DESIGN OF MANY-CAMERA TRACKING SYSTEMS FOR SCALABILITY AND EFFICIENT RESOURCE ALLOCATION

A DISSERTATION
SUBMITTED TO THE DEPARTMENT OF ELECTRICAL ENGINEERING
AND THE COMMITTEE ON GRADUATE STUDIES
OF STANFORD UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

Xing Chen
July 2002

I certify that I have read this dissertation and that in my opinion it is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

Patrick M. Hanrahan
(Principal Adviser)

I certify that I have read this dissertation and that in my opinion it is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

Christoph Bregler

I certify that I have read this dissertation and that in my opinion it is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

Leonidas J. Guibas

Approved for the University Committee on Graduate Studies:

To my husband Jason Fan,

and to my parents

Huaichen Chen and Jiemin Gong

Abstract

There are many applications where it is useful to know the location and/or orientation of people or objects as they move through space. For example, head and hand position can be used as input to drive a virtual reality application, or the joint motion of a dancer can be captured to drive the movement of an animated figure. Among various motion tracking technologies, systems using cameras as optical sensors have been the subject of intensive research for years due to their non-intrusive nature and their immunity to ferromagnetic distortion. A network of cameras allows for a larger working volume, higher tracking accuracy, more robustness and greater flexibility, but also introduces a variety of interesting problems. The architecture of such systems needs to be designed to be scalable to a large number of cameras. The added complexity of such systems makes it difficult to calibrate all of the cameras into a single global coordinate frame in a scalable fashion, and also introduces the problem of how to optimally place cameras to maximize the performance of the tracking system.

In this dissertation, we have designed and implemented M-Track, a scalable tracking architecture for real-time motion tracking with tens of cameras. Its one-processor-per-camera design enables the parallel processing of high-bandwidth image data. The employment of a central estimator based on an extended Kalman filter allows the smooth and asynchronous integration of information from camera-processor pairs. Since camera synchronization is not required, this architecture results in easier deployment, and enables the employment of heterogeneous sensors including cameras with different resolutions and frame rates. The architecture also supports tracking of multiple features and automatic labeling of these features, even when some feature points are temporarily occluded. Three end-to-end applications are built upon this architecture to demonstrate the usefulness of the system.

Next, we present a scalable wide-area multi-camera calibration scheme. A large number of asynchronous cameras can be calibrated into a single consistent coordinate frame by simply waving a bright light in front of them. This can be achieved even when cameras are arranged with non-overlapping working volumes and when no initial estimates of camera poses are available. There is no need for the construction of a universally visible physical calibration object, and the method is easily adaptable to working volumes of variable size and shape.

We then propose a quantitative metric for evaluating the tracking quality given a multi-camera placement configuration. Even though occlusion of tracked objects has a huge impact on motion tracking, previous work only uses the 3D uncertainty caused by limited camera resolution to evaluate quality. Our metric considers both camera resolution and the likelihood of target occlusion. In the formulation of the metric, we propose a novel probabilistic model that estimates the dynamic self-occlusion of targets and verify its validity through experimental data. Finally, we analyze various camera placement configurations using our proposed metric and show the impact on camera placement requirements when considering either only resolution or both resolution and occlusion.

Acknowledgements

This work would not have been possible without the help and encouragement of many people.

First of all, I feel extremely privileged and grateful to have had Dr. Pat Hanrahan as my research advisor for the past five years. He has consistently encouraged me and given me the freedom to explore a variety of research topics. It has been a rewarding learning experience for me to work with him, and his vision, advice, high standards and especially his emphasis on first principles in research have over the years facilitated my growth as both a researcher and a professional.

My most sincere thanks go to Stanford professors who have graciously expended their time and effort for my work: Drs. Leo Guibas and Chris Bregler. I thank them for their technical insights and advice, as well as their friendship and moral support.

I extend my thanks to Dr. Anoop Gupta of Microsoft Research, for initially getting me started on a video streaming project while he was a professor at Stanford. The experience from that project has motivated me in my exploration of the continually converging fields of video, graphics and vision, and their applications in networked environments.

I would like to gratefully acknowledge my good friend and close collaborator in much of the work in this dissertation, James Davis. His work is an excellent and useful complement to mine, and this research would not have progressed this far without his knowledge, insights and hard work.

I would also like to acknowledge my friends and colleagues in the Stanford Graphics Lab. It has been a lot of fun to be around so many intelligent, motivated people whose knowledge and creativity I've often drawn on. In particular, I thank François Guimbretière,

Brad Johanson, Tamara Munzner, and Szymon Rusinkiewicz for not only their professional insights and suggestions, but also for their personal support and friendship. Special thanks go to staff members John Gerth, Ada Glucksman and Heather Gentner for making my working environment pleasant and hassle-free with their technical and administrative support.

My grateful thanks extend to the various organizations that sponsored my work: Intel Corporation, Interval Research, and Sony Corporation.

Finally, my heartfelt thanks go to my beloved better-half, Jason Fan, who has been so patient and supportive over the past few years with my flexible, fun-seeking, graduate school life style while he himself has endured a more regimented one in industry; and I thank my parents, Huaichen Chen and Jiemin Gong, who are engineering professors themselves and have raised me to value logic, curiosity, creativity, and scientific exploration, and who have trusted me and served as an emotional anchor throughout my life. Their love has been a constant source of strength for me and I dedicate this dissertation to them.

