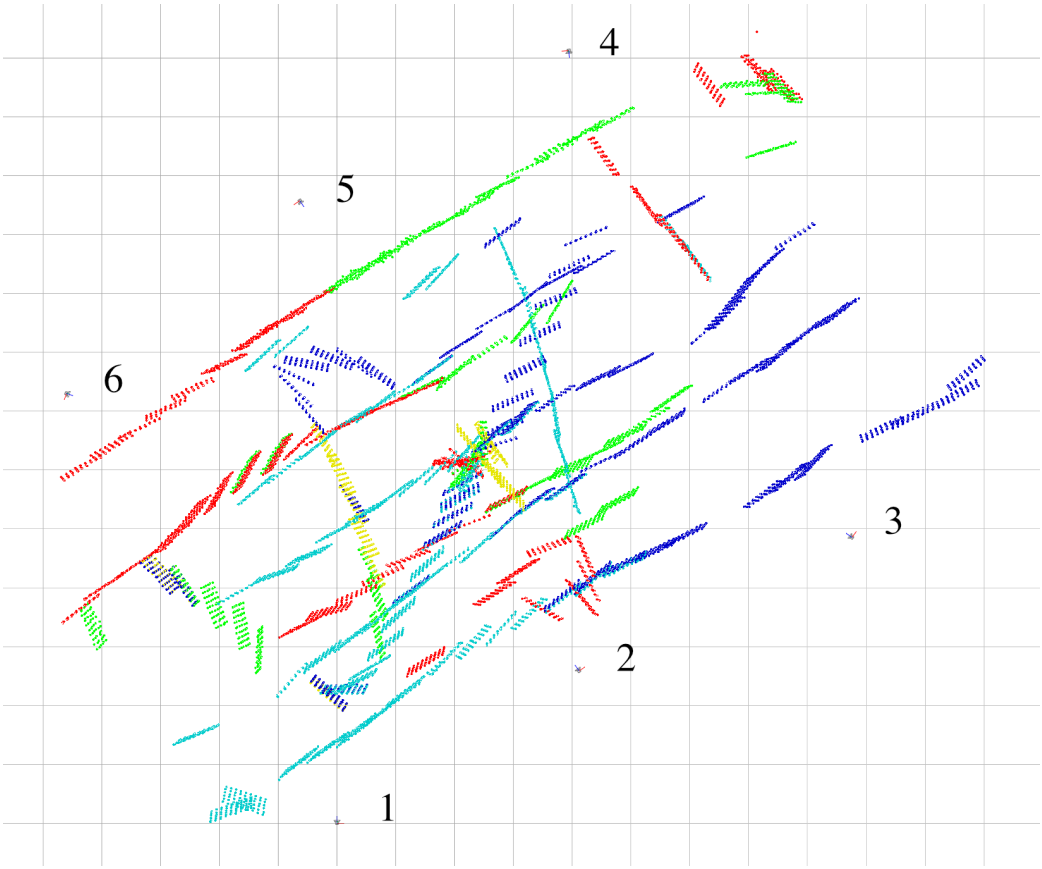

Figure 3.18: Top-down view of the 3D calibration board points from 2D pairwise correspondences (1-2 red, 2-3 green, 3-4 yellow, 4-5 dark blue, 5-6 light blue) where camera 1 is in the bottom left of the image

Extrinsic Refinement Using Sparse Bundle Adjustment

As a final step in the extrinsic calibration process, an SBA was performed using the pairwise extrinsics and 3D point estimates as an initialisation. The SBA optimised over point correspondences between multiple cameras so that the reprojection error for a point seen by more than two cameras was minimised for all cameras (as opposed to just a pair of cameras). This means that correspondences between non-adjacent cameras (i.e. camera 1&3 or 1&6) were also included. Using this additional information, the extrinsics parameters were fine-tuned.

Comparing the reprojection errors of the points before and after the SBA, we can see the improvement in accuracy of the points shown by Figure 3.19 and Table 3.9.

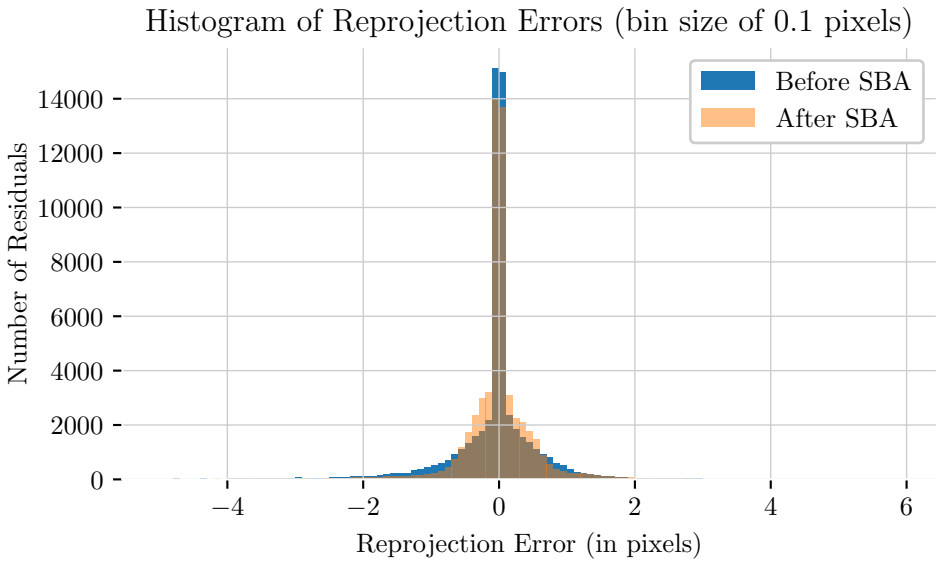

Figure 3.19: Histogram of the calibration reprojeciton errors before and after SBA

After the SBA, the mean reprojection error is closer to zero, and the standard deviation had decreased shown in Table 3.9.

	Mean (Pixels)	Std. Deviation (Pixels)
Before SBA	−0.0872	1.0114
After SBA	−0.0420	0.9695

Table 3.9: Mean and standard deviation of the reprojection errors before and after performing a SBA

The same extrinsic parameters, which were obtained after the SBA, were used for all the trajectory estimation techniques.

Chapter 4

Sparse Bundle Adjustment

4.1 Overview

Given a set of 2D coordinates (u, v) corresponding to 3D points (x, y, z) in a world coordinate frame viewed by a set of cameras, bundle adjustment simultaneously adjusts estimates of the 3D world points, the relative pose of the cameras and the intrinsic parameters of the cameras in order to minimise the error between the 2D reprojections of the 3D estimates and original 2D points. For the case of 3D reconstruction, a known set of camera intrinsic and extrinsic parameters had already been obtained (see Chapter 3). This meant that the only free parameters in the SBA were the 3D points. Therefore, the SBA is reduced to a non-linear minimisation of the reprojection errors for a set of 3D points and corresponding 2D measurements where only the 3D point positions are changed in the optimisation.

4.2 Design

The SBA is constructed such that given a set of measurements $\mathbf{m}$, consisting of the u and v pixel coordinates from DeepLabCut or the simulated data, an optimal set of 3D point estimates $\hat{\mathbf{P}}$ can be found. For each measurement m_i, a function $\mathbf{H_i}(\hat{\mathbf{P}})$, which reprojects the 3D point estimate for m_i was created. The reprojection function $\mathbf{H_i}(\hat{\mathbf{P}})$ was determined by the 3D point it corresponded to and the camera parameters for the camera which observed the point. The reprojected 2D point's u or v estimate from $\mathbf{H_i}(\hat{\mathbf{P}})$ was denoted as $\hat{m}_i$.

$$\hat{m}_i = \mathbf{H_i}(\hat{\mathbf{P}}) \tag{4.1}$$

For a measurement m_i and its corresponding reprojected estimate $\hat{m}_i$ an error residual e_i was calculated.

$$e_i = m_i - \hat{m}_i = m_i - \mathbf{H_i}(\hat{\mathbf{P}}) \tag{4.2}$$

Ideally, one would reduce the square of the error; however, such an implementation would not prevent outliers measurements from skewing the 3D point estimates. A Cauchy robust cost function, shown below in Equation 4.3, was used to try and prevent outliers from skewing the 3D point estimates by reducing the magnitude of their error residual when compared to the naive squared error approach. As shown above in Figure 3.10, measurements from DeepLabCut were assumed to have a standard deviation $\sigma = 5$ pixels with zero mean. Outliers were chosen to be 2D measurements which deviated by more than $3\sigma = 15$ pixels from the 2D reprojections.

$$C(e) = b^2 \ln\left(1 + \frac{e^2}{b^2}\right) \quad \text{where} \quad b = 15 \text{ pixels} \tag{4.3}$$

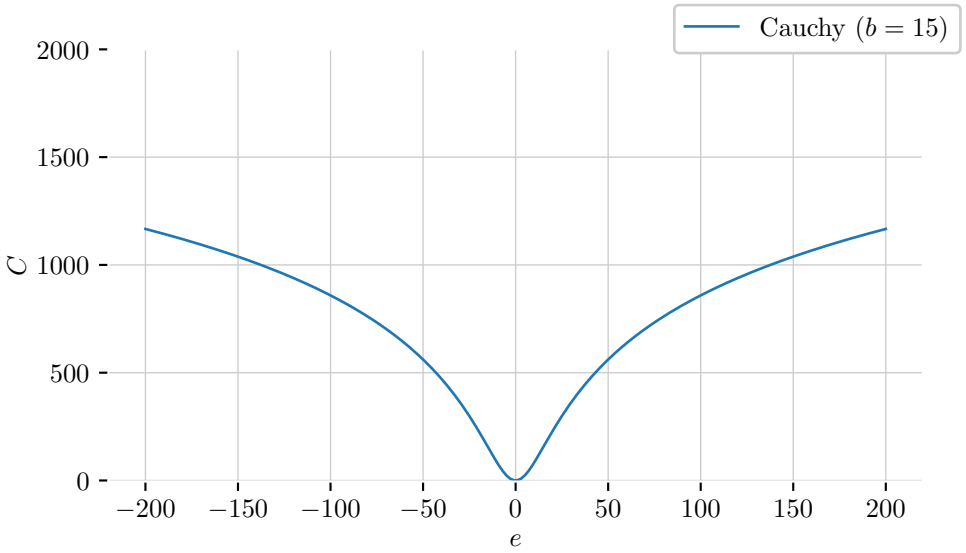

Figure 4.1: Cauchy cost function for the measurement errors

From Figure 4.1, it was seen that for measurement errors which deviate from 0 by less than 15 pixels (3 standard deviations), the cost was dominated by

quadratic term in the equation. Above 15 pixels, the cost of the measurement errors obeys an almost logarithmic relationship.

The resulting problem was a minimisation over the summation over the cost of the residuals for each measurement, shown by Equation 4.4.

$$\min_{\hat{\mathbf{P}}} \quad \sum_{i} C(e_i) = C(m_i - H_i(\hat{\mathbf{P}})) \tag{4.4}$$

The raw data from each real cheetah run consisted of 6 videos of the cheetah, one from each GoPro camera. These videos were synchronised and clipped to be of the same length. From DeepLabCut, 2D points were obtained for each frame from the synchronised videos. The initial 3D point estimates for $\hat{\mathbf{P}}$ were obtained for 2D correspondences with a likelihood above 0.5. In the case of the simulated data, all points in the camera's view were given a likelihood of 1 and all points out of the camera's view had a likelihood of 0. 3D estimates were obtained by triangulating a single 2D point correspondence for the 3D point. If there were multiple 2D point correspondences, only the first correspondence was used.

In order to perform the SBA, a least-squares implementation from the SciPy Python library [60] was used. In order to take advantage of the fact that the Jacobian for the problem was sparsely populated, a Jacobian sparsity pattern was provided. This enabled the computation of elements in the Jacobian known to be zero to be skipped, significantly increasing the speed of the Jacobian computation and thus the speed of the least-squares routine. The elements of the Jacobian were numerically estimated using a 2-point finite difference scheme.

4.3 Simulated Data Results

In order to evaluate the performance of the SBA, two tests were performed. The first of which was a comparison of the error residuals before and after the SBA. This compares the 2D point error for the initial pairwise triangulated 3D points and the estimated 3D points after the SBA. The second test compared 3D position error of the initial triangulated 3D points and the estimated 3D points after the SBA. In order to calculate the 3D point error, the 3D positions were compared to the ground truth simulated 3D points.

As shown, when looking at the estimation statistics for DeepLabCut, outliers can have a severe impact on the mean and standard deviation, which are the

typical statistical measures used for determining accuracy. A more robust set of measures for accuracy, in the presence of outliers, are the median and median absolute deviation (MAD). Tables containing the statistics for the two tests are shown below which list the mean μ, standard deviation σ, median and MAD for each of the simulated tests, before and after the SBA was performed.

4.3.1 Statistics

Before SBA

Test No.	Noise σ (pixels)	Outliers (%)	μ (pixels)	σ (pixels)	Median (pixels)	MAD (pixels)
1	0	0	0.0000	0.0000	0.0000	0.0000
2	0	2	-0.9903	45.4183	0.0000	0.0000
3	0	5	-1.7327	51.3834	0.0000	0.0000
4	5	0	0.0283	5.1846	-0.0199	4.8424
5	5	2	-0.9667	36.6938	0.0604	5.4314
6	5	5	-2.2551	67.9651	0.2188	6.5266
7	10	0	-0.0692	10.4504	-0.0799	9.6960
8	10	2	-0.6370	23.9104	0.1940	10.7497
9	10	5	-2.2821	66.2706	0.3048	12.3092

After SBA

Test No.	Noise σ (pixels)	Outliers (%)	μ (pixels)	σ (pixels)	Median (pixels)	MAD (pixels)
1	0	0	0.0000	0.0000	0.0000	0.0000
2	0	2	-2.1000	40.4312	0.0000	0.0000
3	0	5	-5.5133	44.6218	0.0000	0.0000
4	5	0	-0.0145	4.3227	-0.0396	4.0888
5	5	2	-2.0726	29.0617	-0.0358	4.2172
6	5	5	-4.7134	59.5250	-0.1663	4.4724
7	10	0	0.0170	8.8397	-0.0439	7.5651
8	10	2	-2.0292	21.8183	-0.1234	7.7250
9	10	5	-4.8657	57.0630	-0.2399	7.8971

Table 4.1: 2D reprojection error's mean μ, standard deviation σ, median and median absolute deviation (MAD) before and after SBA

Triangulation					
Noise σ (pixels)	Outliers (%)	μ (m)	σ (m)	Median (m)	MAD (m)
0	0	3.14e-13	1.85e-13	2.88e-13	2.47e-13
0	2	9.57e-02	2.52e+00	3.42e-13	2.91e-13
0	5	1.24e-01	7.67e-01	4.68e-13	4.61e-13
5	0	2.99e-02	1.79e-02	2.64e-02	1.37e-02
5	2	7.33e-02	3.94e-01	2.81e-02	1.76e-02
5	5	1.44e-01	7.47e-01	3.33e-02	2.38e-02
10	0	6.12e-02	3.69e-02	5.33e-02	2.84e-02
10	2	9.01e-02	1.24e-01	5.83e-02	3.50e-02
10	5	1.77e-01	1.33e+00	6.58e-02	4.49e-02
SBA					
Noise σ (pixels)	Outliers (%)	μ (m)	σ (m)	Median (m)	MAD (m)
0	0	4.27e-13	2.00e-13	4.41e-13	2.61e-13
0	2	1.67e-01	7.01e+00	4.78e-13	2.59e-13
0	5	5.94e-02	1.70e+00	5.80e-13	3.97e-13
5	0	2.09e-02	9.68e-03	1.97e-02	9.39e-03
5	2	3.47e-02	5.81e-01	1.99e-02	9.32e-03
5	5	7.93e-02	1.41e+00	2.07e-02	1.07e-02
10	0	4.42e-02	2.12e-02	4.11e-02	2.00e-02
10	2	4.48e-02	2.65e-02	4.15e-02	2.05e-02
10	5	1.32e-01	1.92e+00	4.42e-02	2.12e-02

Table 4.2: 3D position error's mean μ, standard deviation σ, median and median absolute deviation (MAD) before and after SBA

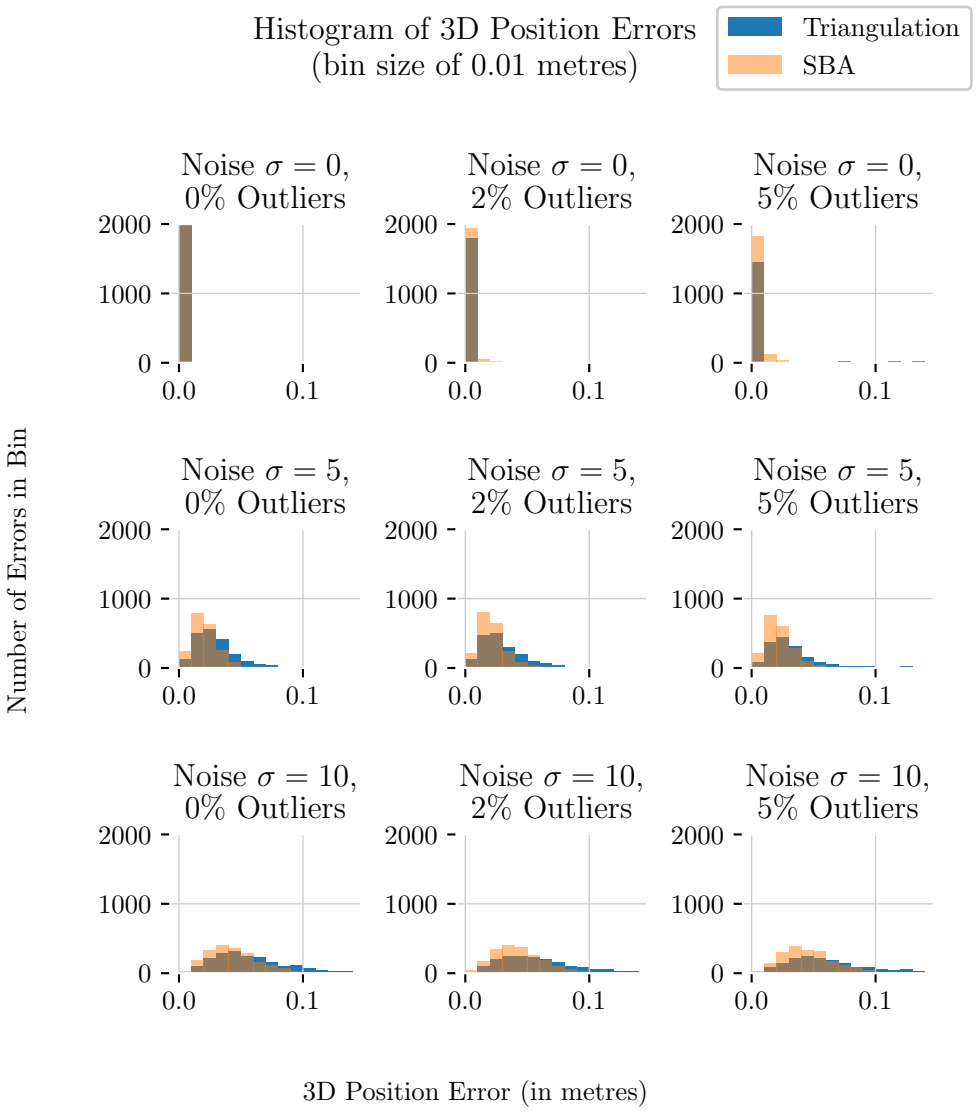

Figure 4.2: Histogram of the 3D position errors

Visual Results

To demonstrate the effect that the SBA has on the data, 3D point cloud visualisations were created which display each marker as a specific colour. All the points for each marker corresponding to different time steps in the trajectory are plotted together to show how the SBA reduces noise and helps reduce the effect of outlier measurements.

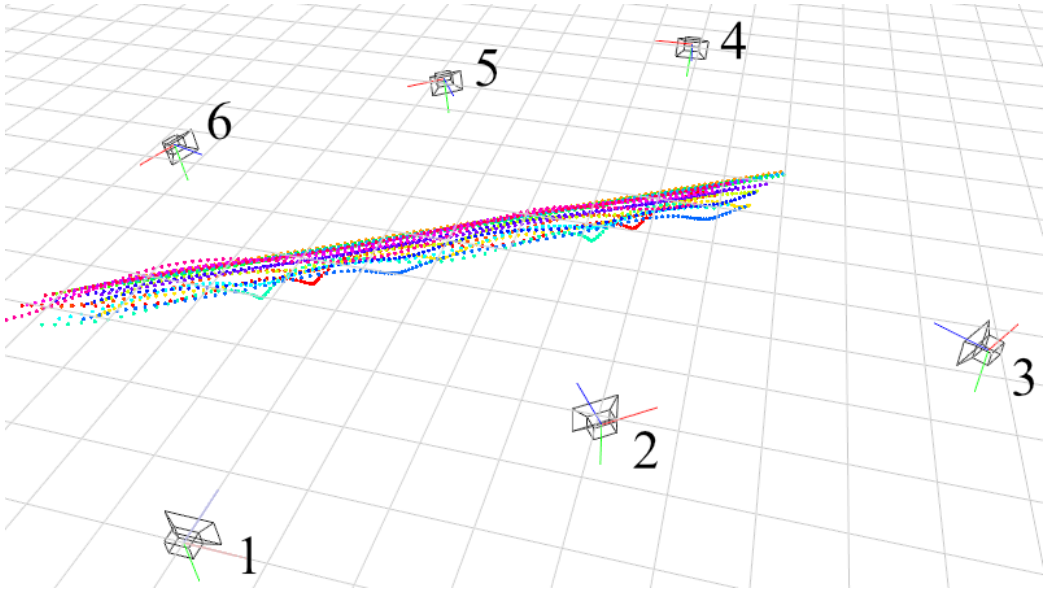

Figure 4.3: Plot of the ground truth 3D point cloud of the simulated cheetah trajectory

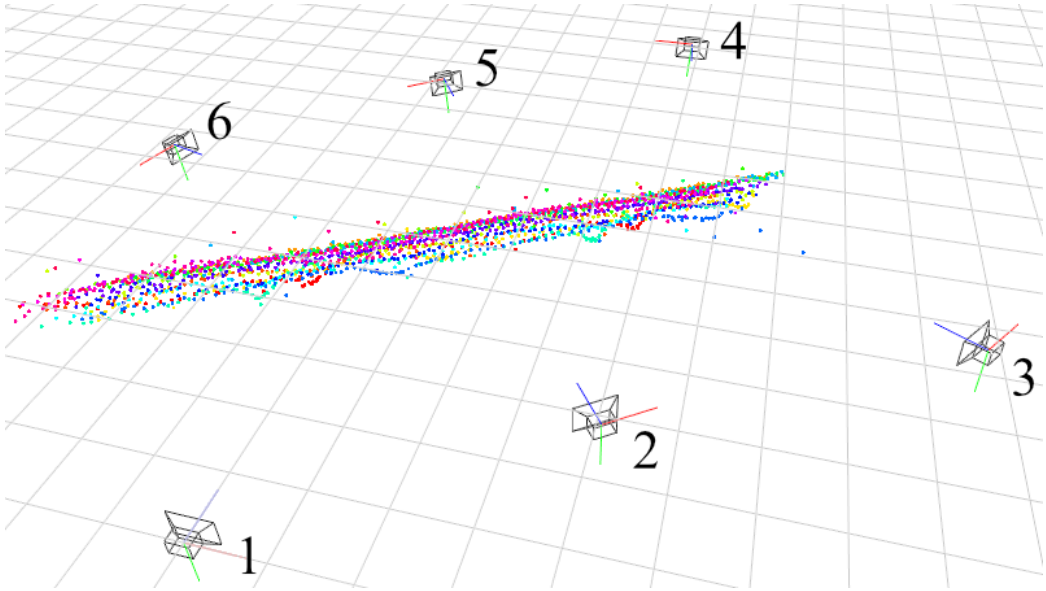

Figure 4.4: Plot of the initial triangulated 3D point cloud of the simulated cheetah trajectory for test 5

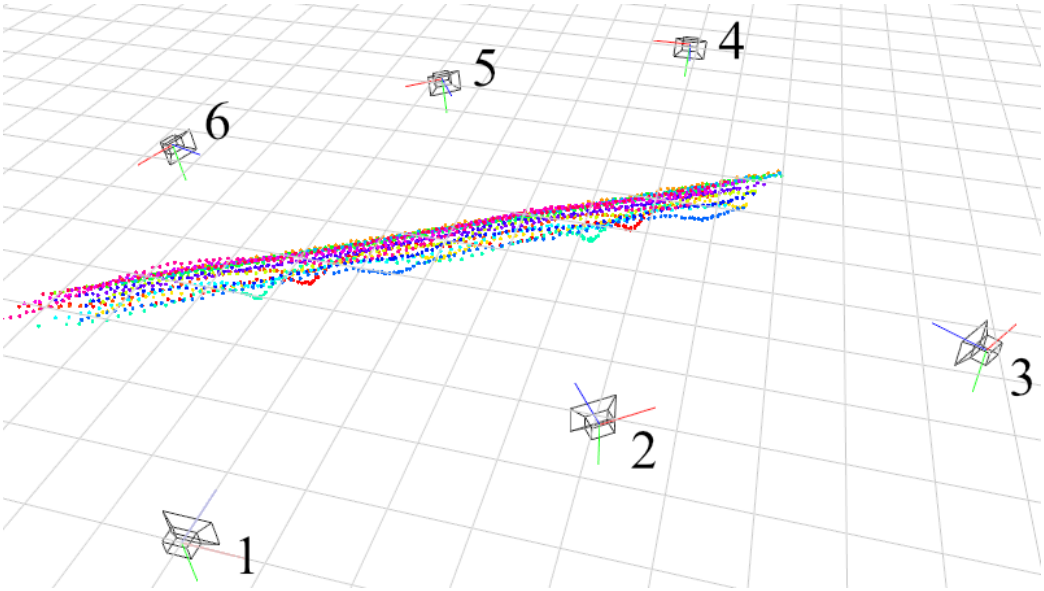

Figure 4.5: Plot of the estimated 3D point cloud of the simulated cheetah trajectory for test 5 after SBA

4.4 Real Data Results

After using the SBA to estimate the 3D marker positions for a real cheetah run, it was immediately apparent that the points did not accurately depict the cheetah. However, the SBA did help in the cleaning up of the data by reducing the noise and removing outliers like in the simulated tests. Shown below in Figure 4.6 and Figure 4.7 are plots of a point cloud of all the markers from before and after the SBA. Each marker was given a different colour.

3D plots of the cheetah, based off data from the SBA, were plotted next to the videos they were captured from to demonstrate the effectiveness of the SBA. These 3D visual representations provided a useful way to determine problems that needed to be addressed. It should be noted that the positions of the cheetahs in these plots do not correspond exactly to the images of the cheetah due to the fisheye effect of the camera.

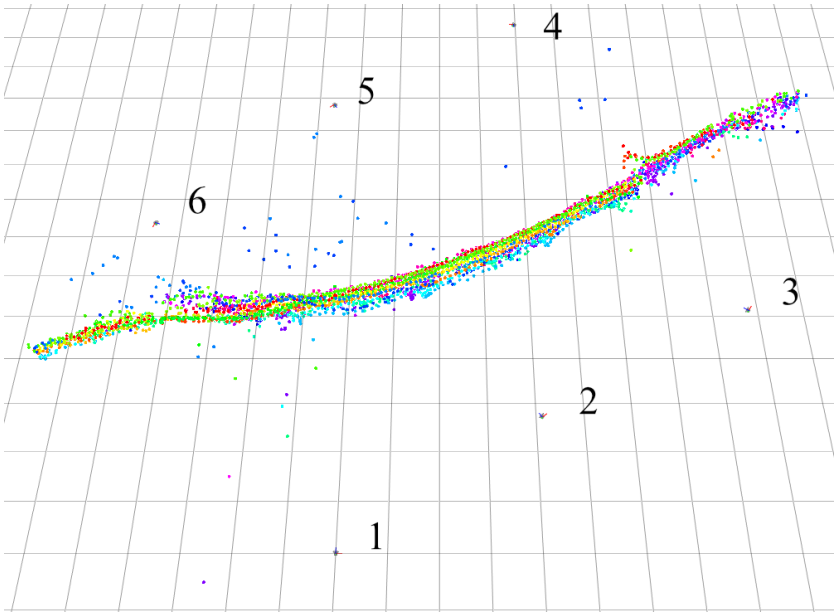

Figure 4.6: Plot of the 3D point cloud of the cheetah trajectory from pairwise triangulation

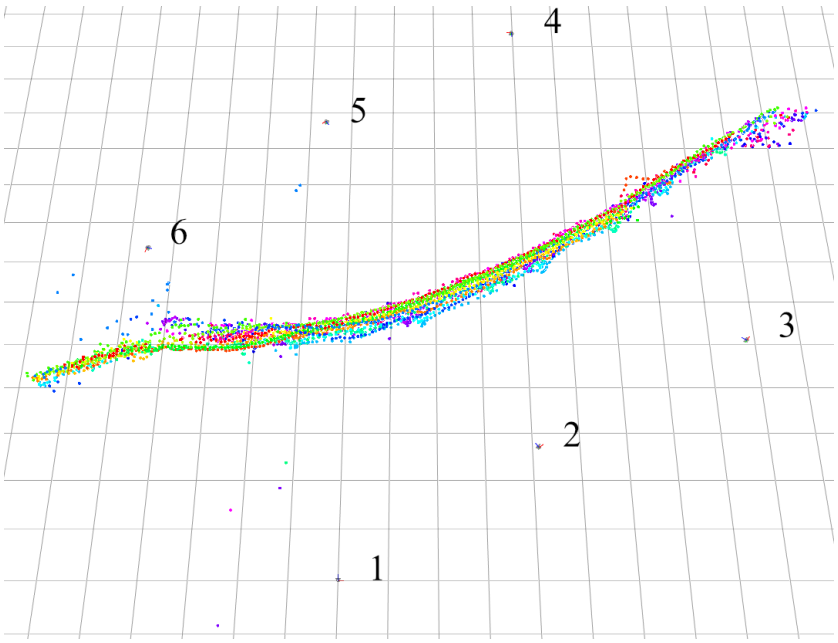

Figure 4.7: Plot of the 3D point cloud of the cheetah trajectory after SBA

4.4.1 Cheetah Run

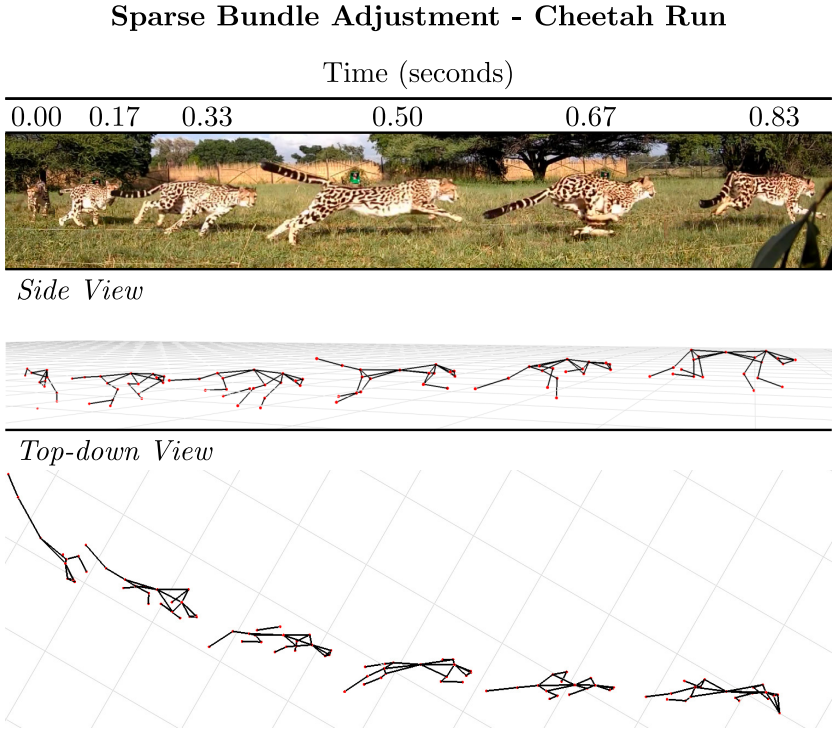

Figure 4.8: Plot of the SBA trajectory estimation for the cheetah run

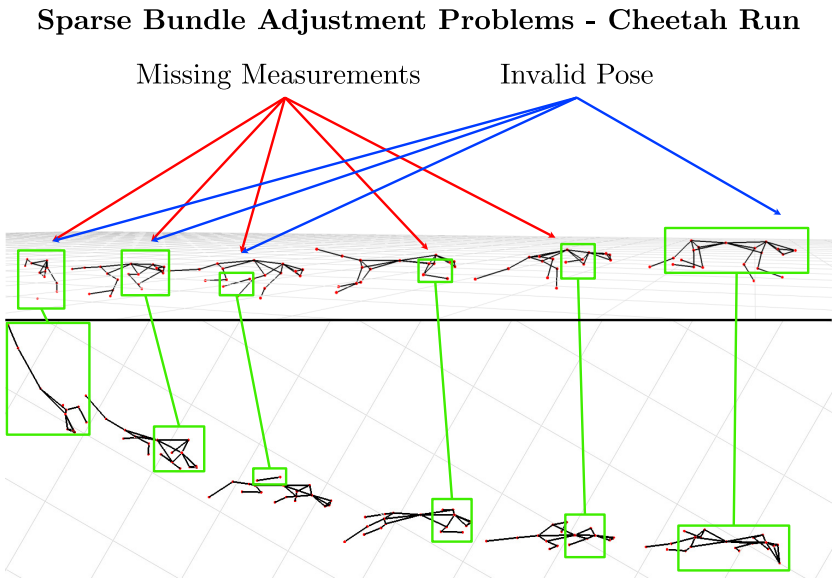

Figure 4.9: Errors in the SBA trajectory estimation for the cheetah run

4.4.2 Cheetah Dive

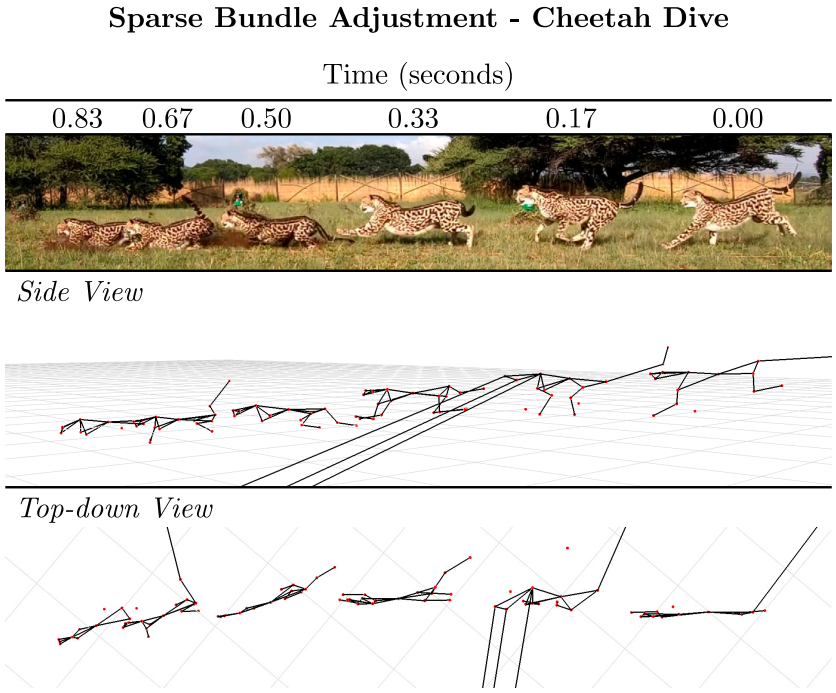

Figure 4.10: Plot of the SBA trajectory estimation for the cheetah dive

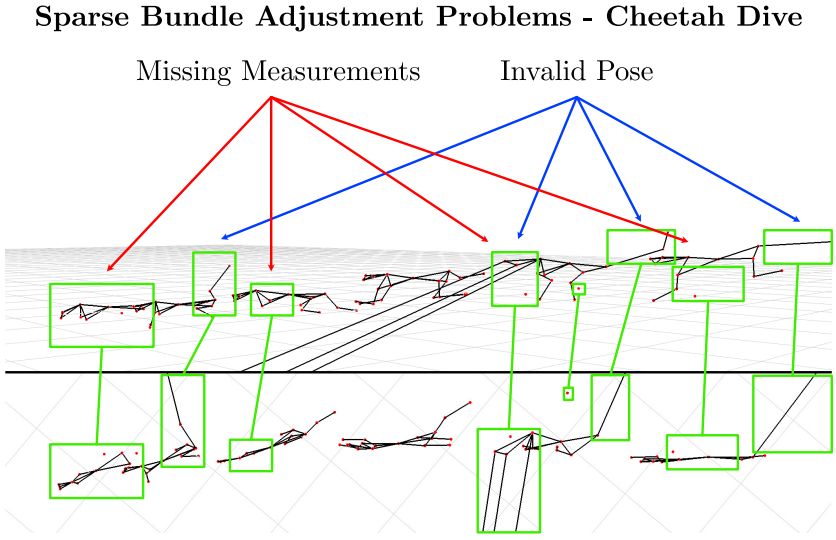

Figure 4.11: Errors in the SBA trajectory estimation for the cheetah dive

4.5 Discussion

From the results for the simulated data, it could be seen that for the 2D errors, the SBA reduces the standard deviation and MAD in all cases (which is what was expected). In nearly all cases for the 3D errors, the mean and the standard deviation were worse after the SBA when there were outliers in the data. Figure 4.12 shows an example of this. This was because, for a few points where the 2D measurement was an outlier for multiple cameras, the 3D estimate would end up far away from its correct position.

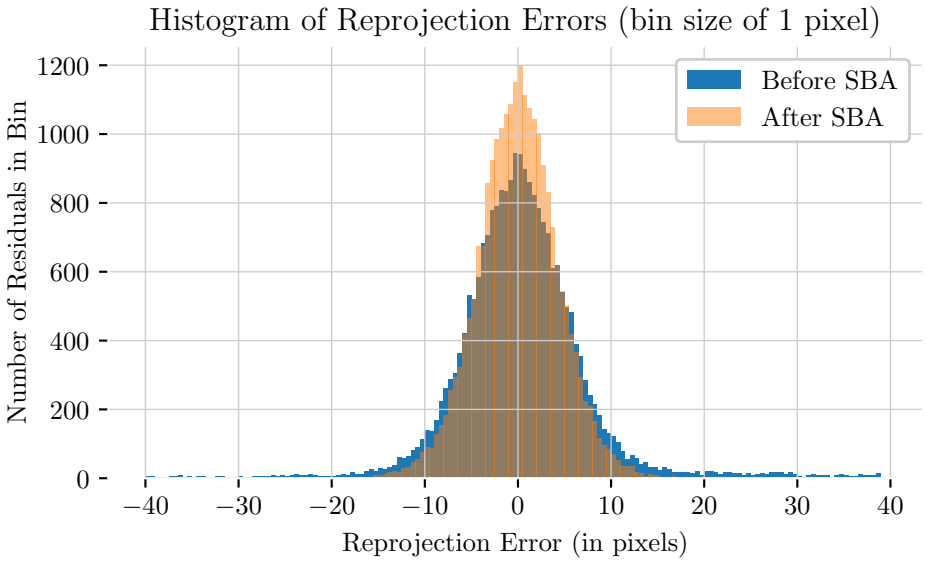

Figure 4.12: Histogram of the 2D reprojection errors for test 5

Arguably the most important measures when it comes to dealing with noise are the robust measures for the 3D position. It was seen that the SBA reduced the MAD and the median values were negligibly small in all cases. For the cases where there was no noise, the MAD was zero as it ignored the outliers. It was difficult to quantify the reduction in outliers; however, upon looking at the point cloud visualisation of the data and the histogram plots, the effect that the SBA has on the outliers is obvious.

The histogram plots in Figure 4.2 give a good indication of how the SBA helped in providing better 3D point estimates. It was visually evident that the SBA reduced the spread of the 3D position errors and effectively reduced the number of outliers.

While the results looked promising for the SBA, there were several problems

around using the 3D point estimates to reconstruct a trajectory. Firstly, as each point was adjusted independently, they did not obey any structure. This resulted in many of the estimated cheetah poses being infeasible when noise or outliers were present. Additionally, marker locations jumped around between frames giving a "jitter" effect when looking at markers temporally. This provided two objectives which guided the selection of the subsequent trajectory estimation methods. The first objective was to provide some form of structural consistency such that the estimated positions were kinematically valid poses for the cheetah. The second objective was to provide temporal consistency between these poses.

Chapter 5

Extended Kalman Filter

5.1 Overview

To address the shortcomings of the SBA, namely the lack of pose structure and temporal consistency, a Kalman filter was designed based on a rigid body model of the cheetah. The goal of the Kalman filter was to estimate the pose of the cheetah for each time step in a trajectory (corresponding to the frames seen by the cameras) based on measurements from the cameras. Due to the non-linearities in the measurement function, an extended Kalman filter (EKF) was used.

5.2 Design

5.2.1 State Variables

The first step in the design of the EKF was to choose the state variables. Since the task involved the tracking of the cheetah's pose, parameters used to define the pose, θ, corresponding to the pose parameters in Table 3.5, were used as state variables. θ consisted of 24 variables which described the global position and orientation of the cheetah as well the angles which defined the relative rotations between the rigid bodies of the cheetah. This addressed the issue of pose structure from the SBA.

As the cameras were recording at 120fps, it was assumed that between frames, the change in pose parameters could be approximated by a constant acceleration model. Thus, in addition to the pose parameters, θ, the velocities and angular velocities, $\dot{\theta}$, and the accelerations and angular accelerations, $\ddot{\theta}$,

were also included as part of the state. This addressed the issue of temporal consistency from the SBA.