Contents

List of Figures

Chapter 1

Introduction

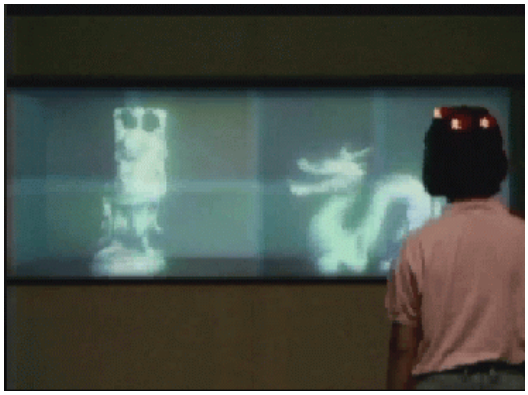
1.1 Applications of motion tracking

There are many applications where it is useful to know the location and/or the orientation of people or objects as they move through space. This type of motion tracking, sometimes called 6 degree-of-freedom (DOF) position tracking, has been used in military applications for years[1]. Recently, motion tracking has been found very useful for virtual reality (VR) and entertainment industry applications. The main distinction between motion tracking in VR and entertainment applications and in military applications is that the subjects being tracked are usually human or human held objects. For example, in a VR application, the head and hand positions of a user can be used to control computer-generated images so that they are always rendered from the point of view of the user and in response to the user's hand gestures. In entertainment, since it is labor intensive to manually animate figures whose intended motion is complex, the motions of an actor can be applied to an animated figure via tracking the motions of all relevant joints and body parts. This tracking process, called *motion capture* in the entertainment industry, can also be applied to other domains. For example, the captured motion of a top athlete can be used in training or sports medicine. Figure 1.1 shows a few examples of such applications.

The requirements on the performance of real-time motion tracking systems based on

[1] In this thesis, the term *motion tracking* refers to the tracking of target position, orientation, and their derivatives including velocity, angular velocity, etc.

(a)



(b)

Figure 1.1: Example applications of position trackers. (a) Motion capture of a dancer's movement; (b) Tracking of user head position as input to drive the display of a virtual reality system.

accuracy, resolution, responsiveness, range, and robustness are important for most applications. VR and entertainment applications have additional requirements due to the variety of environments in which the systems need to be deployed, including many types of human-inhabited environments. These requirements are based on non-intrusiveness to the subjects being tracked, support for multiple targets, and flexibility of deployment and configuration. A more detailed description of these requirements is provided below:

Accuracy and resolution These are indicators of the exactness of the reported target positions by a system, where a target can be considered as a distinct object or as a portion of an object that can have a distinct position or orientation relative to the rest of the object. *Accuracy* is the maximum error of a reported position from its real position and it measures how "correct" it is. For example, a system with an accuracy of 5 millimeters indicates that a reported position is within ± 5 millimeters of the actual position. This is one of the key measures of performance for a motion tracker, since poor position accuracy (and thus poor accuracy of estimation of other motion parameters) can cause erroneous responses from a VR system or generation of distorted motion for animated figures. A related measure is *resolution*, which is the minimum target position change that the system can detect. While good accuracy depends on

good resolution, good resolution does not guarantee good accuracy.

Reponsiveness This is usually measured by the system *latency*, the time lag between the movement of the target and the report of the new position, and the *data rate*, the number of computed positions (and other motion parameters) per second. VR applications are interactive and require low latency, while motion capture applications must capture details of high speed movements and require high data rates.

Non-intrusiveness A tracking system inevitably needs to deploy sensors and other devices in the environment and/or on the target. However, when tracking for VR applications or motion capture, the target being tracked is usually human, or objects held and moved by humans (these objects are called "props"). A system with a lot of cumbersome devices and wires attached on or around the person can be very distracting and obtrusive to his intended movements. In order for the system to be frequently usable, it is important for the human to be able to focus on his task and move freely. This means that the tracking system should be instrumented and architected so that it is as non-intrusive as possible.

Range This is the size of the *working volume* in which the motion needs to be accurately reported. The exact requirements on the range are application dependent. Since the target motion in VR and entertainment applications is usually associated with human movements, the range should be at least room size. There are many other future applications that require a much larger range, such as tracking players on a football field. The tracking system should be architected to be *scalable* to any large working volume.

Support for multiple targets The ability to monitor the movement of multiple targets is important for many applications. For example, an "intelligent" meeting room might need to observe movements of multiple people in the room and respond; or in motion capture, usually the motions of multiple body parts need to be recorded. Systems that permit the independent determination of the positions of multiple objects are desirable. Different applications may also require different levels of multi-target support. This support can vary from simply the output of multiple positions for every

frame to the distinguishing (or "labeling") of each target and the determination of time-coherent *traces* for each labeled target.

Robustness A practical system must be able to withstand the noise, interference and uncertainty of the real world. These factors could cause erroneous position reports or even throw off the system totally by causing it to lose or confuse targets. One particularly serious type of interference is when a sensor is blocked or *occluded* from receiving the signal from the emitter. Noise and interference may not only come from objects in the environment, but can also come from other targets. It is important to design a system to be resistant to these error sources.

Flexibility This refers to all kinds of factors that affect a system's ability to be easily deployed and configured for various applications and its ability to be reconfigured or upgraded over the course of its use. For example, applications may have variable tracking resolution requirements for different subspaces of the working volume; a system employing multiple sensors may desire to use sensors of different resolution, speed and even category. These sensors also may not be synchonized. Moreover, before tracking can be performed, a system needs to be configured with priors such as sensor locations and other parameters, a process that is usually time-consuming itself. Therefore, when a sensor needs to be added, replaced or removed, it would be desirable to be able to *incrementally* reconfigure the system. One important aspect of flexibility is *scalability*, a system's ability to accommodate increase in various system parameters such as the number and type of cameras, number of targets, range, etc.

To summarize, a good motion tracker for VR and entertainment applications should provide high accuracy, good responsiveness, large range, and support for multiple targets while being non-intrusive, robust and flexible. This thesis addresses design of motion tracking systems for these kinds of applications.

1.2 Sensor technologies

Motion trackers use four main categories of motion sensors: optical, magnetic, acoustic and mechanical [52]. Currently the two major sensor types used in VR and entertainment

applications are optical and magnetic.

Magnetic trackers have fixed emitters in the working volume, and have sensors mounted on the target. These trackers estimate the position of the target by measuring the change in the magnetic field as the target (sensor) is moving. Multiple targets are supported by the time-division multiplexing of multiple emitter-sensor pairs. However, the magnetic field is subject to distortions caused by ferrometallic objects and objects that generate electromagnetic fields such as transformers and computer screens. This distortion increases with emitter power.

Optical trackers employ optical sensors to detect visual *features* from images of the target. The sensors vary from photodiodes to ordinary video cameras, and the signals being detected vary from emitted lights controlled by the tracker to readily available ambient lights. Usually *markers* such as small bright light emitters, reflective balls, or paint of a particular color needs to be attached or applied to the targets to make them more distinguishable for the sensor. Optical trackers compute the 3D positions of the targets based on the geometric relationship between the detected feature locations on the image (2D) and the sensor positions (3D) known a priori.

Compared to magnetic sensors, optical sensors are more widely used for VR and entertainment tracking applications for the following reasons. They are (a) inherently more accurate because they don't suffer from ferromagnetic distortion; (b) much less intrusive, because the markers attached to the target are usually much smaller and less cumbersome than those for magnetic trackers; and (c) have bigger range because optical signal can reach farther and emitter power can be increased without the increase of distortion. The main disadvantages of systems based on optical sensors are that they are subject to occlusion and that they have difficulty distinguishing between multiple targets. However, they are the preferred sensor technology because accuracy, non-intrusiveness and range are vital to these applications. Moreover, good system design not only can leverage and extend the advantages of the sensors, but also compensate for their limitations, thus satisfying more advanced application requirements. The main goal of this thesis is to provide insights into multiple facets of system design issues for camera-based trackers.

1.3 Motivation for a network of cameras

1.3.1 Benefits

Though individual optical sensors have perhaps the best sensor accuracy and range properties, advanced applications with increased accuracy, range and robustness requirements necessitate the use of many sensors, or a *network* of sensors (or cameras in the context of this thesis). Figure 1.2 shows a conceptual illustration of a tracking system with many cameras. A network of cameras has the following benefits.

Better coverage and resolution can only be achieved by employing many cameras. For example, if one wants to track the motion of a soccer player in a stadium or to extract the motion of people in a building with a complex floor plan, it is impossible to perform such tasks with only one camera, since a camera has limited range and resolution. Since a camera has a fixed number of pixels, one can either achieve larger coverage but lower resolution by using a wide angle lens, or achieve higher resolution but smaller coverage with a narrow angle lens, but not both. In addition, there is a distance-resolution tradeoff: as the distance from a target to the camera increases, its size on the camera image plane decreases, and as a result tracking resolution and accuracy decreases. Larger coverage for a given resolution can only be achieved by combining the ranges of many cameras placed and/or aimed at different locations. By "many", we mean tens of cameras or even more. Since the term "multi-camera" has been used in the computer vision literature mostly to refer to 2-5 cameras, in this dissertation we use the term *many-camera* to refer to such systems to emphasize the different design space from previously described multi-camera systems.

System robustness can also be improved with the use of many cameras because of the reduction of camera *occlusion*. Occlusion is the phenomenon where a feature is visually blocked from a camera by objects called *occluders*. Occluders can be objects in the scene or the target itself. When occlusion occurs and no camera sees a feature, it is impossible to calculate its 3D postion. It is intuitive that the more cameras that are deployed in the scene, the less occlusion there should be, since features occluded from one camera could be seen by cameras located elsewhere. In addition, in a practical system other factors (such as power failure) could cause a camera to stop functioning. More cameras provide more