A state vector $\mathbf{x}$ was composed of θ, $\dot{\theta}$ and $\ddot{\theta}$, each consisting of 24 states for a total of 72 states in $\mathbf{x}$.

$$\mathbf{x} = \begin{bmatrix} \theta_{(24\times1)} \\ \dot{\theta}_{(24\times1)} \\ \ddot{\theta}_{(24\times1)} \end{bmatrix} \tag{5.1}$$

5.2.2 Process Model

As mentioned above, it was assumed that between frames, the pose parameters underwent constant acceleration. This was assumed for all frames, and thus the state transition matrix, shown in Equation 5.2 was constant across the trajectory.

$$\mathbf{F} = \begin{bmatrix} I_{(24\times24)} & \Delta t I_{(24\times24)} & \frac{\Delta t^2}{2} I_{(24\times24)} \\ 0_{(24\times24)} & I_{(24\times24)} & \Delta t I_{(24\times24)} \\ 0_{(24\times24)} & 0_{(24\times24)} & I_{(24\times24)} \end{bmatrix} \tag{5.2}$$

Since there are no inputs to the system, the a priori state estimate is given by Equation 5.3.

$$\hat{\mathbf{x}}_{k|k-1} = \mathbf{F}\hat{\mathbf{x}}_{k-1|k-1} \tag{5.3}$$

5.2.3 Process Covariance Matrix

To calculate the process covariance $\mathbf{P}$, it was required that the process noise covariance $\mathbf{Q}$ be determined. As the chosen model assumes constant acceleration, the changes in acceleration between time steps are accounted for as noise.

Following the derivations in [48], a piecewise white noise model was used to derive $\mathbf{Q}$ where it was assumed that the acceleration noise was a discrete Weiner process. Modelling the noise this way, the acceleration noise, $w(t)$, is assumed to be constant between each time step but jumps in a discontinuous manner at each time step. $w(t)$ is assumed to be normally distributed with zero mean.

For a subset of the state vector $\mathbf{x}_i = \begin{bmatrix}\theta_i & \dot{\theta}_i & \ddot{\theta}_i\end{bmatrix}^T$ corresponding to a single marker with acceleration noise $w_i(t)$ having zero mean and standard deviation σ_i, the process noise covariance $\mathbf{Q}_i$ is given by Equation 5.7. Γ is the acceleration noise gain for each of the states.

$$\mathbf{\Gamma} = \begin{bmatrix}\frac{\Delta t^2}{2} \\ \Delta t \\ 1\end{bmatrix} \tag{5.4}$$

$$\begin{aligned}\mathbf{Q}_i &= \mathbb{E}[\mathbf{\Gamma} w_i(t) w_i(t) \mathbf{\Gamma}^T] && (5.5)\\ &= \mathbf{\Gamma}\sigma_i^2\mathbf{\Gamma} && (5.6)\\ &= \begin{bmatrix}\frac{\Delta t^4}{4} & \frac{\Delta t^3}{2} & \frac{\Delta t^2}{2} \\ \frac{\Delta t^3}{2} & \Delta t^2 & \Delta t \\ \frac{\Delta t^2}{2} & \Delta t & 1\end{bmatrix}\sigma_i^2 && (5.7)\end{aligned}$$

From this, it was relatively simple to obtain the process covariance noise matrix $\mathbf{Q}$. Letting $w_i(t)$ be the acceleration noise with standard deviation σ_i for the ith value in $\ddot{\theta}$, a (24×24) diagonal matrix $\mathbf{q}$ was composed where the ith diagonal element corresponded to the variance, σ_i^2, for noise of the ith element in $\ddot{\theta}$.

$$\mathbf{q} = \begin{bmatrix}\sigma_1^2 & & \\ & \ddots & \\ & & \sigma_{24}^2\end{bmatrix} \tag{5.8}$$

$$\mathbf{Q} = \begin{bmatrix}\frac{\Delta t^4}{4}\mathbf{q} & \frac{\Delta t^3}{2}\mathbf{q} & \frac{\Delta t^2}{2}\mathbf{q} \\ \frac{\Delta t^3}{2}\mathbf{q} & \Delta t^2\mathbf{q} & \Delta t\mathbf{q} \\ \frac{\Delta t^2}{2}\mathbf{q} & \Delta t\mathbf{q} & \mathbf{q}\end{bmatrix} \tag{5.9}$$

Modelling the noise in this way was convenient as it allowed us to use the variance σ^2 for each of the acceleration states to determine the process noise. The variance for each acceleration states could initially be determined based off an estimated trajectory and then iteratively fine-tuned to improve the results. It was assumed that $\mathbf{Q}$ was constant throughout the trajectory.

In order to determine the variance for each of the acceleration states, rough estimates were initially chosen based on approximations of what was thought to be the change in acceleration between samples $\Delta\ddot{\theta}$. Based on guidelines

from [48], σ_i was chosen to be between $\frac{1}{2}\Delta\ddot{\theta}_i$ and $\Delta\ddot{\theta}_i$. While initially the values were guessed, after running the EKF and converging on a trajectory, the values could be determined from the values of $\ddot{\theta}$ for the trajectory. These values were iteratively refined over many runs. The values used for the tests are shown in Table 5.1.

Acceleration states	Estimated Variance $(m/s^2)^2$ or $(rad/s^2)^2$
$\ddot{x}, \ddot{y}, \ddot{z}$	12.5, 12.5, 12.5
$\ddot{\phi}_1, \ddot{\theta}_1, \ddot{\psi}_1$	50.0, 50.0, 50.0
$\ddot{\phi}_2, \ddot{\theta}_2, \ddot{\psi}_2$	12.5, 312.5, 12.5
$\ddot{\theta}_3$	1250.0
$\ddot{\theta}_4, \ddot{\psi}_4$	1250.0, 312.5
$\ddot{\theta}_5, \ddot{\psi}_5$	5000.0, 450.0
$\ddot{\theta}_6, \ddot{\psi}_6$	9800.0, 800.0
$\ddot{\theta}_7$	61250.0
$\ddot{\theta}_8$	20000.0
$\ddot{\theta}_9$	61250.0
$\ddot{\theta}_{10}$	20000.0
$\ddot{\theta}_{11}$	101250.0
$\ddot{\theta}_{12}$	80000.0
$\ddot{\theta}_{13}$	101250.0
$\ddot{\theta}_{14}$	80000.0

Table 5.1: Variance values used for the acceleration states in the process covariance matrix

5.2.4 Measurement Function

A measurement function for the EKF was required to produce a set of measurement estimates given a set of state estimates $\hat{\mathbf{x}}$. This function was broken into two components. Firstly a 3D pose needed to be obtained from the state estimates. From the 3D pose, each of the 3D marker positions from the pose needed to be projected into the camera views (in order to be compared against the camera measurements).

For a state estimate $\hat{\mathbf{x}}$, the 3D positions of each marker for the cheetah could be determined using the equations from Table 3.6 and Table 3.7. These 3D positions were then projected into the view of each camera using Equation 2.15 (excluding the head position as there was no corresponding head marker measurement). Denoting the estimated measurement for coordinate c (either

u or v) for a marker m seen by camera i as $\hat{\mathbf{z}}_{i,m,c}$, the measurement equation provides a measurement estimate $\hat{\mathbf{z}}$. $\hat{\mathbf{z}}$ is comprised of 240 elements, (6 cameras $\times$ 20 markers $\times$ 2 pixel coordinates).

$$\hat{\mathbf{z}} = \mathbf{h}(\hat{\mathbf{x}}) = \begin{bmatrix} \hat{\mathbf{z}}_{1,l_eye,u} \\ \hat{\mathbf{z}}_{1,l_eye,v} \\ \vdots \\ \hat{\mathbf{z}}_{1,r_back_ankle,v} \\ \hat{\mathbf{z}}_{2,l_eye,u} \\ \vdots \\ \hat{\mathbf{z}}_{6,r_back_ankle,v} \end{bmatrix} \tag{5.10}$$

At each time step k, the Jacobian of the measurement equation, $\mathbf{H}_k$, was numerically estimated using a 2-point finite difference scheme.

$$\mathbf{H}_k = \frac{\partial \mathbf{h}}{\partial \mathbf{x}}\Big|_{\hat{\mathbf{x}}_{k|k-1}} \tag{5.11}$$

5.2.5 Measurement Covariance

The measurement covariance matrix $\mathbf{R}$ was constructed as a diagonal matrix where each diagonal element was equal to the measurement variance, $\sigma^2 = (5 \text{ pixels})^2 = 25$, from DeepLabCut. However, when a measurement from DeepLabCut had a low likelihood, the diagonal element corresponding to that measurement was set to 2704^2 which was the square of the max possible pixel error when a cheetah was in the frame.

$$\mathbf{R} = \begin{bmatrix} \sigma^2 & & \\ & \ddots & \\ & & \sigma^2 \end{bmatrix} \tag{5.12}$$

5.2.6 Outlier Rejection

The measurement variance from DeepLabCut was very small, and so in the case of bad outlier measurements, the filter would diverge. In order to prevent outliers from causing the state to diverge, an extra step was added to the update stage of the EKF such that if an outlier was detected, the innovation for that measurement was set to zero [61].

An outlier could be determined using the innovation, $\tilde{\mathbf{y}}$, and innovation covariance, $\mathbf{S}$. Using the rule of thumb that outliers were measurements which deviated from the mean by more than three standard deviations, outliers were assumed to be measurements whose innovation was three times greater than the square root of the innovation covariance.

5.2.7 Prediction and Update Algorithm

Prediction	
Step	Equation
A priori state estimate	$\hat{\mathbf{x}}_{k\|k-1} = \mathbf{F}\hat{\mathbf{x}}_{k-1\|k-1}$
A priori state covariance	$\mathbf{P}_{k\|k-1} = \mathbf{F}\mathbf{P}_{k-1\|k-1}\mathbf{F}^T + \mathbf{Q}$
Update	
Step	Equation
Innovation	$\tilde{\mathbf{y}}_k = \mathbf{z}_k - \mathbf{h}_k(\hat{\mathbf{x}}_{k\|k-1})$
Innovation covariance	$\mathbf{S}_k = \mathbf{H}_k\mathbf{P}_{k\|k-1}\mathbf{H}_k^T + \mathbf{R}$
Outlier rejection	$\tilde{\mathbf{y}}_{k,i} = 0$ where $\tilde{\mathbf{y}}_{k,i} \geq 3\sqrt{\mathbf{S}_{k,i}}$
Kalman Gain	$\mathbf{K_k} = \mathbf{P}_{k\|k-1}\mathbf{H}_k^T + \mathbf{S}_k^{-1}$
A posteriori state estimate	$\hat{\mathbf{x}}_{k\|k} = \hat{\mathbf{x}}_{k\|k-1} + \mathbf{K_k}\tilde{\mathbf{y}}_k$
A posteriori state covariance	$\mathbf{P}_{k\|k} = (\mathbf{I} - \mathbf{K_k}\mathbf{H}_k)\mathbf{P}_{k\|k-1}$

Table 5.2: Modified EKF algorithm

5.2.8 Initialisation of the EKF

State Initialisation

For each run, the position and orientation of the cheetah were estimated using an initial set of triangulated 3D points. This was done by plotting the estimated x, y and z coordinates for the nose over the trajectory and fitting a straight line to each of the x, y and z trajectories. The initial x, y and z position for the cheetah was equal to the first point of each of the fitted lines, while the yaw of the cheetah was determined from the slope of the fitted lines for the x and y coordinates. The initial x, y and z velocity for the cheetah was obtained using the slope of the fitted lines for the x, y and z coordinates of the nose. All of the other states were initialised to zero. The main two assumptions made for this initialisation scheme were that the cheetah was running in a forward-facing direction and that the roll and pitch of the cheetah were relatively small. While it was possible to have a far more complex initialisation scheme which could be used to initialise the other states for the cheetah, the simple initialisation method mentioned

above worked well and so there was no need to invest more time in the initialisation of the state vector.

State Covariance Initialisation

The state covariance was initialised based on assumptions of how accurately the initial states were likely to represent the actual states of the cheetah. By setting high initial state covariance values, the EKF would quickly converge based on the measurements instead of relying on the state estimates; however, the uncertainty would also allow outliers to have greater influence. These values needed to be tuned by hand for each run.

For the simulated data, the following initial state covariance estimates were used. The variance values for each of the states is shown in Table 5.3. These values populated the diagonal of the state covariance matrix while all other values in the covariance matrix were set to zero.

States	Initial Variance
$\ddot{x}, \ddot{y}, \ddot{z}$	9.0 $(m/s^2)^2$
$\ddot{\phi}_{1...2}, \ddot{\theta}_{1..4}, \ddot{\psi}_{1..4}$	9.0 $(rad/s^2)^2$
$\ddot{\theta}_{5...14}, \ddot{\psi}_{5,6}$	25.0 $(rad/s^2)^2$
$\dot{x}, \dot{y}, \dot{z}$	25.0 $(m/s)^2$
$\dot{\phi}_{1...2}, \dot{\theta}_{1..14}, \dot{\psi}_{1..6}$	9.0 $(rad/s)^2$
x, y, z	25.0 m^2
$\phi_{1...2}, \theta_{1..14}, \psi_{1..6}$	0.6 rad^2

Table 5.3: Variance values used for the acceleration states in the process covariance matrix

5.2.9 Kalman Smoother

As the videos were being processed "offline", the entire set of measurements, $\mathbf{z}_1 \dots \mathbf{z}_n$, could be used to compute maximum likelihood state estimates for the trajectory using a Kalman smoother. For this, a Rauch-Tung-Striebel Kalman smoother [51] was implemented using to Equation 5.13, Equation 5.14 and Equation 5.15.

$$\mathbf{A}_k = \mathbf{P}_{k|k}\mathbf{F}^T\mathbf{P}^{-1}_{k+1|k} \tag{5.13}$$

$$\hat{\mathbf{x}}_{k|n} = \hat{\mathbf{x}}_{k|k} + \mathbf{A}_k(\hat{\mathbf{x}}_{k+1|n} - \hat{\mathbf{x}}_{k+1|k}) \tag{5.14}$$

$$\mathbf{P}_{k|n} = \mathbf{P}_{k|k} + \mathbf{A}_k(\mathbf{P}_{k+1|n} - \mathbf{P}_{k+1|k})\mathbf{A}_k^T \tag{5.15}$$

5.3 Simulated Data Results

Similarly to the SBA, tests comparing the 3D position error of the 3D points for both the EKF and the Kalman smoother were performed. In order to calculate the geometric 3D error, the 3D positions were compared to the ground truth simulated 3D points.

Both the standard and robust measures for accuracy were obtained, and are listed in the table below which lists the mean μ, standard deviation σ, median and MAD for each of the simulated tests for the EKF and Kalman smoother.

EKF					
Noise σ (pixels)	Outliers (%)	μ (m)	σ (m)	Median (m)	MAD (m)
0	0	5.17e-03	9.62e-03	3.17e-03	2.41e-03
0	2	6.24e-03	9.99e-03	3.96e-03	2.73e-03
0	5	8.40e-03	1.48e-02	5.01e-03	3.37e-03
5	0	9.23e-03	9.74e-03	7.13e-03	4.29e-03
5	2	1.08e-02	1.34e-02	7.84e-03	4.92e-03
5	5	1.30e-02	2.05e-02	9.03e-03	5.82e-03
10	0	1.60e-02	1.25e-02	1.34e-02	7.19e-03
10	2	1.52e-02	1.32e-02	1.20e-02	7.34e-03
10	5	1.56e-02	1.27e-02	1.24e-02	7.77e-03
Smoother					
Noise σ (pixels)	Outliers (%)	μ (m)	σ (m)	Median (m)	MAD (m)
0	0	2.81e-03	9.13e-03	9.29e-04	6.56e-04
0	2	3.30e-03	9.17e-03	1.36e-03	1.03e-03
0	5	4.99e-03	1.32e-02	2.27e-03	1.50e-03
5	0	4.92e-03	8.95e-03	3.05e-03	1.83e-03
5	2	6.48e-03	1.22e-02	4.03e-03	2.27e-03
5	5	8.59e-03	1.90e-02	5.41e-03	3.28e-03
10	0	8.74e-03	1.02e-02	6.78e-03	4.65e-03
10	2	8.71e-03	1.06e-02	6.43e-03	3.97e-03
10	5	8.46e-03	1.01e-02	6.17e-03	4.16e-03

Table 5.4: 3D position error's mean μ, standard deviation σ, median and median absolute deviation (MAD) from the EKF and Kalman Smoother

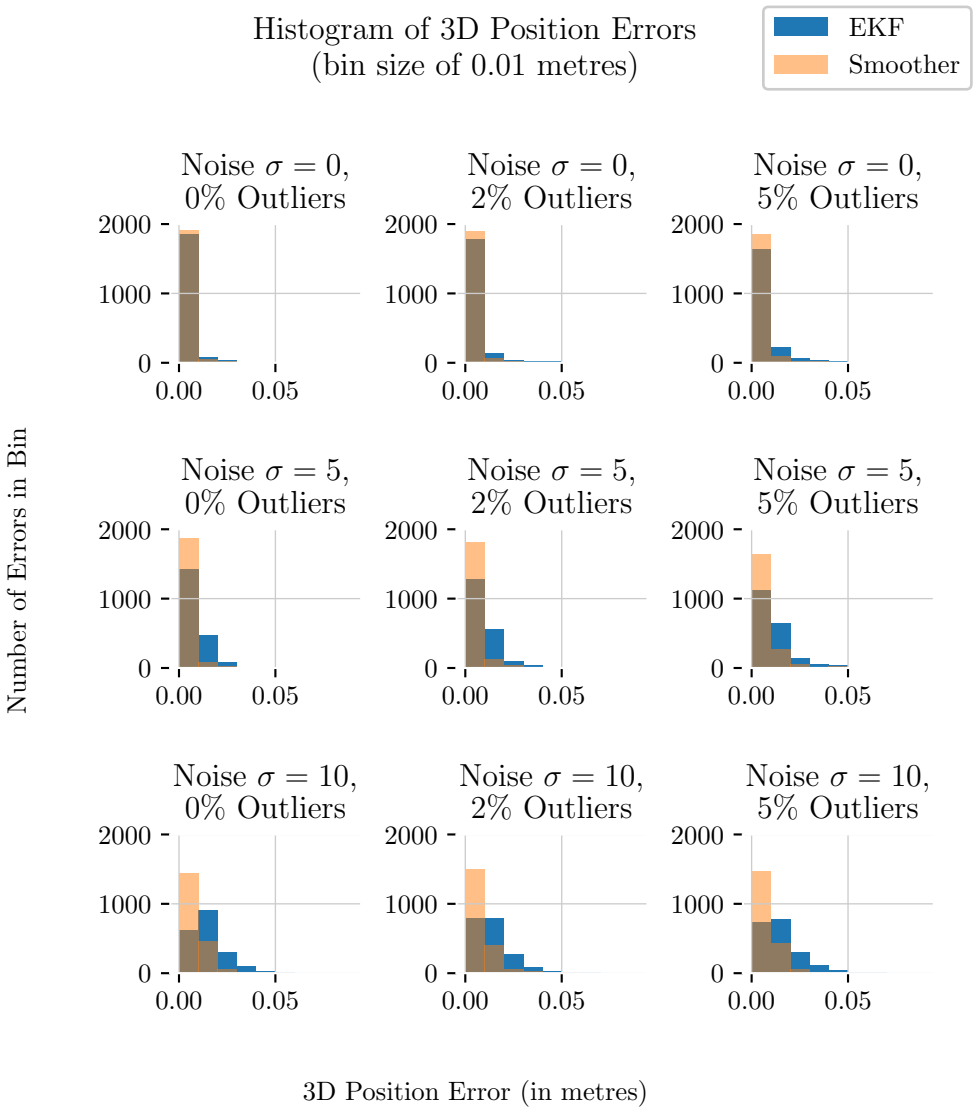

Figure 5.1: Histogram of the 3D position errors

For the model-based approaches, it was possible to test how closely the states from the EKF matched the states of the model used for simulation. Since the model used was the same, if there were no errors in estimating the states, the states would be the same as those from the simulation.