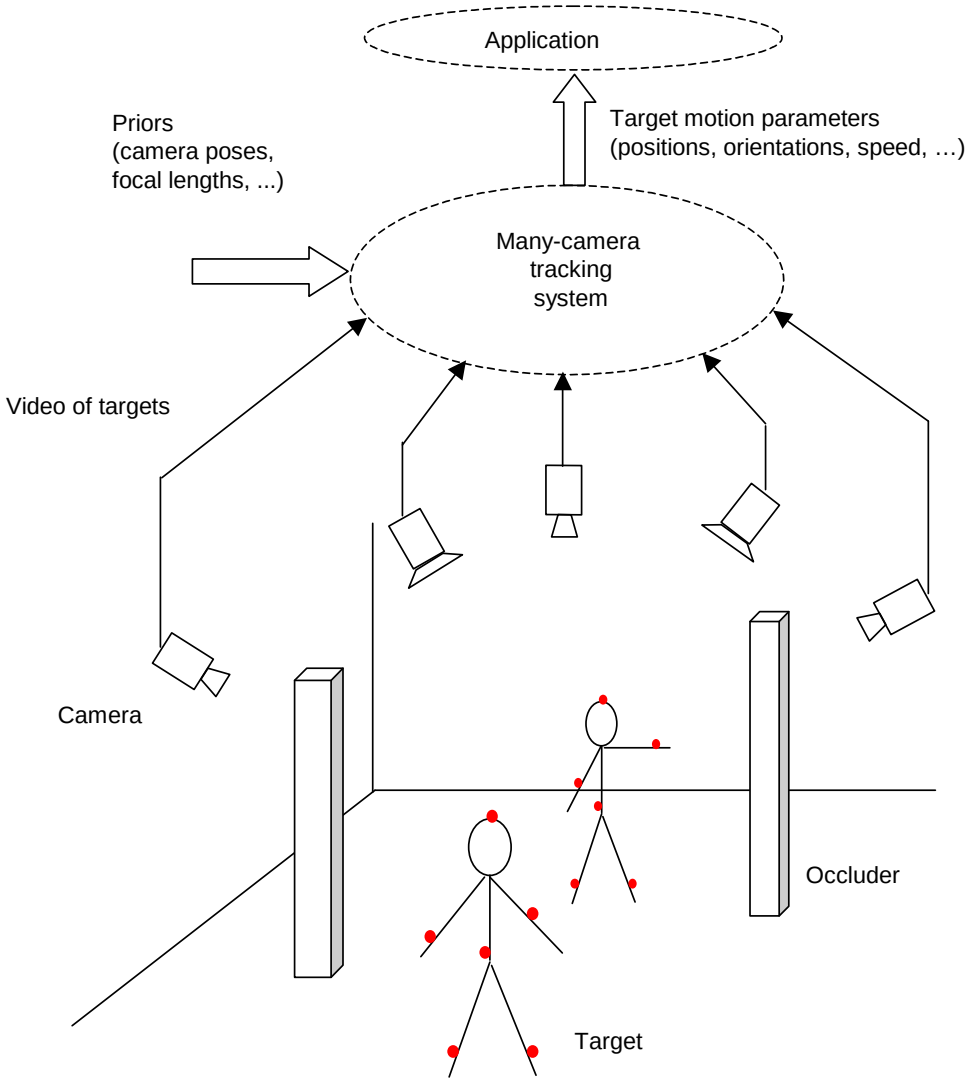


Figure 1.2: Schematic of a many-camera tracking system. Tens of cameras observe moving targets. The tracking system determines the 3D positions of the targets based on detection of their 2D positions in the camera images and on some prior knowledge such as where all the cameras are located. Cameras can be asynchronous and have heterogeneous parameters. They are also subject to occlusion.

redundancy. With a sufficient number of cameras, proper system design can ensure that the failure of any one or more cameras will not result in the failure to track the target anywhere in the working volume.

Another consideration for using a network of cameras is cost. Though one may argue that future technology might enable us to build a "super-sensor" with exceptional range and resolution, a system composed of many off-the-shelf commodity cameras is usually more cost-effective than one with a smaller number of custom, high performance sensors.

1.3.2 Issues to be solved

As described from the previous section, a tracking system can improve some aspects of its performance by employing a network of cameras. However, there are issues remaining to be addressed. Moreover, the use of a network of many cameras also introduces other challenges in system design. These are described below.

Though by employing many cameras one can extend the range and resolution of a system, the tracking accuracy of a system is dependent on many other factors. For one thing, as mentioned and seen in Figure 1.2, in order to compute target position, the system needs to know the pose (i. e. position and orientation) and other optical parameters (such as focal length) of all cameras as priors. The process of measuring and determining these geometric and optical parameters is called *calibration*, which usually has to be done before the actual tracking can be performed. The accuracy of target position directly depends on the accuracy of camera calibration. For advanced applications with large and geometrically complex working volumes, traditional calibration methods may not handle certain complex camera layouts. There needs to be a calibration scheme that can handle a wide and complex coverage area.

Target position accuracy also depends on the accurate measurement of 2D target location on the image plane of a camera. However, cameras inherently have a range-distance-resolution trade-off, and they are subject to occlusion. Poorer resolution means larger error in the measurement of the feature's 2D position on the image, and therefore less accurate 3D position. Occlusion of a feature prevents a camera from getting any information about

this feature at all. Both phenomena may cause more error and uncertainty in the determination of the feature's 3D position. From Figure 1.2 it is not hard to see that the severity of these phenomena are affected by *where cameras are located* relative to the targets and occluders, and by their other settings such as their fields of view. Therefore, in order to achieve accurate and robust tracking, it is important to explore what configurations[2] to set for all of the cameras in order to achieve minimum 3D position error (or best accuracy) for a given motion tracking task.

A third factor that affects the accuracy is how a system handles *asynchronous* measurements. By "asynchronous" we mean that cameras do not all take pictures at the same time. Many published multi-camera tracking algorithms assume that all cameras take their measurements simultaneously, which requires physically wiring and sending a synchronization signal to every camera. This is a non-trivial engineering nuance, especially when the number of cameras is large [80]. Asynchronous data from multiple cameras causes algorithms that assume synchonized cameras to report erroneous motion information. Therefore, it is desirable to have a tracking system architecture and associated algorithms that are able to support a large number of asynchronous cameras.

Support for multiple targets remains an issue for camera-based tracking systems. As mentioned in Section 1.1, this is needed by many motion tracking applications. However there is an inherent trade-off between *detecting* a feature in a scene and *identifying* it. For example, a simple feature such as a point is easier and faster to detect from an image than other more complex types of features, but is harder to distinguish from other feature points. Most current motion capture systems have a low level of support for multiple targets: the systems just output a set of positions per frame, and leave the labeling to manual labor later– a very tedious, time-consuming process. A system architecture that has a higher level of support for multiple targets leads to a more efficient end-to-end motion recovery pipeline.

Flexibility, and especially scalability, is an issue that arises and becomes increasingly important when the number of cameras in a system becomes large. First, the calibration

[2]The configuration of a camera refers both the location and orientation at which the camera is placed as well as its optical parameter settings. For the sake of simplicity, we will use the imprecise term camera "placement" to refer to the setting of both the geometric and optical parameters of a camera.

of many cameras using traditional methods quickly becomes labor intensive and time-consuming, which makes the deployment and configuration of a system difficult. In order to deploy a many-camera system efficiently, there needs to be a camera calibration scheme that can scale to support of a large number of cameras with minimum labor. Second, with many cameras it is possible to allow variable placement density of cameras so that spatially varying tracking resolution can be achieved, which constitutes one aspect of system flexibility. However, there needs to be a systematic and quatitative way to compute where cameras should be laid out to achieve such variable coverage resolution. Third, in terms of system architecture, it is important not only to allow asynchronous cameras, but also to allow *heterogeneous* cameras. "Heterogeneous" means that the cameras may have different ranges, resolutions, and frame rates. As the number of cameras becomes large, this is not an unreasonable assumption. The use of heterogeneous cameras not only makes the the system more flexible, but also could effectively leverage the advantages of each kind. Another aspect of system flexibility is its upgradability. With a large number of cameras, it is extremely important to have a system architecture that supports the incremental addition, deletion, and upgrade of a subset of the cameras.

Responsiveness is another performance measure that needs to be considered with the increase of the number of cameras. More cameras means more raw data to process, and more overhead for communication, thus potentially leading to either longer latency or an increase in unprocessed (wasted) data. It is desirable that the a system is scalable in terms of its processing power and maintains its responsiveness as the number of camera increases.

In this dissertation a scalable many-camera tracking architecture is proposed, together with a novel camera calibration scheme and a formulation and study of the selection of optimal camera configurations. This work addresses how to improve the level of accuracy, flexibility and multi-target support for a many-camera tracking system through good system design, specifically through improved camera calibration, camera placement, and system architecture. Several applications are implemented to demonstrate the system's overall capability, and the various performance aspects described earlier. Though some care is taken to assure that the system is responsive enough for the demonstrated applications, the general issue of system responsiveness is not the focus of this thesis.

1.4 Related work

In this section we review some previous work in multi-camera tracking, especially from the perspective of the amount of support provided for a large number of asynchronous cameras and for labeling of multiple features or targets. Previous work directly related to calibration of a network of cameras or camera placement is discussed separately in Chapters 3 and 4, respectively.

Computer vision-based human motion tracking systems can be divided into two general types: those that are developed by research institutions and those that are developed and commercialized by industry. Systems developed by the two communities have very different focuses.

The motion capture systems that are developed by computer vision researchers are usually focused on addressing issues relating to the modeling aspect of the problem. (A comprehensive survey from this perspective can be found in [53].) A usual assumption is that no marker is attached on the person to be tracked; therefore, a lot of work is devoted to how to segment useful features of the target from the background based on solely the natural appearance of the target and how to represent the segmented entities. For example, the extracted features can range from edges [39] and silhouettes [7, 71, 34], to "blobs" that are regions of coherent color [19, 43, 38] or inter-frame flow[44]. One state of the art system is the Pfinder system and its extension developed by Wren et al. [89, 90], in which the human body is tracked using statistical models to segment the image into blobs. Using these natural and complex features not only reduces the intrusiveness of the system but also may make it easier to distinguish among multiple features because they are described by more parameters. However, since these features are more complex than points, the segmentation and extraction steps take significant processing and usually do not provide the frame rate that VR and entertainment applications require. Because these features are usually coarse and imprecise, they are less suitable for applications that have a high accuracy requirement on the position data, such as the tracking of a user's eye position to render a image from that point of view. Most of these systems consist of only 2-5 synchronized cameras with limited range and do not address the camera synchronization issue. Neither do they address the use of heterogenous cameras. In addition, since most of these systems have very few cameras,

target occlusion is very likely to occur, but most of the work reports results for cases where all of the feature points are seen. It is not reported how often target occlusion occurs, and how the system would perform in this case of temporary lack of full information about the target. We consider occlusion an important and unavoidable issue in camera-based tracking and address it explicitly. We employ a large number of cameras and integrate their information separately (see Section 2.2.3.1), provide a way of predicting target motion in the case of temporary feature point occlusion though the use a target dynamic model (see Section 2.2.3.2) and explicitly design the camera layout to reduce the likelihood of occlusion (see Chapter 4).

One research system that employs many cameras is the Virtualized Reality system developed at Carnegie Mellon University [56, 48, 68, 80, 74]. In that system more than 30 cameras are mounted on a dome-like structure (and later, in a room), and are synchronized to record what is happening in the work volume. Target information such as shape, motion, etc. are then estimated through post-analysis of the recorded data from the cameras. Though this system is not intended for our desired real-time motion tracking, the experience that the researchers obtained and reported nonetheless provides useful insight into the issues involved in building a system involving many cameras. One issue that they specifically reported is how much effort was required to synchronize a large number of cameras [49]. This is the basis for our belief that support of asynchronous cameras is one important criterion for a scalable tracking architecture.

A tracking system that does address asynchronous input is the head tracker developed at the University of North Carolina (UNC)[87, 86, 88, 4, 3, 31]. In this system the sensors (photodiodes) are mounted on the person and observe *outward* infrared light-emitting diodes (or *beacons*) mounted on the ceiling. At any time only one beacon is seen, and therefore only partial information about the target position is received. Welch [88] proposed a specific way of using the Kalman filter called *single-constraint-at-a-time* (SCAAT) to combine these partial observations that are obtained sequentially and asynchronously. The researchers also mentioned that this formulation would also allow the use of heterogenous sensors. In our system, we use many cameras that look *inward* at the target(s), so the sensor types, properties and models are quite different from theirs. Similar to SCAAT, we also employ a Kalman filter in a way that enables partial measurements to be integrated at a

given time to deduce the global state (motion) information. This enables proper integration of measurements from asynchronous and heterogenous cameras. However, unlike SCAAT, which purposely takes a *single* measurement at time to generate a state estimate even if the measurements were taken synchronously, M-Track groups multiple measurements that are indeed taken simultaneously together (but are still subsets of all possible measurements) and estimate global motion from these multiple measurements. In addition, the tracking system reported in [88] has time-division-multiplexed usage of beacons, e. g. at any one time there is only one beacon on, so it is not clear if and how multiple target points can be supported.

Commercial motion capture systems [17, 77] have a different focus from research systems. Because they need to be deployed and used in a variety of environments, these systems are focused on being real-time, robust, and accurate. They require the attachment of active or passive markers on the target so that they can segment these point features easily and quickly, with a small increase in the system's intrusiveness. They usually employ many (10s or 20s of) cameras to obtain a large working volume and to reduce occlusion. However, cameras are still required to be synchronized, and not much effort has been reported in addressing asynchronous and heterogenous cameras. Moreover, these systems provide very limited support of tracking of multiple points. Point features are simple and it is difficult to distinguish one from another. Most systems usually just output a set of 3D positions per frame and leave the correlation and labeling of these points to post (manual) processing. Many man-hours are needed to manually correlate the 3D positions of one frame to those in a subsequent frame. This becomes even more challenging and time-consuming when some points disappear and reappear from one frame to the next due to occlusion.

Due to the large number of cameras that our system needs to support and the potentially large amount of data to be processed in real-time, we have decided to adopt the commercial systems' basic approach to use simple point features in our system, the M-Track. This allows fast and robust 2D feature extraction. The M-Track has also been architected so that it can accommodate a large number of asynchronous cameras, and provide support of the automatic tracking and labeling of multiple points, resulting in greater flexibility and scalability than current state-of-the-art commercial motion capture systems. We have also developed a scalable algorithm to efficiently calibrate such systems, and a quantitative

model to help optimize multi-camera placement configurations. The M-Track architecture is described in Chapter 2, and the calibration and placement of many cameras are described in Chapters 3 and 4, respectively.