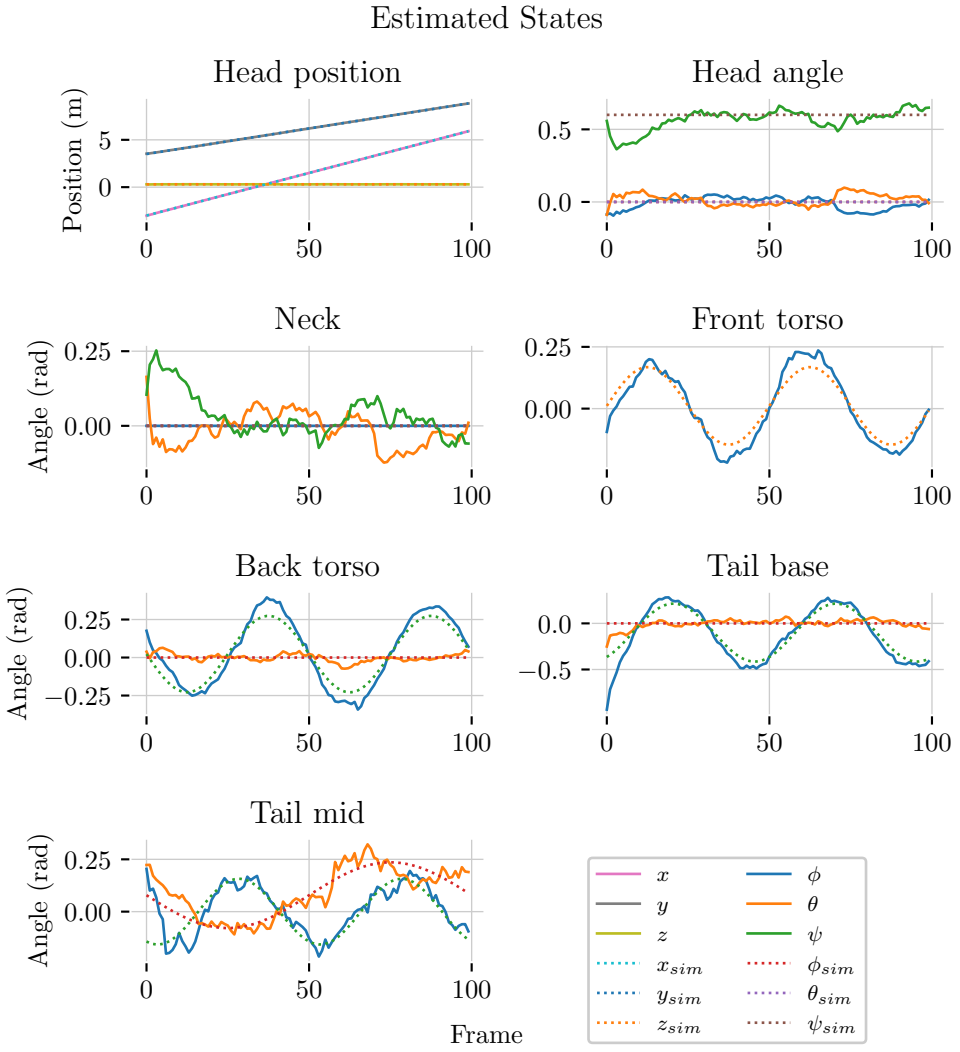

Figure 5.2: Comparison of the states from the EKF (before the Kalman smoother was run) and the states from the simulation for test 5 denoted by the *sim* subscript in the legend

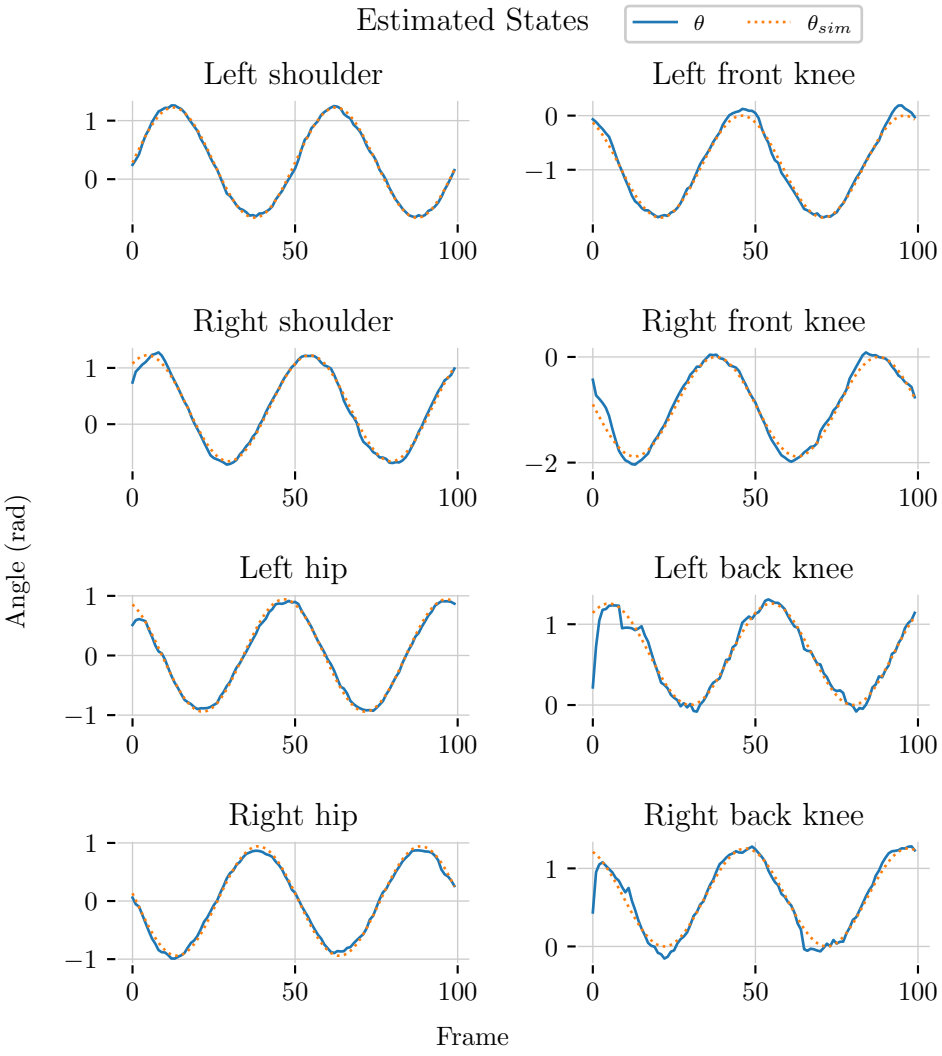

Figure 5.3: Comparison of the states from the EKF (before the Kalman smoother was run) and the states from the simulation for test 5 denoted by the *sim* subscript in the legend

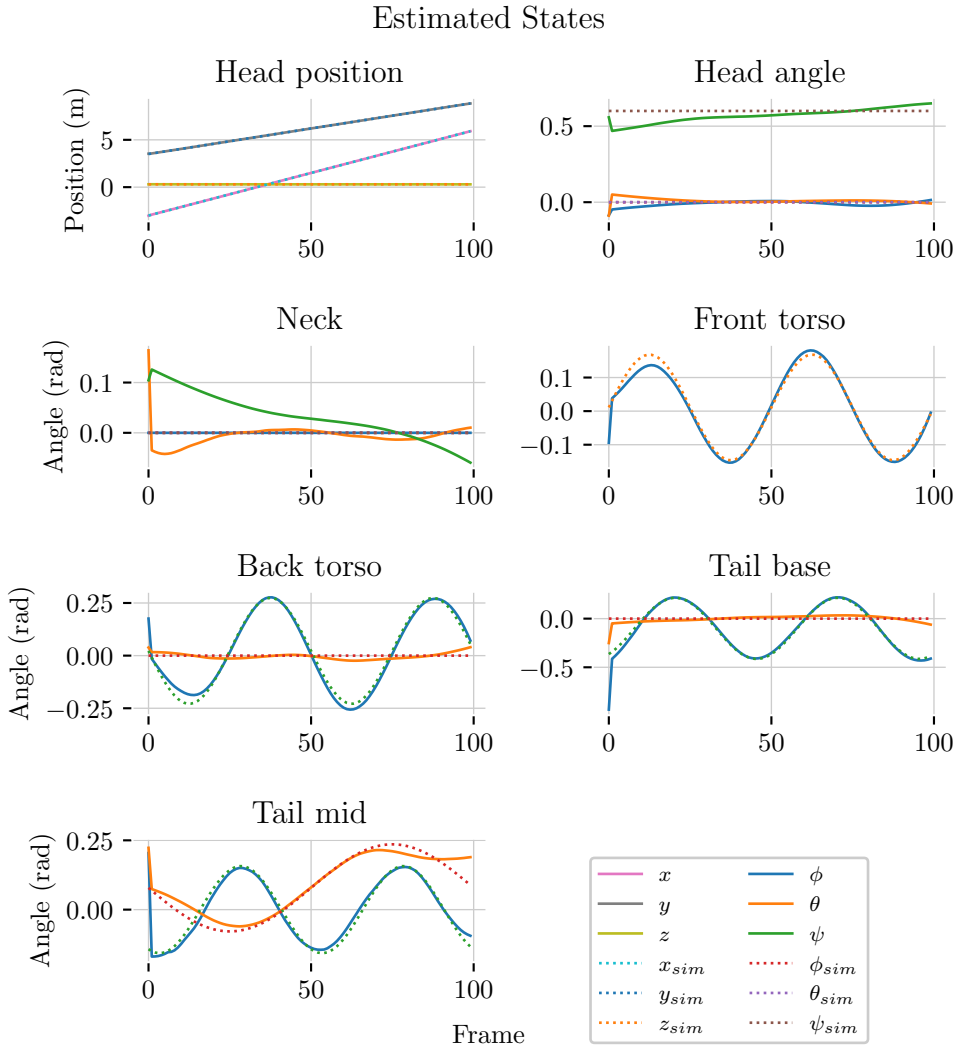

Figure 5.4: Comparison of the states from the EKF (after the Kalman smoother was run) and the states from the simulation for test 5 denoted by the *sim* subscript in the legend

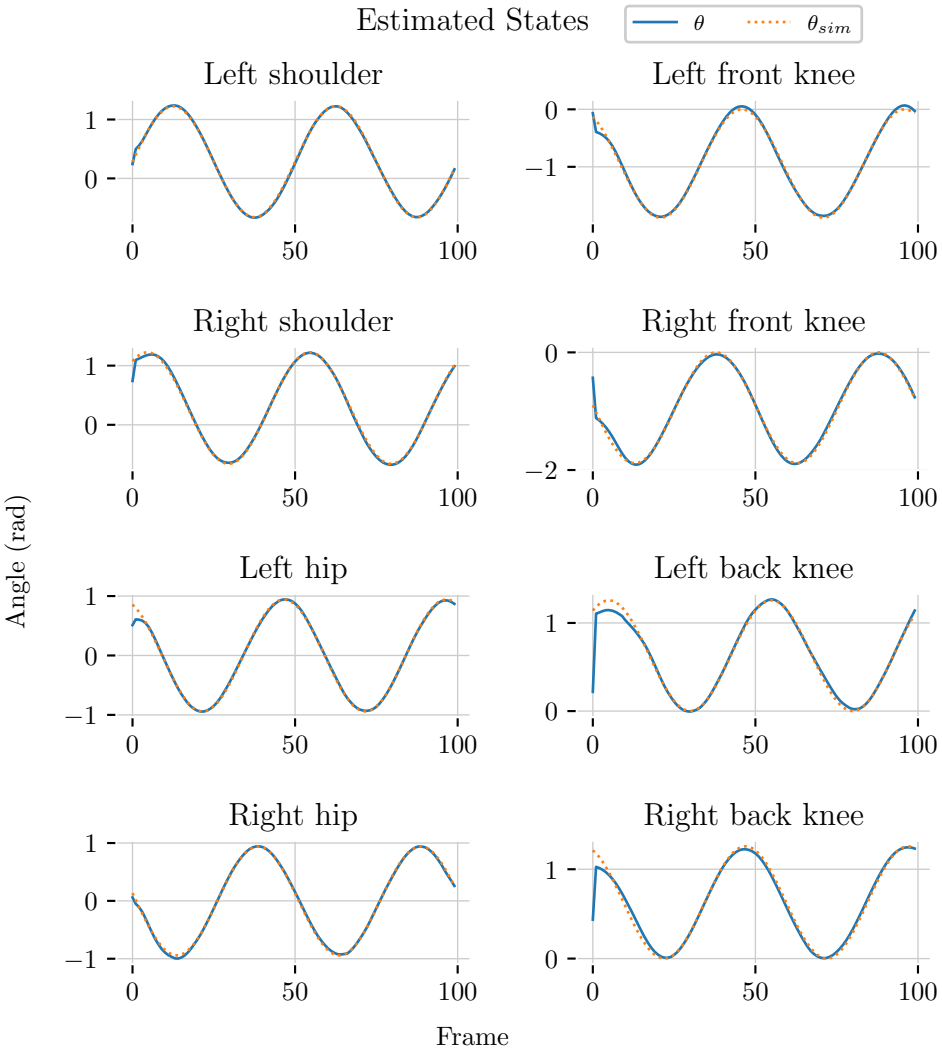

Figure 5.5: Comparison of the states from the EKF (after the Kalman smoother was run) and the states from the simulation for test 5 denoted by the *sim* subscript in the legend

5.4 Real Results

In order to visualise the results for the real cheetah run and dive, the states of the cheetah model were plotted. The results are shown below.

5.4.1 Cheetah Run

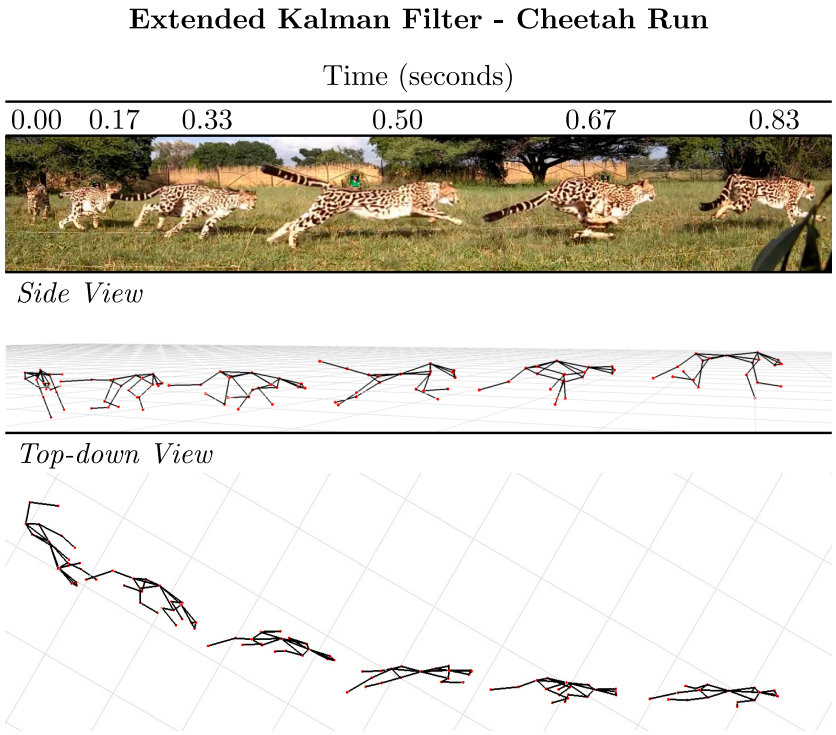

Figure 5.6: Plot of the EKF trajectory estimation for the cheetah run

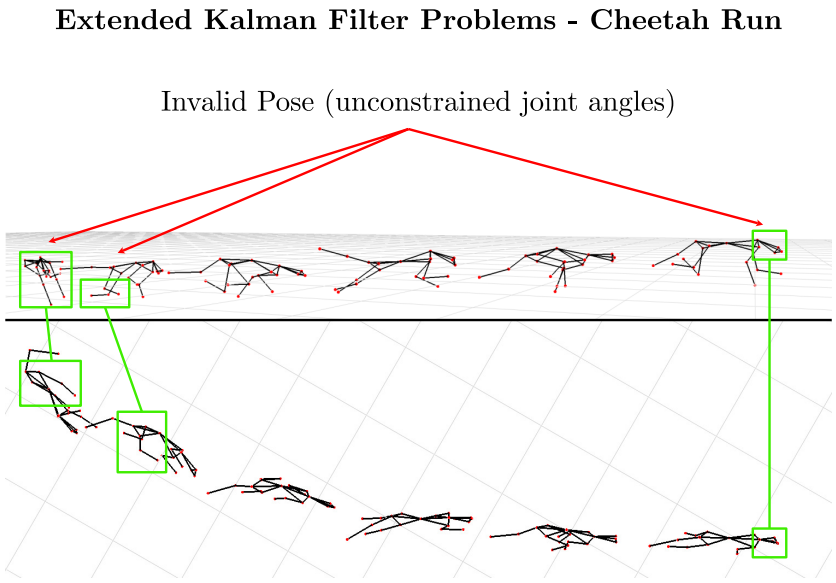

Figure 5.7: Errors in the EKF trajectory estimation for the cheetah run

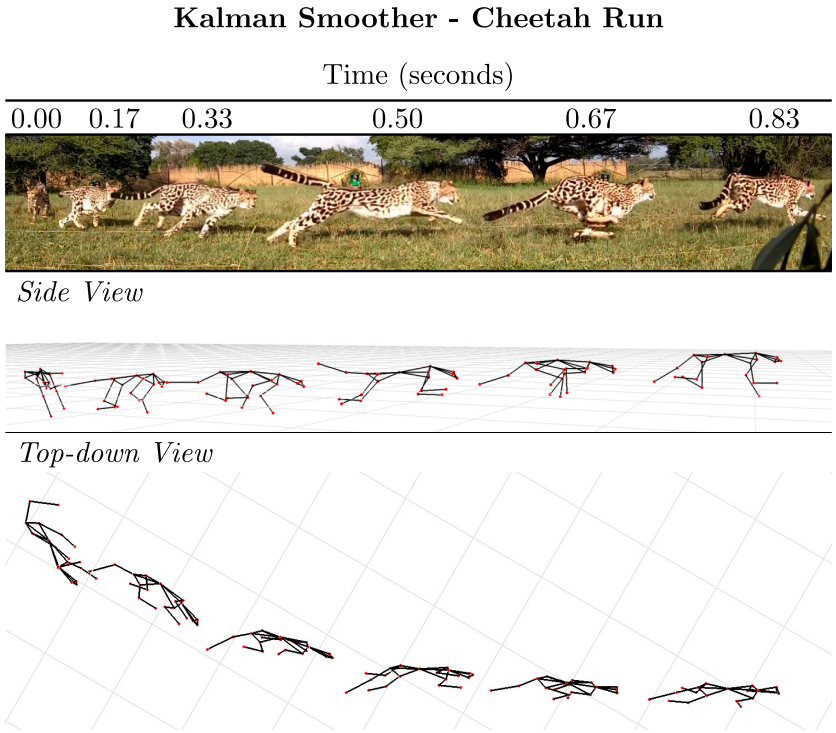

Figure 5.8: Plot of the Kalman smoother trajectory estimation for the cheetah run

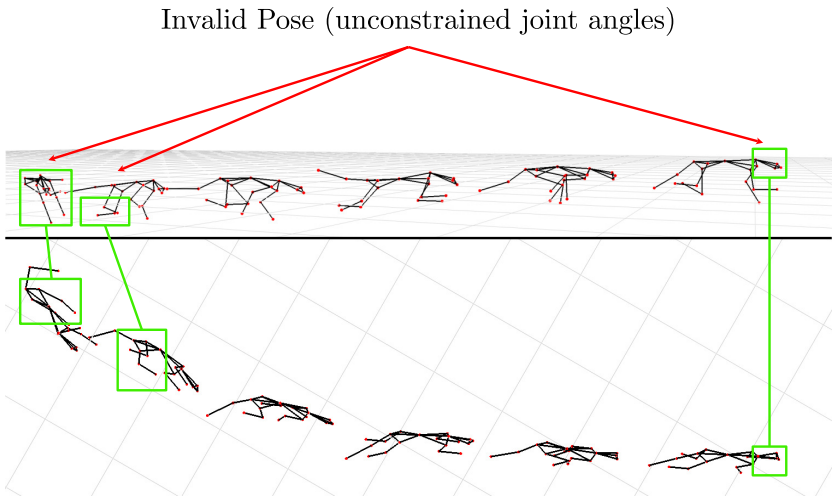

Figure 5.9: Errors in the Kalman smoother trajectory estimation for the cheetah run