1.5 Contributions of this dissertation

The contributions of this dissertation include the following:

- the formulation and experimental verification of a many-camera calibration scheme that enables the calibration of a wide area system of asynchronous cameras with respect to a single global coordinate system. It is simple and does not require the physical construction of a large calibration object. This method is especially useful for the calibration of a many-camera system in which not all cameras see a common area, a configuration where traditional methods cannot be applied directly. Moreover, the method also allows incremental calibration as cameras are being added to the system.

- the formulation and experimental verification of a quality metric for comparing various camera configurations, as a first and necessary step toward optimal camera layout. The metric takes into account the 3D position error caused by both occlusion and limited image resolution. As part of the formulation of the metric, a novel probabilistic model is proposed that estimates the dynamic self-occlusion of targets and its validity is verified through experimental data.

- the determination of design trade-offs between resolution and occlusion based on simulation of various many-camera configurations using the above quality metric.

- the design of a system architecture for real-time motion tracking that is scalable to the number of cameras. Its one-processor-per-camera structure allows parallel processing of high-bandwidth input image data. The use of an central estimator based on Extended Kalman Filtering (EKF) not only allows asynchronous input from multiple cameras, but also enables smooth, incremental integration of information from local camera-processor pairs. This results in easier deployment than for systems requiring

synchronization among all cameras, and enables the employment of heterogeneous sensors, including cameras with different resolutions and frame rates. In addition, the temporal correlation that EKF provides allows the continuous automatic labeling of multiple features in subsequent frames once they are labeled in the first frame.

- the development of several motion tracking applications utilizing a 2D tracking system or a 3D wide-area tracking system based on the above architecture.

1.6 Organization of this dissertation

The remainder of this dissertation is organized as follows: Chapter 2 describes the hardware and software architecture of the many-camera tracking system and the applications built based upon this architecture. Chapter 3 presents the scalable camera calibration scheme that works well with wide-area many-camera systems. Chapter 4 describes the quality metric for camera placement and shows simulation results that illustrate the impact of occlusion and limited resolution on optimal camera placement. Chapter 5 discusses some interesting future research directions in many-camera tracking.

Chapter 2

M-Track: A Scalable, Asynchronous Architecture

2.1 Introduction

As described in Chapter 1, a goal of this dissertation is the design of a tracking system architecture that meets the various requirements posed by VR and entertainment applications. In this chapter we describe M-Track, a scalable, asynchronous architecture that we have developed to meet these requirements. Compared to previous work, it can track with a much larger number of cameras, and the cameras need not be synchronized. It can also track multiple feature positions in real-time and allows the continuous automatic labeling of these features in subsequent frames once they are labeled in the first frame, which many current motion capture systems do not support. We have implemented three end-to-end VR and motion capture applications where the tracker is integrated with other modules that utilize its output motion data. We demonstrate the validity and usefulness of this architecture by showing that the performance of the enabled applications meets application requirements.

The rest of the chapter is organized as follows. Section 2.2 describes in detail the M-Track architecture, and how it supports asynchronous and heterogenous cameras as well as multiple targets. Section 2.3 describes three applications that use M-Track. Section 2.4 provides a summary of this chapter.

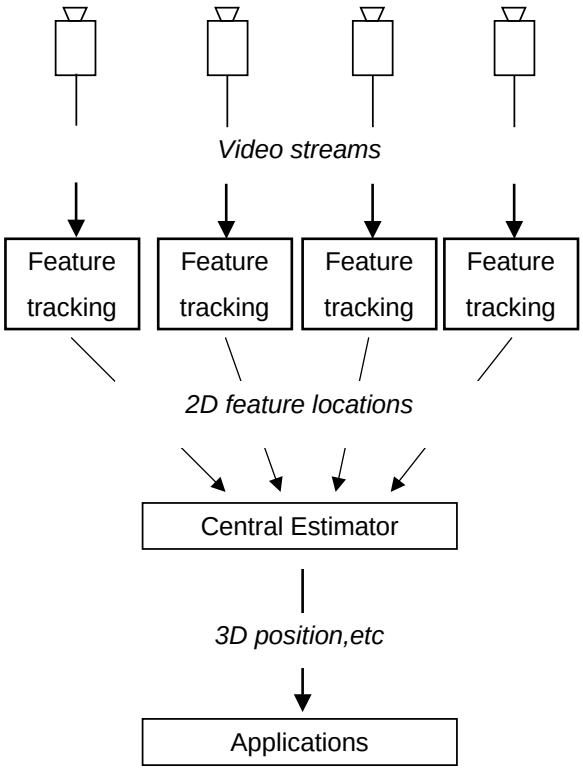


Figure 2.1: Overview diagram of the M-TRACK architecture.

2.2 M-Track architecture

2.2.1 General overview

A high level overview diagram of the M-Track architecture is shown in Figure 2.1.

A large number of cameras are distributed across a wide working volume to observe a target with point features. The video stream from each camera is digitized and processed by a dedicated CPU. Each processor has a computer vision based “feature tracker” running on it that is in charge of detecting and locating the target feature points in the image. This process, the *client*, is 2D and happens in the image-space for each camera. The client needs to be able to do the following:

1. It should be able to distinguish target features from the background in a robust and flexible manner. In our implementation, bright LEDs are attached to the targets

and thresholding is used to distinguish them from the background. Because lighting changes in the operating environment could change the average brightness of the background and other bright objects may exist in the scene, a combined absolute/relative thresholding scheme is used to improve the robustness of the feature detection process.

2. It should be able to perform real-time tracking. To enable this, subsampling is used to reduce the latency of the 2D processing in the case of a slow processor.