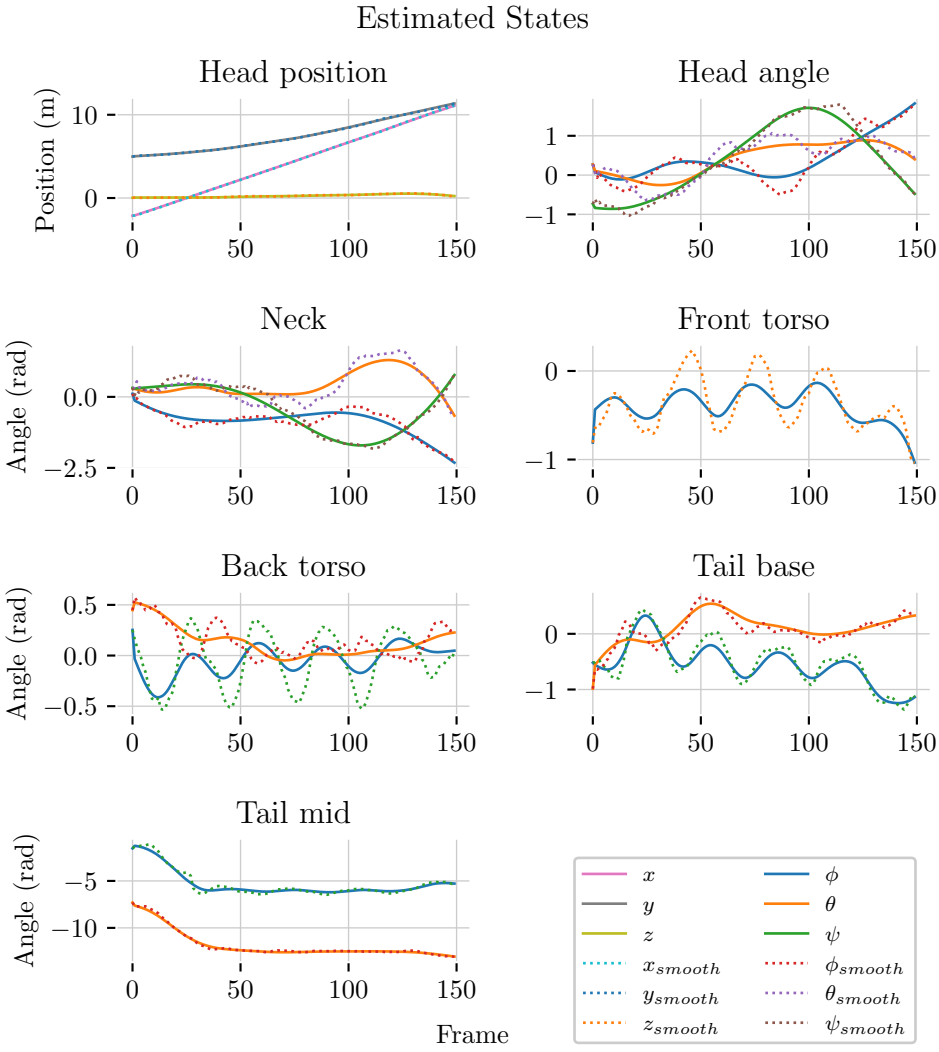

Figure 5.10: States estimated from a real cheetah run using the EKF and Kalman smoother denoted by the *smooth* subscript in the legend

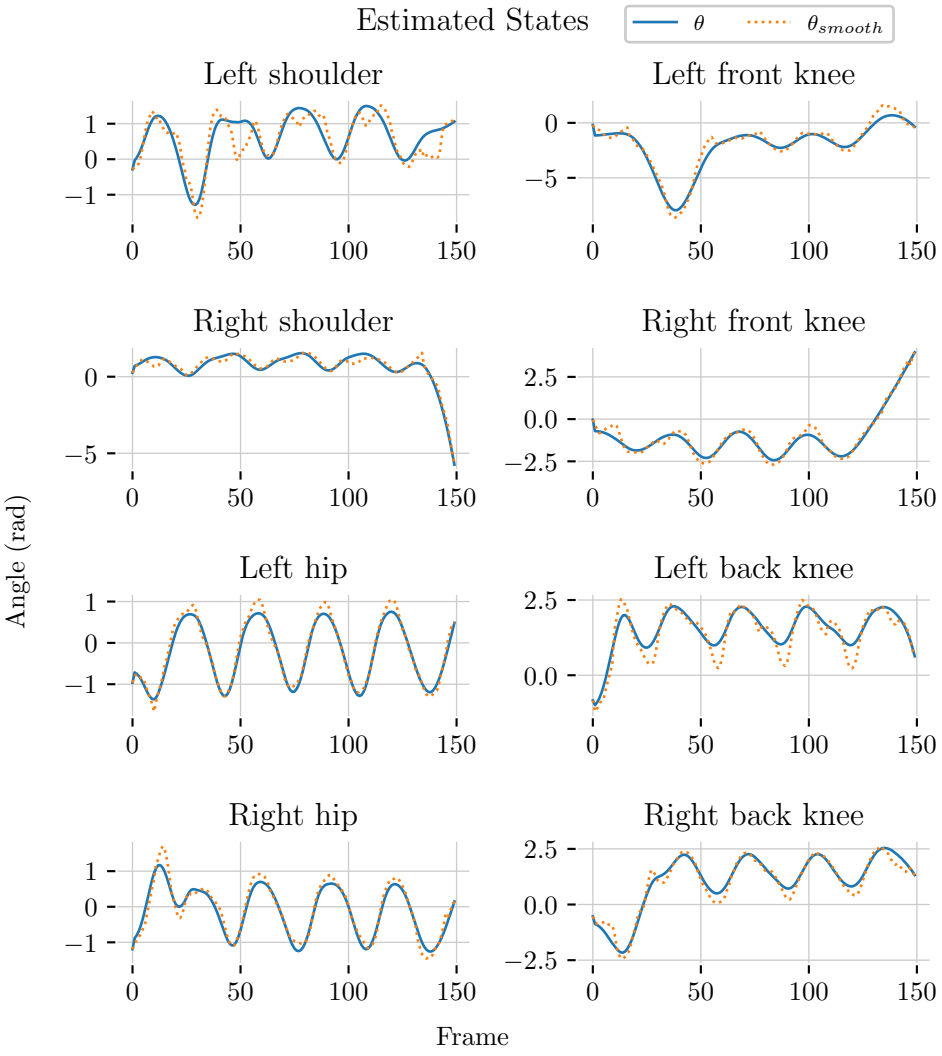

Figure 5.11: States estimated from a real cheetah run using the EKF and Kalman smoother denoted by the *smooth* subscript in the legend

5.4.2 Cheetah Dive

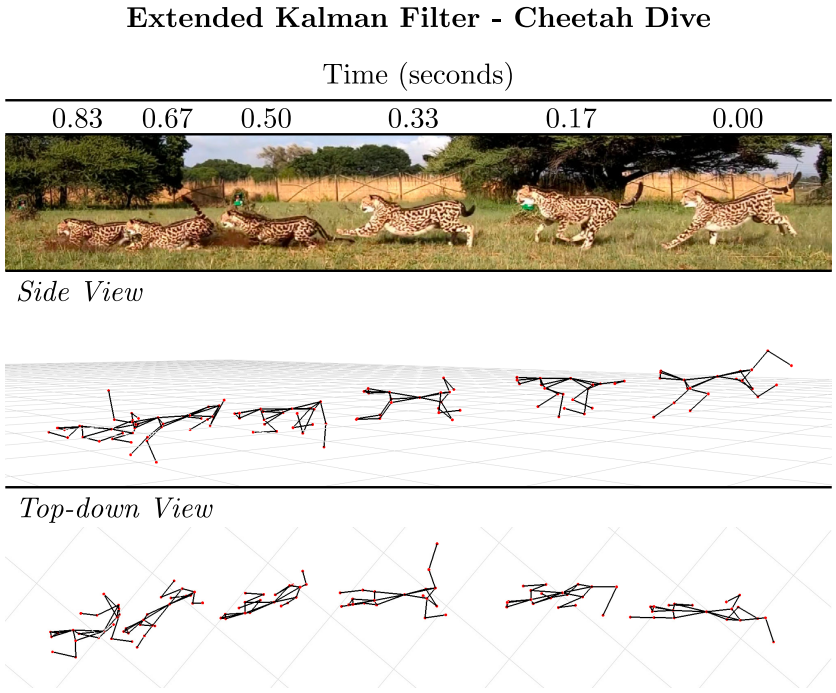

Figure 5.12: Plot of the EKF trajectory estimation for the cheetah dive

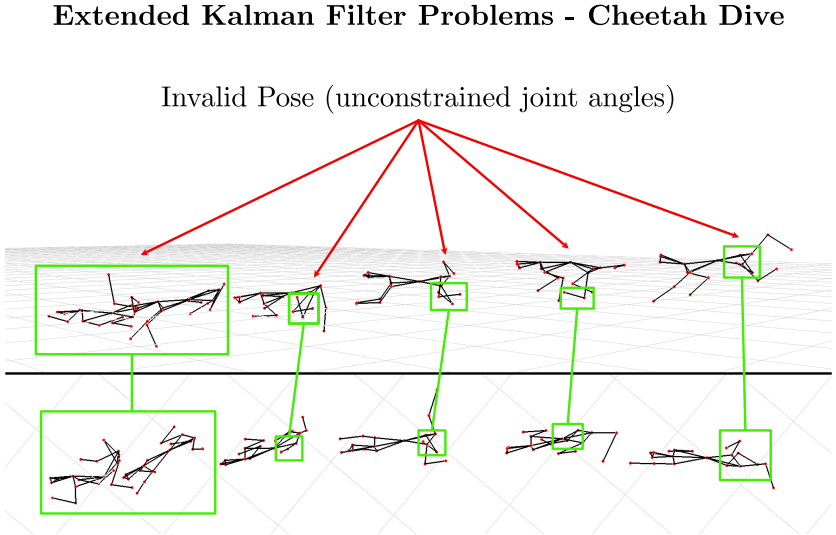

Figure 5.13: Errors in the EKF trajectory estimation for the cheetah dive

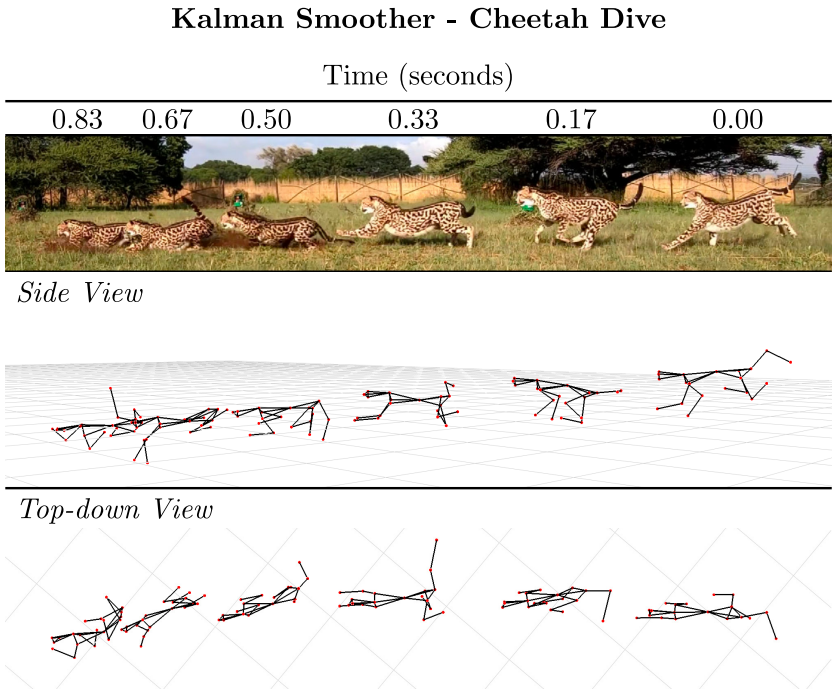

Figure 5.14: Plot of the Kalman smoother trajectory estimation for the cheetah dive

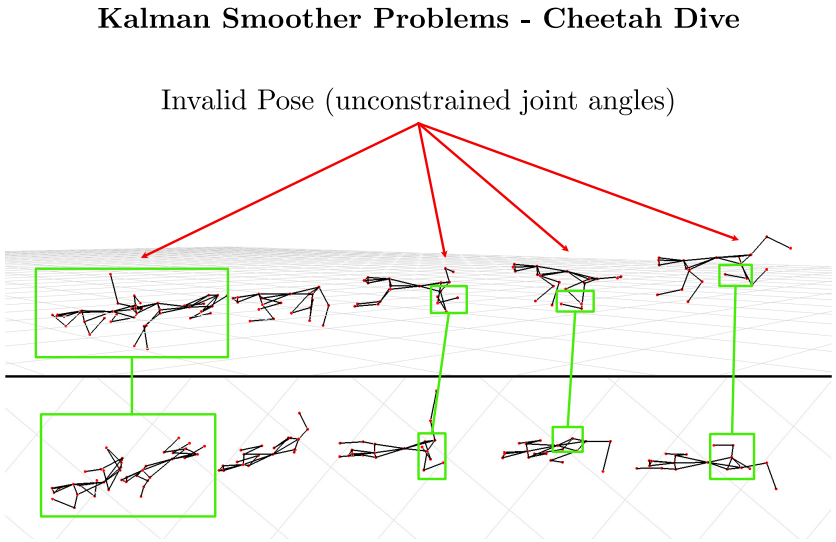

Figure 5.15: Errors in the Kalman smoother trajectory estimation for the cheetah dive

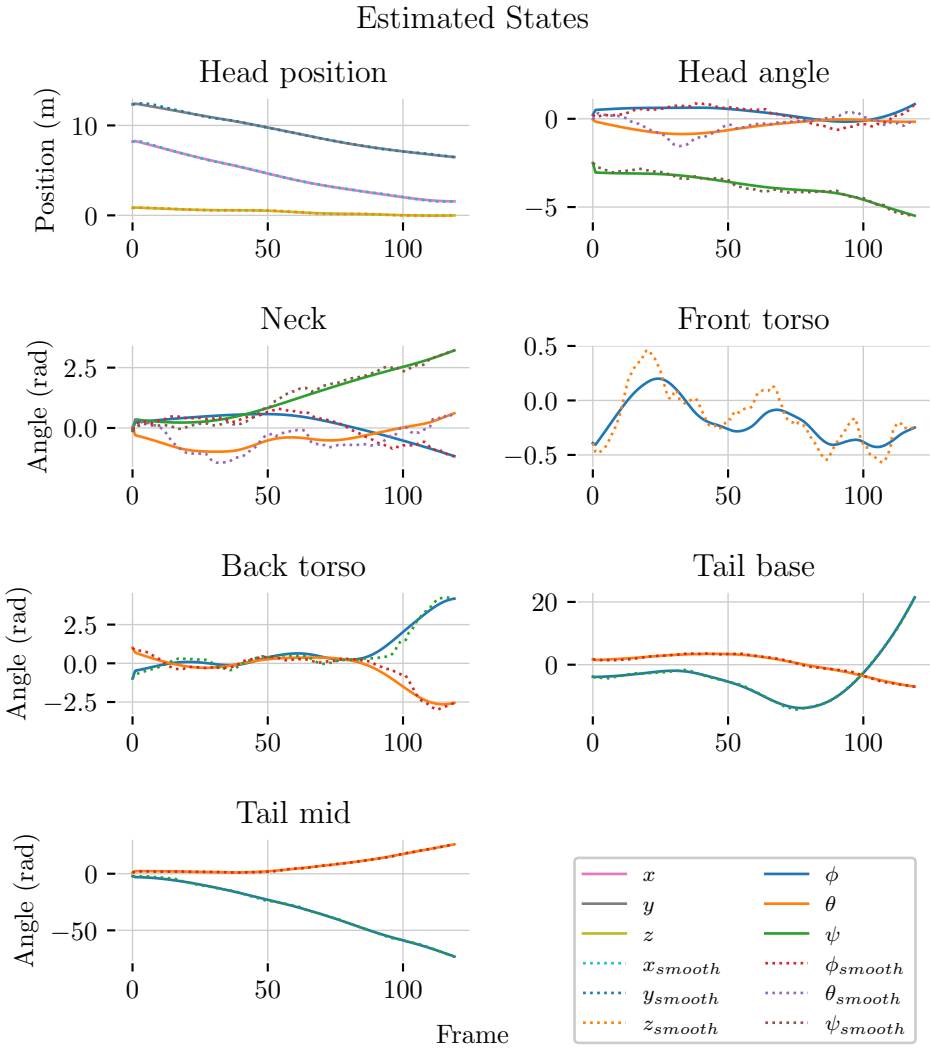

Figure 5.16: States estimated from a real cheetah run using the EKF and Kalman smoother denoted by the *smooth* subscript in the legend

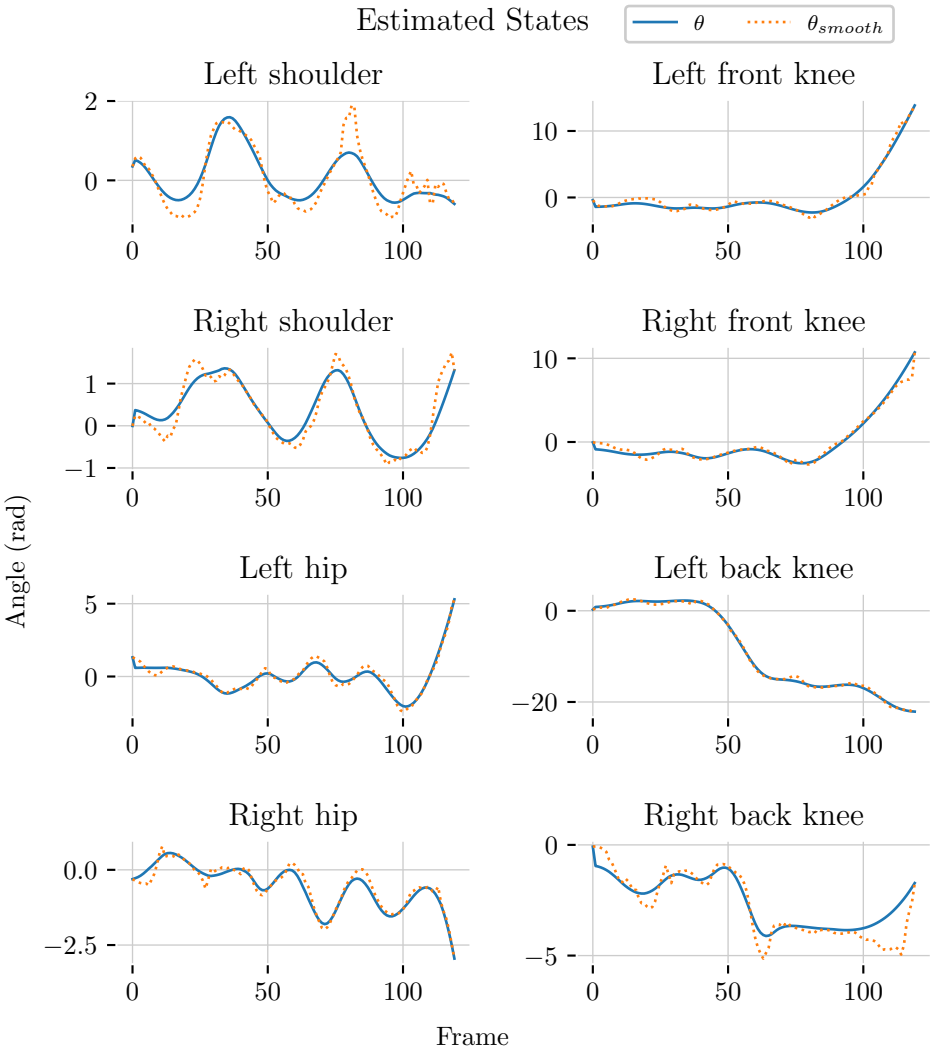

Figure 5.17: States estimated from a real cheetah run using the EKF and Kalman smoother denoted by the *smooth* subscript in the legend

5.5 Discussion

From the results, it was immediately evident that the introduction of a kinematic model reduced the spread of the 3D data. It was expected that the model would reduce the effect of outliers when compared to the SBA. The histogram plots show that the EKF and Kalman smoother performed sig-

nificantly better than the SBA. For both the EKF and Kalman smoother the standard deviation of the 3D error ranged between roughly 0.5cm to 1.5cm, while the MAD remained less than 1cm for all of the tests. For the cases where there was no noise or outliers, there were still small errors, and this was attributed to the initialisation of the EKF and sub-optimal process covariance matrices.

The EKF and Kalman smoother made significant improvements over the SBA; however, several problems occurred when being used for estimation on real cheetah data. From the real data, the cheetah run produced good results (visually); however, problems were experienced with the cheetah dive. Firstly, while the cheetah was constrained to a kinematic model, there was no easy way to constrain joint angles. This resulted in joints rotating right around when measurements were missing due to the constant acceleration model. As could be seen from the cheetah dive, the EKF diverged. This was due to a large number of incorrect measurements caused by the occlusion of markers when the cheetah dived onto the ground. One possible solution that was thought of later was to use a damped model where joints would decelerate in the absence of measurements. Another solution may have been to use a control input which would actively try to restore the cheetah to a "neutral" configuration [11]. Another issue due to the lack of constraints on joint angles was that the head often rotated around to face backwards. As the head was structurally similar when facing in the forwards or reverse direction, the EKF was often able to fit the data better by rotating the head around.

Compared to the SBA, the "jitter" effect between time steps was no longer a problem. In addition, a full cheetah pose was now obtained at each time step. However, it was essential to ensure that the cheetah model had constraints on the angles of joints to prevent the issues mentioned above. This was the primary motivating factor behind the final trajectory estimation method. Another issue that was less frequent, but did occur when analysing real data, was the divergence of the EKF. These two issues motivated the testing of a further method for trajectory estimation, which would constrain the angles to their natural ranges and aim to prevent divergence.

Chapter 6

Full Trajectory Optimisation

While the EKF significantly improved the trajectory estimation techniques when compared to the SBA, there were still areas where improvements could be made. In order to prevent divergence, it was decided that the entire trajectory be optimised simultaneously as there was no need to estimate the trajectory in real-time. Additionally, we wanted to impose constraints on the joints of the cheetah as in some instances with the EKF the joints would bend in the wrong direction or perform a full revolution if measurements were missing.

In order to do this, an optimisation problem was set up where we simultaneously estimated the 3D pose of the cheetah for each frame. The goal of the optimisation problem was to minimise a cost function consisting of two smaller cost functions. The first was a *measurement cost* based off the reprojection error of the 3D points and the second, a *model cost* based off the noise from a constant acceleration model (which was the same as the model used in the EKF).

6.1 Design

6.1.1 Parameters and Variables

Before beginning the design, it is useful to describe a set of parameters and variables which are used in the trajectory estimation. Parameters are listed in Table 6.1.

Parameter	Description
n	Number of time steps in the trajectory
p	Number of pose parameters (24)
c	Number of cameras (6)
m	Number of markers on the cheetah (20)
Δt	The time interval between time steps

Table 6.1: Parameters used for the trajectory estimation

A trajectory, $\mathbf{x}$, consisted of a set of pose parameters at each time step. These pose parameters are the positions and angles, $[x, y, z, \phi_1 \dots \theta_{14}]$, which allowed us to define the 3D marker positions of the cheetah. The first and second order time derivatives of $\mathbf{x}$, $\dot{\mathbf{x}}$ and $\ddot{\mathbf{x}}$, described the "velocities" and "accelerations" of the pose parameters respectively. An array, $\mathbf{s}$, contained the 3D marker positions for each time step.

Variables	Shape	Description
$\mathbf{x}$	$n \times p$	Pose parameters across trajectory
$\dot{\mathbf{x}}$	$n \times p$	Time derivative of $\mathbf{x}$
$\ddot{\mathbf{x}}$	$n \times p$	Time derivative of $\dot{\mathbf{x}}$
$\mathbf{s}$	$n \times m \times 3$	3D marker positions
$\mathbf{w}$	$n \times p - 1$	Model noise
$\mathbf{v}$	$n \times c \times m \times 2$	Measurement noise

Table 6.2: Variables to be determined as part of the trajectory estimation

An array, $\mathbf{y}$, contained 2D pixel measurements from DeepLabCut and had shape $n \times c \times m \times 2$.

6.1.2 Structuring the Optimisation Problem

The optimisation problem consisted of a cost function and constraints, similarly to [62]. Constraints were to be split into two groups. *Model constraints* aimed to satisfy a constant acceleration model for the cheetah while also ensuring that the estimated pose of the cheetah was valid throughout the trajectory. *Measurement constraints* aimed to ensure that the estimated poses of the cheetah corresponded to the measurements obtained from DeepLabCut throughout the trajectory.

Model Constraints

Similarly to the EKF, the model constraints between time steps used a constant acceleration model. In reality, the pose parameters did not obey this model and so $\mathbf{w}$ was added to account for the acceleration errors (or disturbances) for each pose parameter between time steps. Implicit Euler integration was used to formulate the equations below.

$$\mathbf{x}_k = \mathbf{x}_{k-1} + \Delta t \dot{\mathbf{x}}_k \quad \text{for } k = \{2, \ldots, n\} \tag{6.1}$$

$$\dot{\mathbf{x}}_k = \dot{\mathbf{x}}_{k-1} + \Delta t \ddot{\mathbf{x}}_k \quad \text{for } k = \{2, \ldots, n\} \tag{6.2}$$

$$\ddot{\mathbf{x}}_k = \ddot{\mathbf{x}}_{k-1} + \mathbf{w}_k \quad \text{for } k = \{2, \ldots, n\} \tag{6.3}$$

In addition to the constraints for the motion model, bound constraints on the pose parameters were also added. This was done to limit the joints to their natural ranges of motion.

Parameter	Minimum (radians)	Maximum (radians)	Description
ϕ_1	$-\pi/6$	$\pi/6$	Head roll
θ_1	$-\pi/6$	$\pi/6$	Head pitch
ϕ_2	$-\pi/6$	$\pi/6$	Neck roll
θ_2	$-\pi/6$	$\pi/6$	Neck pitch
ψ_2	$-\pi/6$	$\pi/6$	Neck yaw
θ_3	$-\pi/6$	$\pi/6$	Front torso pitch
θ_4	$-\pi/6$	$\pi/6$	Back torso pitch
ψ_4	$-\pi/6$	$\pi/6$	Back torso yaw
θ_5	$-2\pi/3$	$2\pi/3$	Tail base pitch
ψ_5	$-2\pi/3$	$2\pi/3$	Tail base yaw
θ_6	$-2\pi/3$	$2\pi/3$	Tail mid pitch
ψ_6	$-2\pi/3$	$2\pi/3$	Tail mid yaw
θ_7	$-\pi/2$	$\pi/2$	Left shoulder pitch
θ_8	$-\pi$	0	Front left knee pitch
θ_9	$-\pi/2$	$\pi/2$	Right shoulder pitch
θ_{10}	$-\pi$	0	Front right knee pitch
θ_{11}	$-\pi/2$	$\pi/2$	Left hip pitch
θ_{12}	0	π	Back left knee pitch
θ_{13}	$-\pi/2$	$\pi/2$	Right hip pitch
θ_{14}	0	π	Back right knee pitch

Table 6.3: Bound constraints for the cheetah pose parameters

Measurement Constraints

The estimated pose parameters needed to be related to the measurements in order to determine how well the estimated pose related to what was seen by the cameras. While it is possible to obtain the 2D measurement estimates directly from a set of pose parameters, this results in large equations which need to be computed many times per iteration of the optimisation. Initially, the optimisation had been set up this way; however, the computers we were running the optimisations on would run out of memory when formulating the problem this way.

In order to simplify the problem of obtaining the measurement estimates, the problem was split into two parts. Firstly, a constraint was set up which related the pose parameters at each time step to a set of 3D marker positions through a function, $\mathbf{f}$.

$$\mathbf{s}_k = \mathbf{f}(\mathbf{x}_k) \quad \text{for } k = \{2, \ldots, n\} \tag{6.4}$$

A second constraint was set up which projected these 3D marker positions to 2D measurement estimates, $\hat{\mathbf{y}}$, for each camera and equated the measurement estimates to the measurements, $\mathbf{y}$, from DeepLabCut. As there would be errors between the measurements and measurement estimates, a variable $\mathbf{v}$ was added to account for this "measurement error". A projection function, $\mathbf{h}$, for each camera was used to project the 3D points in order to obtain the 2D measurement estimates.

$$\mathbf{y}_{i,j} = \mathbf{h_j}(\mathbf{s}_{i,j}) + \mathbf{v}_{i,j} \quad \text{for } i = \{1, \ldots, n\},\ j = \{1, \ldots, c\} \tag{6.5}$$

Cost function

The optimisation was structured as a non-linear program. Given the constraints above, we have a set of measurement errors, $\mathbf{v}$, and acceleration errors, $\mathbf{w}$, which could be used to form a cost function. Both of these errors needed to be normalised in order to be minimised with equal "fairness".

The measurement error, e_{meas}, can be formulated in a way very similar to that of the SBA. Ideally, we want each of the measurement errors to be zero. Unfortunately, because of inaccuracies in the measurements from DeepLabCut, we could expect that there would always be some measurement error, but we wanted to minimise this by minimising the sum of the squares of the error values. In order to reduce the effect of outliers, a re-descending robust cost function was used:

$$C_{robust}(e) = \begin{cases} \frac{e^2}{2} & \text{for} \quad |e| < a \\ a|e| - \frac{a^2}{2} & \text{for} \quad |e| \geq a, \quad |e| < b \\ ab - \frac{a^2}{2} + \frac{a(c-b)^2}{2}\left(1 - \left(\frac{c-|e|}{c-b}\right)^2\right) & \text{for} \quad |e| \geq b, \quad |e| < c \\ ab - \frac{a^2}{2} + \frac{a(c-b)^2}{2} & \text{for} \quad |e| \geq c \end{cases}$$
$$\text{where} \quad a = 3, b = 10, c = 20 \tag{6.6}$$

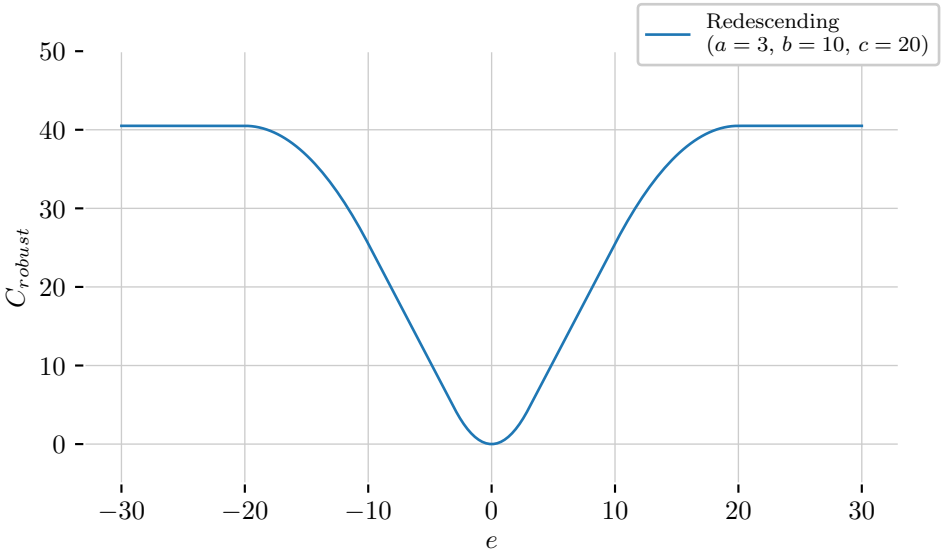

Figure 6.1: Redescending cost function for the normalised measurement errors

The values for a, b and c were chosen to produce Figure 6.1 such that for measurement errors which deviate from 0 by more than 4 standard deviations (equating to 20 pixels), the cost of the outliers is constant. For measurement errors which deviate from 0 between 2 and 4 standard deviations (10 to 20 pixels), the cost of the outliers is linear. For measurement errors which deviate from 0 by less than 2 standard deviations (10 pixels), the cost is quadratic.

The measurement costs were normalised by dividing by the standard deviation of the measurements, $\sigma_{meas} = 5$ pixels, from DeepLabCut.

$$e_{meas} = \sum_{i=1}^{n} \sum_{j=1}^{c} \sum_{k=1}^{m} \sum_{l=1}^{2} C\left(\frac{\mathbf{v}_{i,j,k,l}}{\sigma_{meas}}\right) \tag{6.7}$$

The model error, e_{model}, was the sum of squares of acceleration noise. In order to normalise this noise, the squared acceleration error values were divided by the acceleration variance of the pose parameter they corresponded to. Alternatively, the normalisation can be thought of as dividing by the standard deviation before squaring the value.

$$e_{model} = \sum_{i=1}^{n} \sum_{j=1}^{p} \frac{(\mathbf{w}_{i,j})^2}{\sigma^2_{model\,j}} = \sum_{i=1}^{n} \sum_{j=1}^{p} \left(\frac{\mathbf{w}_{i,j}}{\sigma_{model\,j}}\right)^2 \tag{6.8}$$

The minimisation problem could then simply be formulated as per Equation 6.9 subject to all of the model and measurement constraints.

$$\min_{\mathbf{x},\, \dot{\mathbf{x}},\, \ddot{\mathbf{x}},\, \mathbf{s},\, \mathbf{w},\, \mathbf{v}} \quad e_{meas} + e_{model} \tag{6.9}$$

6.1.3 Implementation

The optimisation was implemented in Python using Pyomo [63], [64] and IPOPT [65]. IPOPT, an interior point optimiser, was compiled from source code to take advantage of the speed benefits from the Intel Math Kernel Library (MKL) and the family of Harwell Subroutine Library (HSL) linear solvers. IPOPT was configured to use the MA86 linear solver for this problem in order to to take advantage of its parallel processing capabilities.

Similarly to the EKF, for each run, the position and orientation of the cheetah were estimated using an initial set of triangulated 3D points. This was done by plotting the estimated x, y and z coordinates for the nose over the trajectory and fitting a straight line to each of the x, y and z trajectories. The initial x, y and z positions for the cheetah trajectory were obtained using the fitted line, while the yaw of the cheetah was determined from the slope of the fitted lines for the x and y coordinates. The initial x, y and z velocity for the cheetah was obtained using the slope of the fitted lines for the x, y and z coordinates of the nose. All of the other states were initialised to zero (such that the cheetah was in a valid, upright pose).

When running the tests, it was found that the simulated trajectory took 200 iterations to solve, taking 9 minutes to run on a desktop PC with a quad-core Intel Core i7 6700 CPU. For the real data, the cheetah run took 261 iterations to solve (12 minutes), and the cheetah dive took 481 iterations to solve (22 minutes).

6.2 Simulated Data Results

Noise σ (pixels)	Outliers (%)	μ (m)	σ (m)	Median (m)	MAD (m)
0	0	3.15e-03	3.18e-03	2.52e-03	5.93e-04
0	2	3.63e-03	3.17e-03	2.99e-03	9.83e-04
0	5	4.20e-03	3.32e-03	3.45e-03	1.28e-03
5	0	5.01e-03	3.98e-03	3.94e-03	1.99e-03
5	2	5.31e-03	4.02e-03	4.24e-03	2.35e-03
5	5	5.54e-03	4.31e-03	4.36e-03	2.63e-03
10	0	9.25e-03	6.13e-03	7.54e-03	4.68e-03
10	2	9.25e-03	6.38e-03	7.46e-03	4.61e-03
10	5	9.78e-03	6.02e-03	8.23e-03	4.81e-03

Table 6.4: 3D position error's mean μ, standard deviation σ, median and median absolute deviation (MAD) from the trajectory optimisation

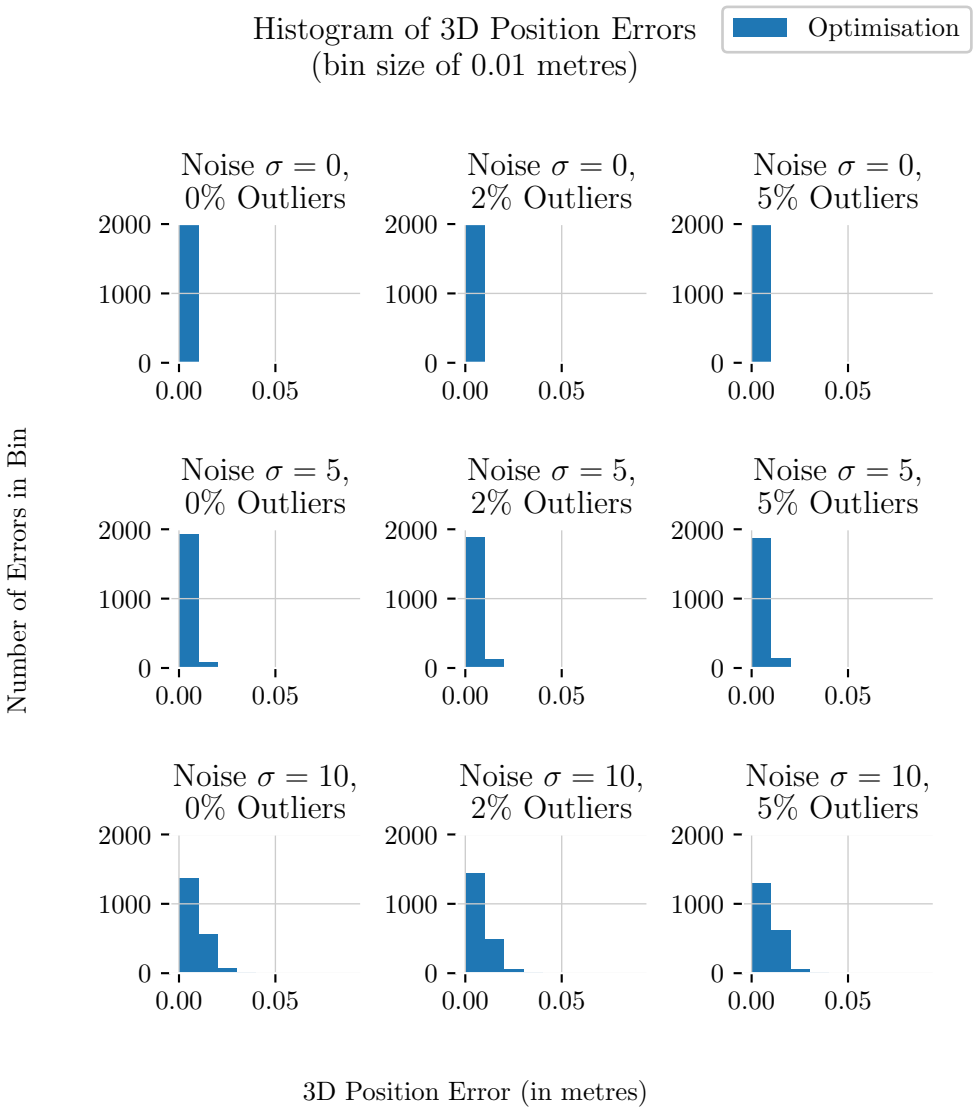

Figure 6.2: Histogram of the 3D position errors

Similarly to the EKF, graphs comparing the states of the simulated model and the full trajectory estimation were compared.

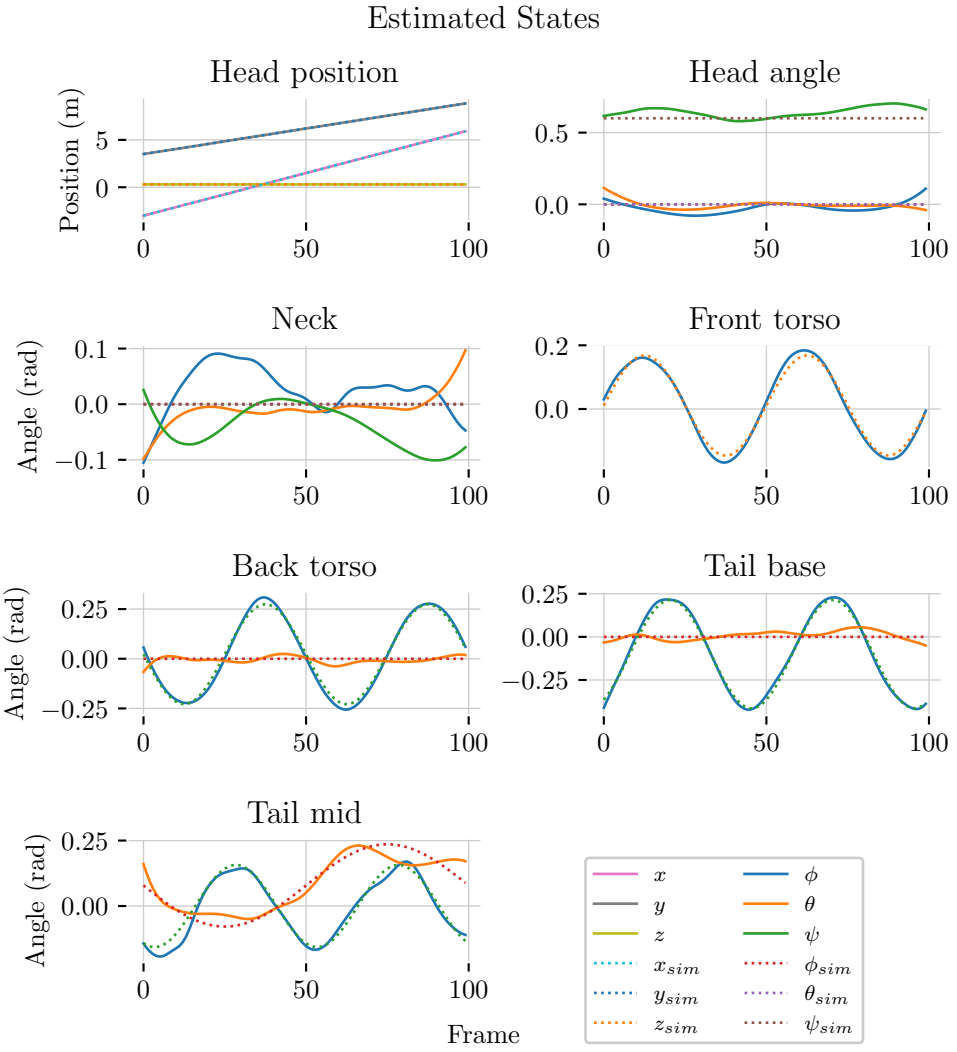

Figure 6.3: Comparison of the states from the full trajectory estimation and the states from the simulation for test 5

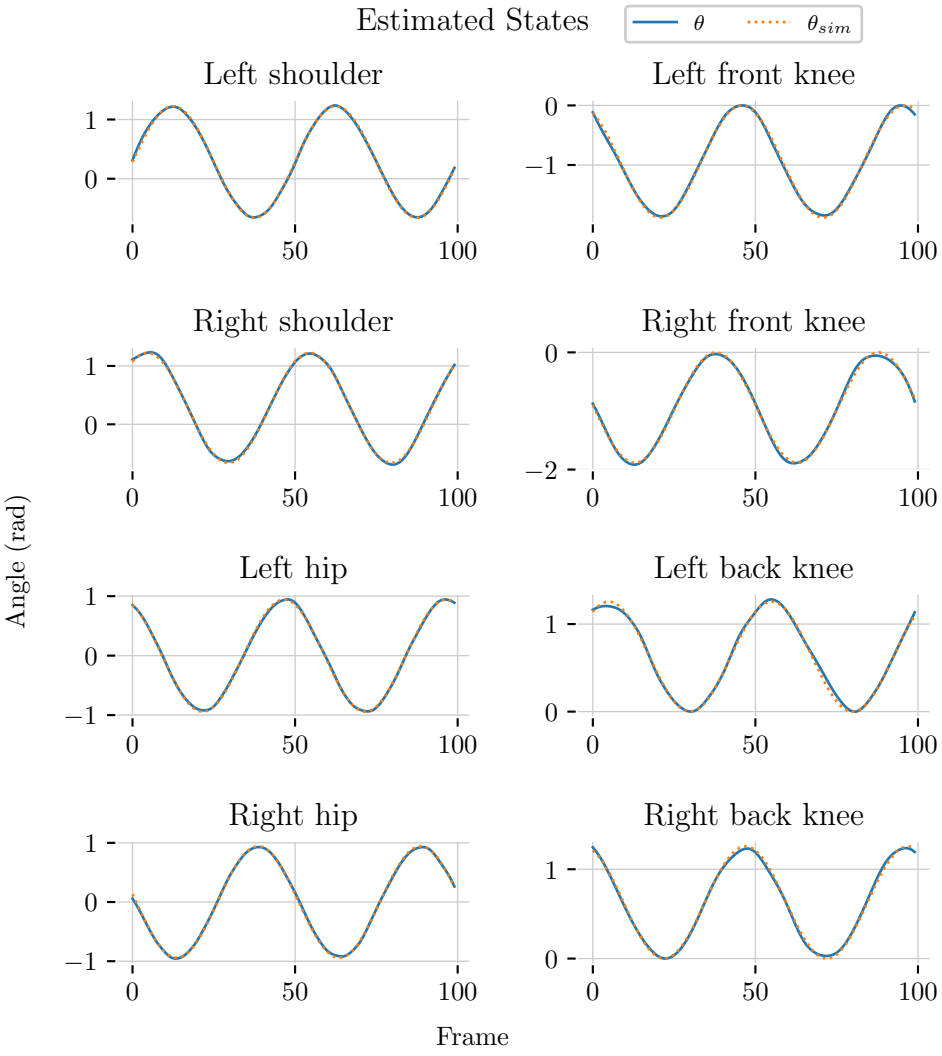

Figure 6.4: Comparison of the states from the full trajectory estimation and the states from the simulation for test 5

6.3 Real Results

In order to visualise the results for a real cheetah run and dive, the states of the cheetah model were plotted. The results are shown below.