The results of the 2D image analysis from various clients are sent asynchronously to a central processor called the *server*. The central server runs an estimator that integrates the 2D information from various different cameras and estimates the 3D *state* of the target, such as its position and orientation. This target state information is then sent to some end-user application, such as a visualization program, that will update its displayed contents according to the target's position or pose. Besides the basic 2D to 3D inference, this central state-space estimator also needs to be able to do the following:

1. It should have some filtering capability to smooth the input data, which inevitably contains thermal noise from the camera and quantization noise from the 2D processing stage.

2. It should be able to handle asynchronous input. Since the 2D information from various cameras arrives at different times, each representing a possibly partial observation of the target state, it should be able to update state information in a *sequential* fashion.

3. It should be able to support input from heterogenous clients.

4. It should be able to track multiple independent targets.

5. It should introduce as little latency as possible because our goal is to do real-time tracking.

Based on the above criteria, we have chosen to use an Extended Kalman Filter (EKF), a well-known tool for parameter estimation from noisy data, as our estimator. This filter

operates in a prediction-update cycle. It can take 2D measurements from one camera at a time, thereby enabling asynchronous and heterogeneous camera support, and incrementally estimate the target position and orientation. It can smooth noisy data, and the incremental update process is time-efficient. Most important, the temporal correlation between feature points of different frames enables the support of asynchronous cameras and the tracking and labeling of multiple independent points.

The network interconnecting the clients and the server must take the following issues into account as well:

1. Since the multiple independently running clients and the server need to communicate with each other via a regular packet-switched network, a common communication protocol needs to be designed to enable exchange of data and control information. The protocol also needs to support convenient manual configuration of the system.

2. Though Kalman filtering does not require that all clients perform their measurements synchronously, it does require that all clients have a universal notion of time. The technique used for providing this universal notion of time must be sufficiently robust to prevent time drift between any client and the server.

3. The variable delay through the network also causes out-of-order packets arriving at the server. An approach must be used to prevent use of out-of-order packets, keeping in mind that the real-time tracking requirement on the server places a premium on low latency.

In the following sections we describe in detail the various modules of the M-Track: the 2D feature tracking in image-space at distributed clients, the state-space estimator, and the networking module.

2.2.2 Distributed image-space processing

The input end of M-Track consists of a number of clients, each being a camera-processor pair. Each camera observes a fraction of the working space, but all cameras combined observe the whole space. Video streams from each camera are sent to the processor and a computer vision algorithm is run to locate the interesting point features. The locations

Figure 2.2: A video frame from a camera with the detected LEDs marked with yellow crosses. The background appears to be very dark compared to the LEDs because we have put a red filter in front of each camera lens so that much of the light outside the passband of the LED is attenuated. The use of only an absolute threshold is not flexible, and the use of only a relative threshold can result in a very low threshold setting that is not robust to noise.

of the point features in the image, as well as other information about the point feature that could be useful for later stages, such as size, color, etc. are then sent to the server.

In our implementation, a large number of NTSC cameras are mounted to cover the working area. Video streams from the cameras are digitized up to 60Hz either by an SGI Indy or a Matrox Meteor II digitizing board on a PC. Bright red or white LEDs are attached to various parts of the target as the "markers" to be detected by the client software. Since the LEDs usually appear much brighter than the background objects, the basic algorithm to locate the feature points is very simple. We can just use thresholding and classify any pixel that is brighter than a threshold T as "bright". As images of LEDs typically occupy more than one pixel on the camera CCD, we additionally aggregate regions of neighboring "bright" pixels into a single observed measurement. We then take the mean location of participating pixels as the observed location. If needed, the size of the aggregated area can also be calculated. Figure 2.2 shows the video of one camera with the detected LED locations overlaid.

In practice, two issues need particular attention when the above basic algorithm is put into use: threshold determination and latency reduction.

As previously mentioned, a pixel is classified as a potential marker if its pixel value is

larger than a threshold. But what should the threshold value be? The choice of an absolute value in advance does not work well because the average brightness of the background varies depending on the lighting condition of the tracking environment. For example, a low threshold might not work when one turns on all the lights in the room. Background brightness also varies spatially. For example, a monitor in the environment can appear as bright as the LED in the image. One could also use a relative threshold scheme, where a background image is taken at the beginning of the tracking program when no LED is in the scene. The background image becomes the basis for a per-pixel thresholding scheme. In other words, pixels are classified if they are brighter than their corresponding background image pixels by T_r. But in the case where the background is very dark, the could result in a very small threshold. The danger of having a very low threshold is that this causes spurious marker detection due to slight lighting change or noise in the background. Given all these considerations, we use a combined relative/absolute threshold scheme. More specifically, let the background image be denoted as $B(u, v)$, where u and v are the 2D image coordinates. At tracking time, a pixel $I(u, v)$ is classified as an LED pixel if $I(u, v) > max(B(u, v) + T_r, T_a)$ where we call T_r the relative threshold and T_a the absolute threshold. What this formula says is that an LED pixel must be brighter than the background by T_r, and be at least as bright as T_a. Figure 2.3 shows a scanline of a background image and, given the relative and absolute thresholds, the final per-pixel threshold function.

The use of background-based thresholds works well in most situations, as long as the brightness of the background does not vary greatly with time. However, this scheme doesn't work when there are computer monitors in the background. This is because the content on the monitor is time-varying. Areas of the monitor that are dark at the time when the background pictures are taken can later become very bright and cause erroneous pixel classification. In order to get around this problem, we have an interactive tool to allow the user to specify "blind" regions on the image plane of a client, and any pixel in those regions is considered useless and is simply discarded. Consequently, neither a monitor nor an LED would get detected as a marker if its image falls onto those regions. But our hope is that, since there are multiple cameras distributed through the scene, that LED will fall on non-blind regions of other cameras and get detected.