6.3.1 Cheetah Run

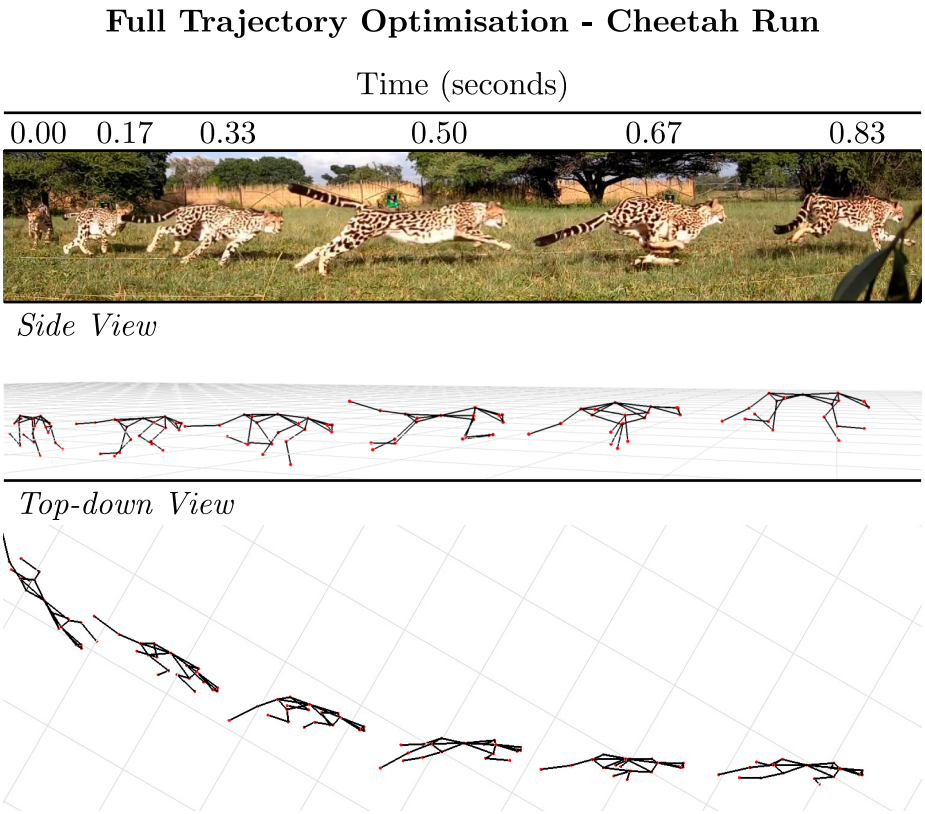

Figure 6.5: Plot of the full optimisation trajectory estimation for the cheetah run

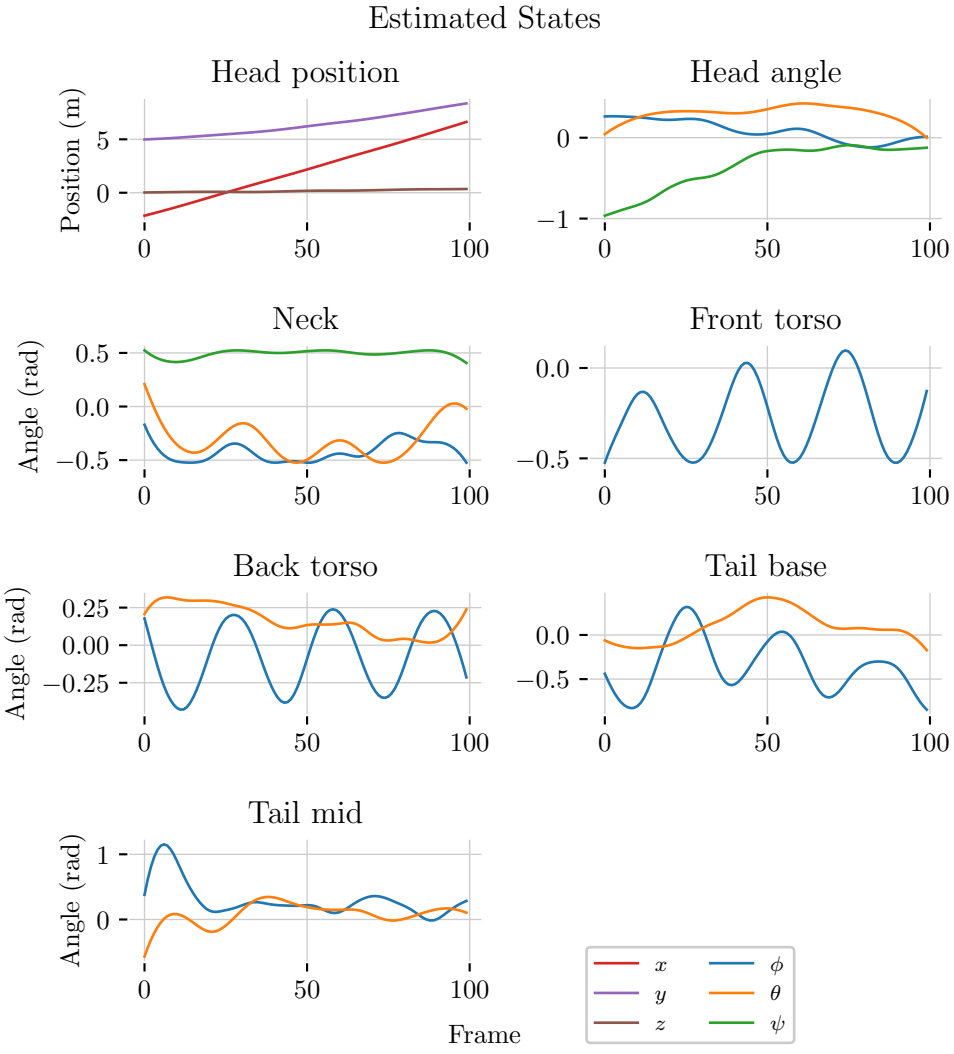

Figure 6.6: States estimated from a real cheetah run from the simultaneous trajectory estimation

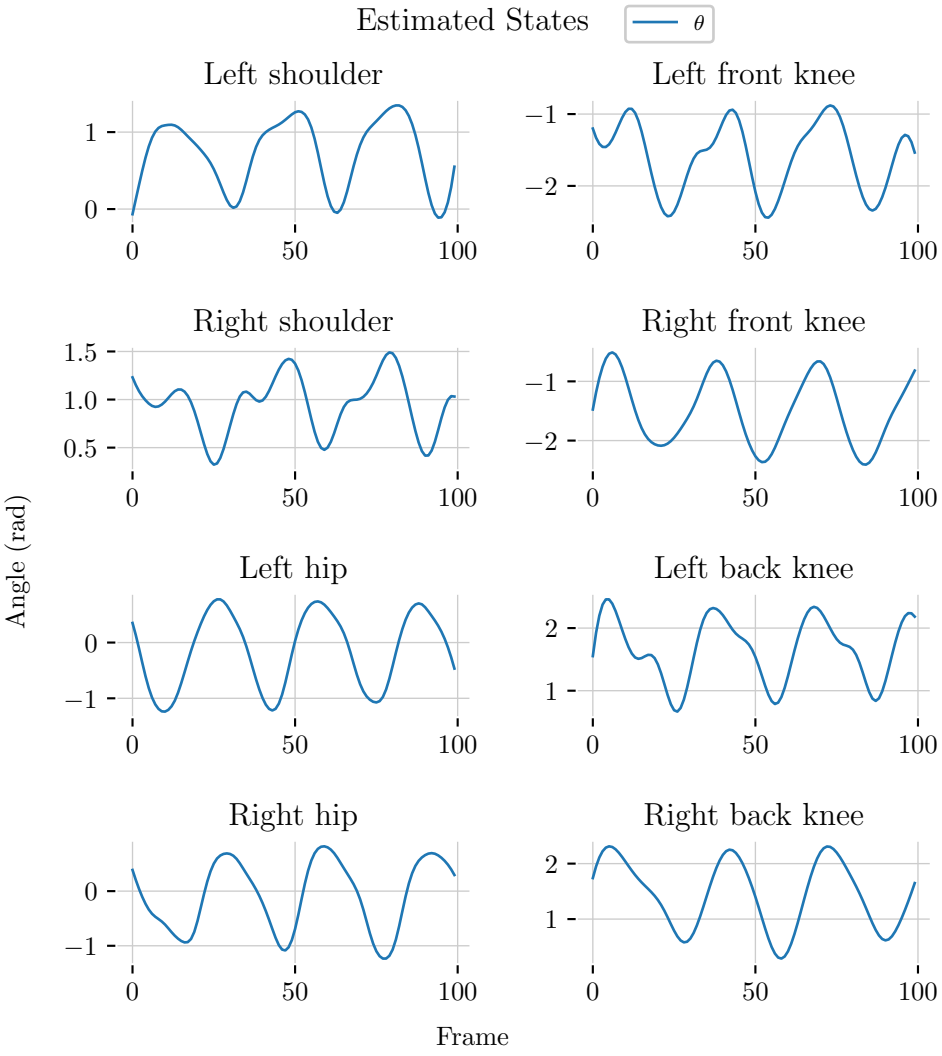

Figure 6.7: States estimated from a real cheetah run from the simultaneous trajectory estimation

6.3.2 Cheetah Dive

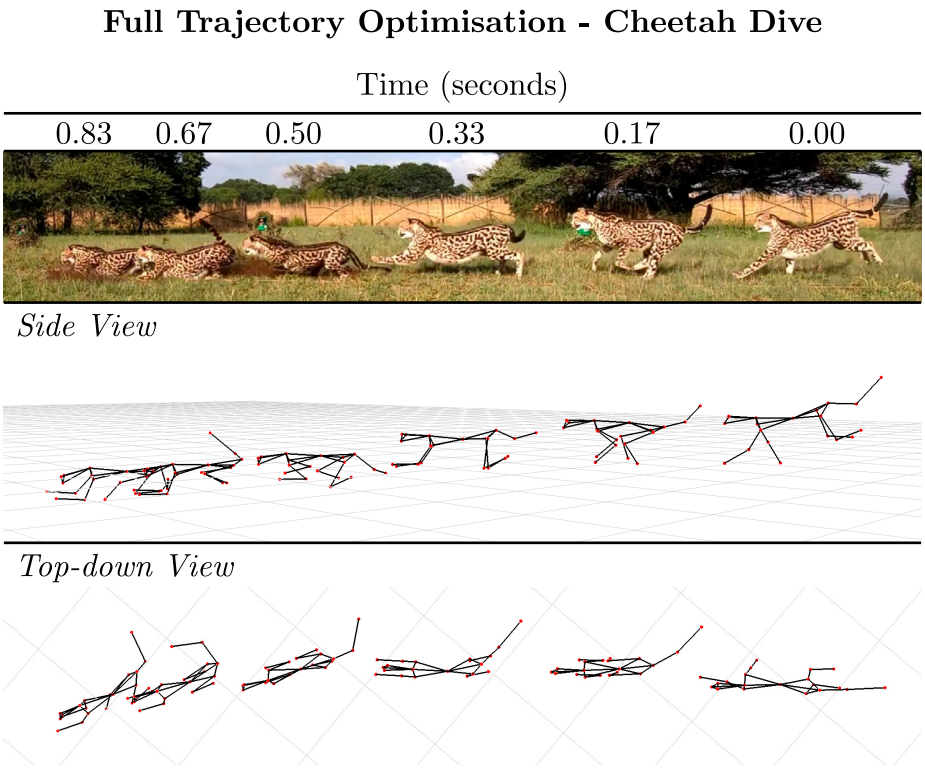

Figure 6.8: Plot of the full optimisation trajectory estimation for the cheetah dive

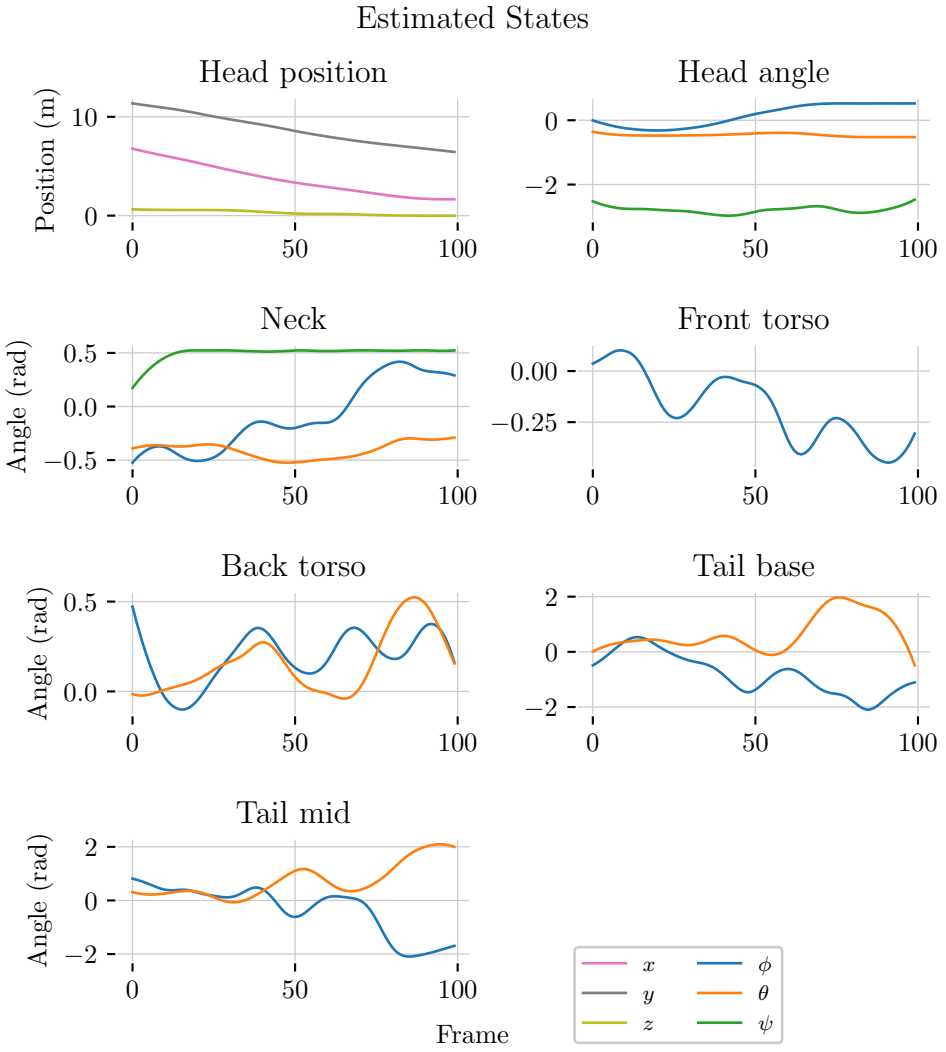

Figure 6.9: States estimated from a real cheetah run from the simultaneous trajectory estimation

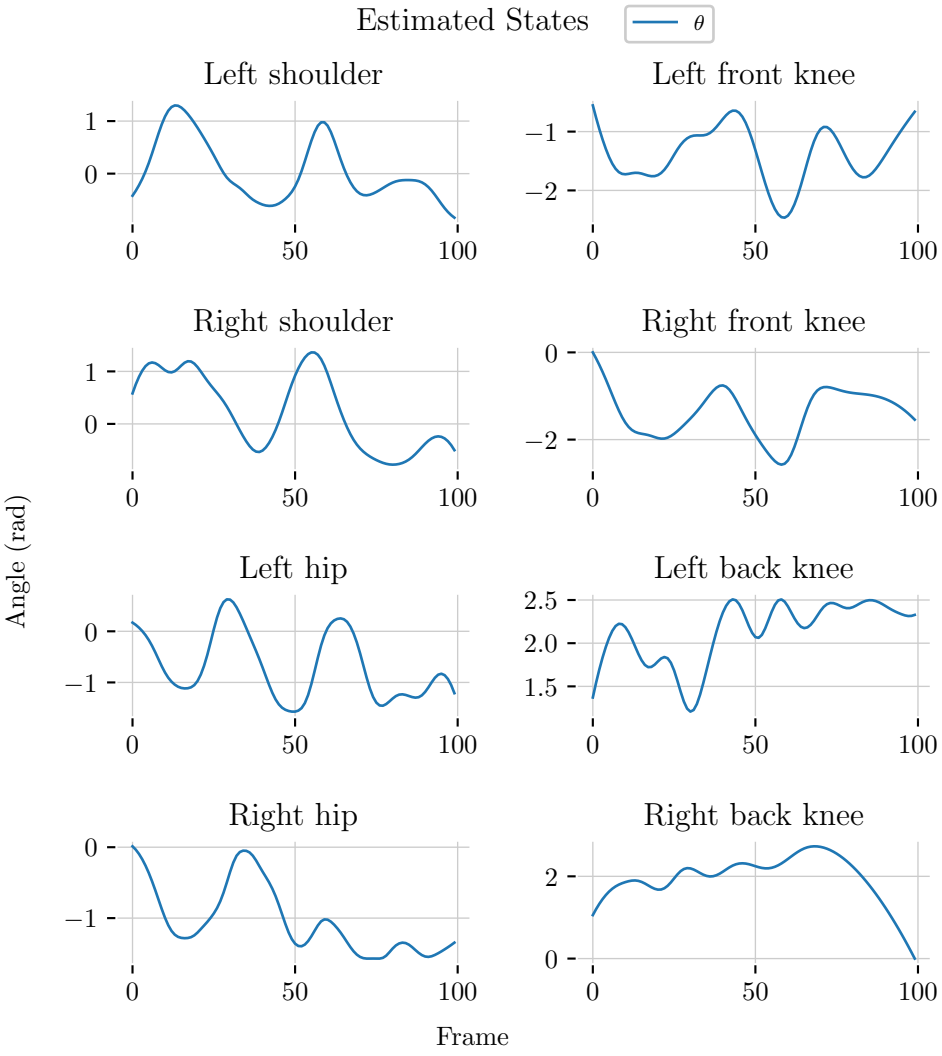

Figure 6.10: States estimated from a real cheetah run from the simultaneous trajectory estimation

6.4 Discussion

It can be seen from the results above that the full trajectory optimisation provides good results similar to those of the EKF and Kalman smoother. While the accuracy statistics are similar to the Kalman smoother, it is important to note that the extra constraints placed on the joints of the model ensure

that the cheetah's pose is always feasible. Also, the issue of angles wrapping around for real data was no longer a problem due to these constraints. On both sets of real data, the full trajectory optimisation produced results which appeared to describe the real actions of the cheetah accurately.

It was expected that the model would reduce the effect of outliers when compared to the SBA. The histogram plots show that the full trajectory estimation performed better than the SBA and EKF and was on par with the Kalman smoother. The standard deviation of the 3D error ranged between roughly 0.5cm to 1.5cm, while the MAD remained less than 1cm for all of the tests.

The improvements over the methods before included removing the "jitter" effect between time steps experienced by the SBA, fitting the data to a kinematic model and constraining the angles to their natural ranges to solve the angle wrapping issue experienced by the EKF. Additionally, since the entire trajectory was optimised over as a whole, the divergence experienced by the EKF was no longer an issue.

It should be noted that when the full trajectory optimisation method was being initially tested, two issues were experienced. The first issue was that the cheetah pose was not being initialised into a valid configuration. This caused the optimiser to get stuck dealing with the constraint violations without minimising the costs for the model or the measurement errors. By initialising the pose parameters all to zero, such that no constraints were violated, the issue was solved. The second issue was that the optimisation did not always converge which we found was due to the cheetah's x, y and z position being initialised to zero. By using triangulated data and performing a linear regression for the position of the nose, the cheetah's x, y and z pose parameters were initialised to be roughly in the right positions. Since implementing these initialisations, the full trajectory optimisation approach has performed well.

Chapter 7

Conclusion

7.1 Discussion

This project has demonstrated the effectiveness of several techniques for estimating the trajectory of a cheetah using 2D marker estimates from video footage. While the tests performed were by no means exhaustive, the data gathered provides a useful overview of the pros and cons of each method.

The SBA, while not providing the best results, was by no means a poor method of estimating the position of the markers. Its use can be motivated by the simplicity of its implementation since a state-based model of a subject is not needed. The SBA can also be used as a tool for the initialisation of other methods. While missing measurements pose a problem as the 3D marker positions cannot be estimated, given that there are no significant gaps in the data, one can use interpolation techniques to estimate the positions of missing points in a trajectory.

The EKF provided good results when a model of the subject was provided. There was a large jump in complexity when implementing the EKF when compared to the SBA; however, this extra complexity led to much better results. It should be noted that the accuracy of the EKF is dependent on the implementation of the model that one uses. The rigid body model of the cheetah is by no means a realistic skeletal model of the cheetah, but it provided a similar enough representation to study the cheetah's locomotion using real data. An advantage of the EKF is that it can be used for real-time estimation. While this was not a requirement for this project, it would be interesting to see a multi-camera pose estimation system for cheetahs which could operate in real-time.

The full trajectory estimation method proved to be an excellent method for analysing the cheetah data. The main two advantages of this method were that the pose parameters could be constrained to their natural ranges and that the 3D marker positions were constrained to the pose of the cheetah. While this method seems to produce good trajectory estimations consistently, certainly more so than the EKF and SBA, the robustness of this method was not explored fully during this project.

Another aspect which was not mentioned in the evaluation of these methods was how long they took to run and the processing resources required. These were not of concern for this project; however, both the SBA and the full trajectory estimation took around 10 minutes to find optimal solutions for a 100 frame trajectory containing both noise and outliers. Additionally, part of the full trajectory estimation includes Pyomo generating the code which IPOPT uses to perform the optimisation, which takes several minutes. The EKF and EKS were both able to process the full trajectory in only a few seconds, which could probably be sped up by restructuring the code. In the case that speed is a requirement, the EKF and EKS are likely the best methods to use.

Ultimately, this project has shown that it is possible to obtain good trajectory estimations using data which is obtained using tools like DeepLabCut. Testing has shown that model-based methods can handle missing measurements, more than double the expected measurement noise and large percentages of outliers. Hopefully, these methods will assist in the research of animals like the cheetah and that more methods for 3D markerless motion capture will become available with the advancements in deep learning and pattern recognition.

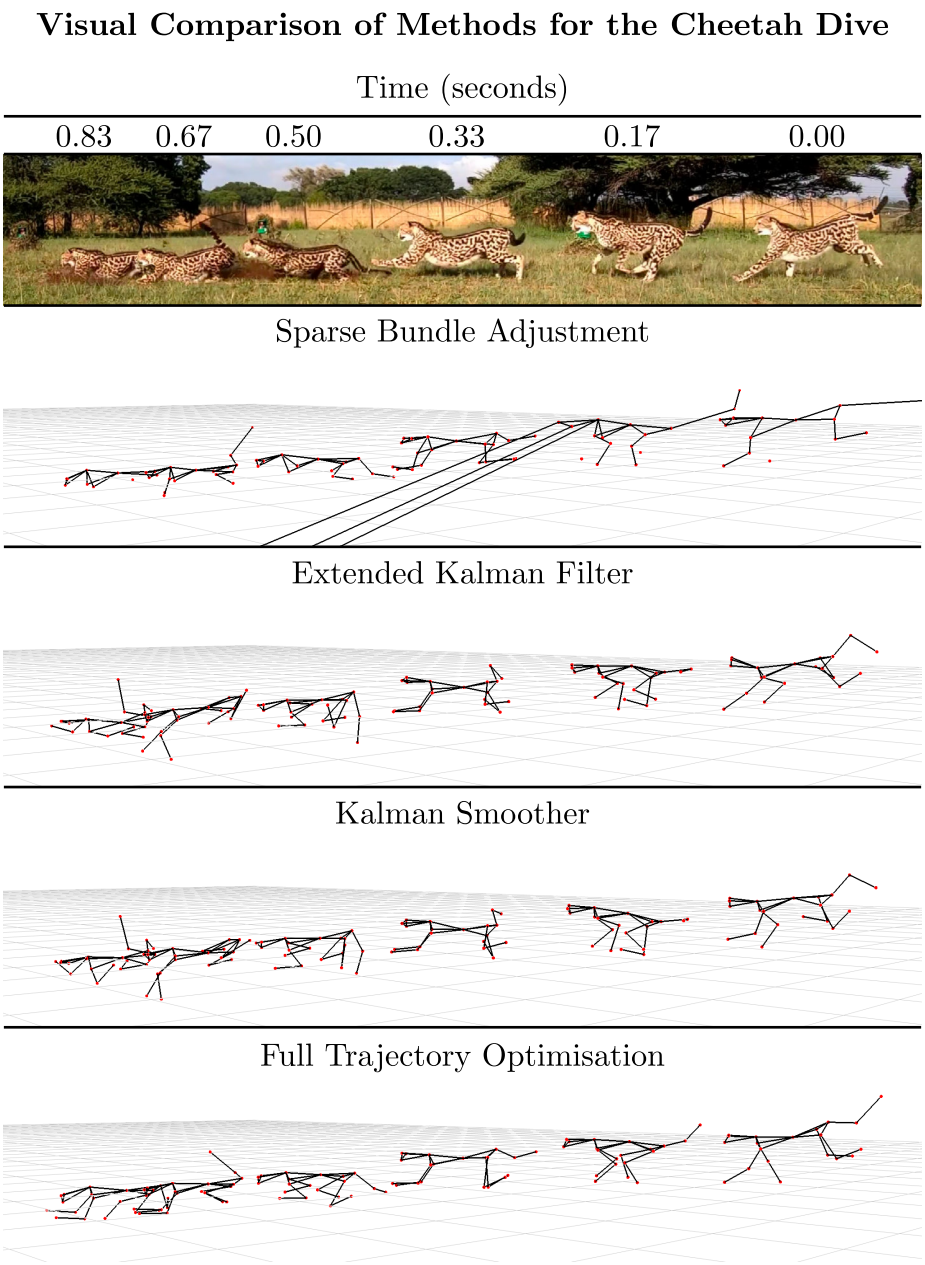

Figure 7.1: Visual comparison of the different trajectory estimation methods for the cheetah dive

7.2 Future Work

Building on from this project, there are many areas where future work could be performed. These include further exploration of the trajectory estimation

methods in this project as well as looking at other techniques for markerless pose estimation.

In looking to improve upon the model-based estimation approaches, better methods for estimating the process covariance matrices for the EKF and the variance weightings for the pose parameters in the full trajectory estimation. Barratt et al. [66] propose using a machine learning approach to find these parameters.

A survey by Simon et al. [67] suggests methods for constraining the states when performing non-linear state estimation including placing soft constraints on the states in a Kalman filter and using other methods such as moving horizon estimation (MHE). These are worth exploring as they are fast to compute and may allow for accurate pose estimation in a real-time system.

Regarding testing, far more tests need to be performed in order to establish which methods work better than others in different circumstances. For instance, in a two-camera setup, one cannot rely on measurements from other pairs of cameras, and so model-based approaches to pose and trajectory estimation will likely outperform methods like SBA when it comes to outlier and noise rejection.

Finally, tests should be performed comparing the effects of model inaccuracies. Using a motion capture system like a Vicon camera system to obtain high-quality 3D trajectory estimates of a real subject and comparing these 3D estimates to those obtained using model-based, markerless motion capture techniques would be a valuable exercise.

Bibliography

[1] N. Sharp, "Timed running speed of a cheetah (acinonyx jubatus)", *Journal of Zoology*, vol. 241, no. 3, pp. 493–494, 1997.

[2] A. M. Wilson, J. Lowe, K. Roskilly, P. E. Hudson, K. Golabek, and J. McNutt, "Locomotion dynamics of hunting in wild cheetahs", *Nature*, vol. 498, no. 7453, pp. 185–189, 2013.

[3] S. Seok, A. Wang, M. Y. Chuah, D. Otten, J. Lang, and S. Kim, "Design principles for highly efficient quadrupeds and implementation on the MIT cheetah robot", in *2013 IEEE International Conference on Robotics and Automation*, IEEE, 2013, pp. 3307–3312.

[4] G. Bledt, M. J. Powell, B. Katz, J. Di Carlo, P. M. Wensing, and S. Kim, "Mit cheetah 3: Design and control of a robust, dynamic quadruped robot", in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2018, pp. 2245–2252.

[5] A. Patel and M. Braae, "Rapid turning at high-speed: Inspirations from the cheetah's tail", in *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, IEEE, 2013, pp. 5506–5511.

[6] R. Briggs, J. Lee, M. Haberland, and S. Kim, "Tails in biomimetic design: Analysis, simulation, and experiment", in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, IEEE, 2012, pp. 1473–1480.

[7] E. Insafutdinov, L. Pishchulin, B. Andres, M. Andriluka, and B. Schiele, "Deepercut: A deeper, stronger, and faster multi-person pose estimation model", in *European Conference on Computer Vision*, Springer, 2016, pp. 34–50.

[8] T. Nath, A. Mathis, A. C. Chen, A. Patel, M. Bethge, and M. W. Mathis, "Using DeepLabCut for 3d markerless pose estimation across species and behaviors.", *Nature protocols*, vol. 14, no. 7, pp. 2152–2176, 2019.

[9] T. D. Pereira, D. E. Aldarondo, L. Willmore, M. Kislin, S. S.-H. Wang, M. Murthy, and J. W. Shaevitz, "Fast animal pose estimation using deep neural networks", *Nature methods*, vol. 16, no. 1, pp. 117–125, 2019.

[10] J. M. Graving, D. Chae, H. Naik, L. Li, B. Koger, B. R. Costelloe, and I. D. Couzin, "Deepposekit, a software toolkit for fast and robust animal pose estimation using deep learning", *eLife*, vol. 8, e47994, 2019.

[11] A. Patel, B. Stocks, C. Fisher, F. Nicolls, and E. Boje, "Tracking the cheetah tail using animal-borne cameras, gps, and an imu", *IEEE sensors letters*, vol. 1, no. 4, pp. 1–4, 2017.

[12] T. B. Moeslund and E. Granum, "A survey of computer vision-based human motion capture", *Computer vision and image understanding*, vol. 81, no. 3, pp. 231–268, 2001.

[13] M. Zhao, T. Li, M. Abu Alsheikh, Y. Tian, H. Zhao, A. Torralba, and D. Katabi, "Through-wall human pose estimation using radio signals", in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 7356–7365.

[14] T. B. Moeslund, A. Hilton, and V. Krüger, "A survey of advances in vision-based human motion capture and analysis", *Computer vision and image understanding*, vol. 104, no. 2-3, pp. 90–126, 2006.

[15] M. Windolf, N. Götzen, and M. Morlock, "Systematic accuracy and precision analysis of video motion capturing systems—exemplified on the vicon-460 system", *Journal of biomechanics*, vol. 41, no. 12, pp. 2776–2780, 2008.

[16] Vicon, *What is motion capture?*, `https://www.vicon.com/about-us/what-is-motion-capture`, Accessed: 2020-05-29.

[17] J. Han, L. Shao, D. Xu, and J. Shotton, "Enhanced computer vision with Microsoft Kinect sensor: A review", *IEEE transactions on cybernetics*, vol. 43, no. 5, pp. 1318–1334, 2013.

[18] T. Dutta, "Evaluation of the Kinect sensor for 3-d kinematic measurement in the workplace", *Applied ergonomics*, vol. 43, no. 4, pp. 645–649, 2012.

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

[20] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition", *arXiv preprint arXiv:1409.1556*, 2014.

[21] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions", in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 1–9.

[22] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition", in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[23] L. Deng, "The mnist database of handwritten digit images for machine learning research [best of the web]", *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 141–142, 2012.

[24] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database", in *2009 IEEE conference on computer vision and pattern recognition*, IEEE, 2009, pp. 248–255.

[25] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft COCO: Common objects in context", in *European conference on computer vision*, Springer, 2014, pp. 740–755.

[26] Z. Cao, G. Hidalgo, T. Simon, S.-E. Wei, and Y. Sheikh, "Openpose: Realtime multi-person 2d pose estimation using part affinity fields", *arXiv preprint arXiv:1812.08008*, 2018.

[27] R. Alp Güler, N. Neverova, and I. Kokkinos, "Densepose: Dense human pose estimation in the wild", in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 7297–7306.

[28] M. Andriluka, L. Pishchulin, P. Gehler, and B. Schiele, "2d human pose estimation: New benchmark and state of the art analysis", in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2014.

[29] C. Ionescu, D. Papava, V. Olaru, and C. Sminchisescu, "Human3.6m: Large scale datasets and predictive methods for 3d human sensing in natural environments", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 36, no. 7, pp. 1325–1339, Jul. 2014.

[30] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks", in *European conference on computer vision*, Springer, 2014, pp. 818–833.

[31] M. Huh, P. Agrawal, and A. A. Efros, "What makes imagenet good for transfer learning?", *arXiv preprint arXiv:1608.08614*, 2016.

[32] C. Shorten and T. M. Khoshgoftaar, "A survey on image data augmentation for deep learning", *Journal of Big Data*, vol. 6, no. 1, p. 60, 2019.

[33] P. Corke, *Robotics, Vision and Control: Fundamental Algorithms in MATLAB*, 1st. Springer Publishing Company, Incorporated, 2013, ISBN: 3642201431.

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

[35] C. G. Harris, M. Stephens, *et al.*, "A combined corner and edge detector.", in *Alvey vision conference*, Citeseer, vol. 15, 1988, pp. 10–5244.

[36] R. Hartley and A. Zisserman, *Multiple view geometry in computer vision*. Cambridge university press, 2003.

[37] R. Szeliski, *Computer vision: algorithms and applications*. Springer Science & Business Media, 2010.

[38] Z. Zhang, "A flexible new technique for camera calibration", *IEEE Transactions on pattern analysis and machine intelligence*, vol. 22, no. 11, pp. 1330–1334, 2000.

[39] *MATLAB version 9.8.0.1323502 (R2020a)*, The Mathworks, Inc., Natick, Massachusetts, 2020.

[40] H. C. Longuet-Higgins, "A computer algorithm for reconstructing a scene from two projections", *Nature*, vol. 293, no. 5828, pp. 133–135, 1981.

[41] D. Nistér, "An efficient solution to the five-point relative pose problem", *IEEE transactions on pattern analysis and machine intelligence*, vol. 26, no. 6, pp. 756–770, 2004.

[42] H. Li and R. Hartley, "Five-point motion estimation made easy", in *18th International Conference on Pattern Recognition (ICPR'06)*, IEEE, vol. 1, 2006, pp. 630–633.

[43] R. I. Hartley and P. Sturm, "Triangulation", *Computer vision and image understanding*, vol. 68, no. 2, pp. 146–157, 1997.

[44] B. Triggs, P. F. McLauchlan, R. I. Hartley, and A. W. Fitzgibbon, "Bundle adjustment—a modern synthesis", in *International workshop on vision algorithms*, Springer, 1999, pp. 298–372.

[45] M. I. Lourakis and A. A. Argyros, "SBA: A software package for generic sparse bundle adjustment", *ACM Transactions on Mathematical Software (TOMS)*, vol. 36, no. 1, pp. 1–30, 2009.

[46] R. E. Kalman, "A new approach to linear filtering and prediction problems", 1960.

[47] N. Thacker and A. Lacey, "Tutorial: The Kalman filter", *Imaging Science and Biomedical Engineering Division, Medical School, University of Manchester*, p. 61, 1998.

[48] R. Labbe, *Kalman and Bayesian filters in Python*, `https://github.com/rlabbe/Kalman-and-Bayesian-Filters-in-Python`, 2014.

[49] E. A. Wan and R. Van Der Merwe, "The unscented Kalman filter for nonlinear estimation", in *Proceedings of the IEEE 2000 Adaptive Systems for Signal Processing, Communications, and Control Symposium (Cat. No. 00EX373)*, IEEE, 2000, pp. 153–158.

[50] F. Gustafsson, F. Gunnarsson, N. Bergman, U. Forssell, J. Jansson, R. Karlsson, and P.-J. Nordlund, "Particle filters for positioning, navigation, and tracking", *IEEE Transactions on signal processing*, vol. 50, no. 2, pp. 425–437, 2002.

[51] H. E. Rauch, F. Tung, and C. T. Striebel, "Maximum likelihood estimates of linear dynamic systems", *AIAA journal*, vol. 3, no. 8, pp. 1445–1450, 1965.

[52] G. J. Bierman, *Factorization methods for discrete sequential estimation*. Courier Corporation, 2006.

[53] G. A. Einicke, "Optimal and robust noncausal filter formulations", *IEEE Transactions on Signal Processing*, vol. 54, no. 3, pp. 1069–1077, 2006.

[54] C. V. Rao, J. B. Rawlings, and D. Q. Mayne, "Constrained state estimation for nonlinear discrete-time systems: Stability and moving horizon approximations", *IEEE transactions on automatic control*, vol. 48, no. 2, pp. 246–258, 2003.

[55] N. Hashemian and A. Armaou, "Fast moving horizon estimation of nonlinear processes via carleman linearization", in *2015 American Control Conference (ACC)*, IEEE, 2015, pp. 3379–3385.

[56] M. Kok, J. Hol, and T. Schön, "An optimization-based approach to human body motion capture using inertial sensors", in *19th World Congress of the International Federation of Automatic Control (IFAC), Cape Town, South Africa, August 24-29, 2014*, International Federation of Automatic Control, 2014, pp. 79–85.

[57] X. Xinjilefu, S. Feng, and C. G. Atkeson, "Dynamic state estimation using quadratic programming", in *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, IEEE, 2014, pp. 989–994.

[58] F. Mueller, F. Bernard, O. Sotnychenko, D. Mehta, S. Sridhar, D. Casas, and C. Theobalt, "Ganerated hands for real-time 3d hand tracking from monocular RGB", in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 49–59.

[59] J. Diebel, "Representing attitude: Euler angles, unit quaternions, and rotation vectors", *Matrix*, vol. 58, no. 15-16, pp. 1–35, 2006.

[60] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, *et al.*, "Scipy 1.0: Fundamental algorithms for scientific computing in python", *Nature methods*, vol. 17, no. 3, pp. 261–272, 2020.

[61] H. Fang, M. A. Haile, and Y. Wang, "Robustifying the Kalman filter against measurement outliers: An innovation saturation mechanism", in *2018 IEEE Conference on Decision and Control (CDC)*, IEEE, 2018, pp. 6390–6395.

[62] B. Nicholson, R. López-Negrete, and L. T. Biegler, "On-line state estimation of nonlinear dynamic systems with gross errors", *Computers & chemical engineering*, vol. 70, pp. 149–159, 2014.

[63] W. E. Hart, J.-P. Watson, and D. L. Woodruff, "Pyomo: Modeling and solving mathematical programs in python", *Mathematical Programming Computation*, vol. 3, no. 3, pp. 219–260, 2011.

[64] W. E. Hart, C. D. Laird, J.-P. Watson, D. L. Woodruff, G. A. Hackebeil, B. L. Nicholson, and J. D. Siirola, *Pyomo–optimization modeling in python*, Second. Springer Science & Business Media, 2017, vol. 67.

[65] A. Wächter and L. T. Biegler, "On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming", *Mathematical programming*, vol. 106, no. 1, pp. 25–57, 2006.

[66] S. Barratt and S. Boyd, "Fitting a Kalman smoother to data", *arXiv preprint arXiv:1910.08615*, 2019.

[67] D. Simon and T. L. Chia, "Kalman filtering with state equality constraints", *IEEE transactions on Aerospace and Electronic Systems*, vol. 38, no. 1, pp. 128–136, 2002.