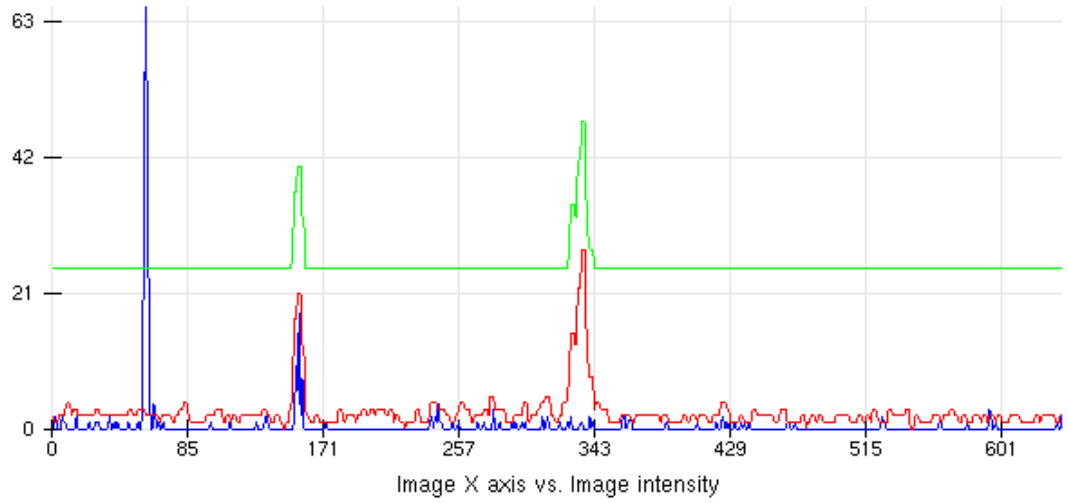


Figure 2.3: Setting the per-pixel threshold. The x-axis is the image x-axis, and the y-axis is the pixel intensity (between 0-255) of the red channel (because we used a red filter to increase the LED/background contrast, as seen in Figure 2.2). The red line is a scanline taken at system start-up time with no LED in the scene,and is therefore assumed to be the "background". The green line shows the per-pixel threshold, given a relative threshold of 20 and an absolute threshold of 25. It is the greater of the background intensity plus the relative threshold, and the absolute threshold. The blue line is a scanline at a given run time with a bright LED in the scene. The threshold should be set so that the it is always lower than the blue peak formed by the LED and higher than the noises in the scene.

Latency is another issue. Because there are 640 by 240 pixels to process in every NTSC video field, and they come to the client CPU at 60Hz, a slower CPU in the Indy workstations available to us cannot keep up. Even though a more state-of-the-art CPU can process such data at least at 30 Hz, the same problem would arise if a higher resolution camera is used. In these cases image subsampling is needed to reduce the total data that must be processed. The subsampling-based mechanism that we use is a multi-resolution approach. We search every N pixels to see if there is any bright pixel. Once we find any, we search the neighboring region of this bright pixel using a finer grid to find any additional bright pixels. The assumption is that the image of an LED is large enough that it falls on the rough search grid. Obviously, if the initial coarse grid is chosen to be too large, we could miss finding the LED. Since even when N is set to 2 we cut down the computation by 75%, and the LED sizes in the images are at least 4 pixels, we can reduce the latency a great deal and not suffer from losing track of the LED. An enhancement to this mechanism is to use prediction from the previous frames to guide our search for the LED in the current frame. For LEDs found in the previous frame, their respective velocities can be approximated by simply differentiating the positions in the last two frames [1]. This information is then used to predict the location of the LED in the current frame. The prediction locations are then used as the seed of our search in the new frame. Only the neighboring regions of these predicted locations are searched initially, and only when these searches yield no LEDs do we fall back to a full frame (grid-based multi-resolution) search. Our experiments have found that one can use less subsampling with this enhancement when LEDs don't move fast in the image, and thus don't go in and out of the field of view of a camera that frequently. This means that it works better with wide angle cameras.

Figure 2.4 shows one image seen by a camera with the detected LED location overlaid. The predicted location and the surrounding search region is also shown.

[1]The velocity estimated by this simple differentiation by a single camera is not very accurate due to all kinds of noise in the measuring process. A more accurate and smooth velocity estimate based on data from multiple cameras can be provided by the server, and will be discussed in Section 2.2.4.1.

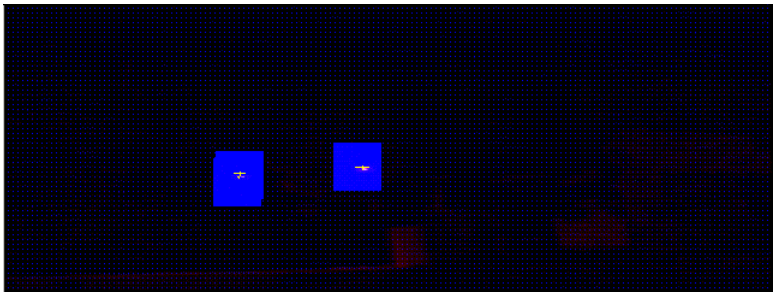


Figure 2.4: An NTSC video field from a camera with two detected LEDs and their prediction regions shown in blue. In the next frame, the client will search in the prediction region first for bright LEDs rather than searching the whole frame exhaustively.

2.2.3 Central state-space estimator

In this section we describe the central estimator and how it estimates target 3D motion from 2D information provided by the many feature trackers. We will focus on how, by employing an EKF properly, it can enable the use of many asynchronous and heterogeneous cameras, and how it supports the tracking and labeling of multiple points.

2.2.3.1 Tracking using asynchronous cameras

In this section we describe how the central estimator tracks motion using many asynchronous cameras. It is able to utilize inputs from many such cameras by employing the Kalman filter in a non-conventional manner. As a by-product, cameras with heterogenous parameters are supported at same time. This section contains a brief introduction to Kalman filtering, then focuses on our specific way of applying it to support asynchronous and heterogeneous cameras. A more extensive introduction to Kalman filtering and the specific models and parameters used in our system can be found in Appendix A.

The Kalman filter is a set of mathematical equations that enables the recursive estimation of the state of a process, given a sequence of discrete, noisy measurements of the process from one or more devices. The filter requires the prior knowledge of two models: a process dynamic model that allows one to predict the state at a time instant from the state at another time instant, and a measurement model which allows one to predict what the measurements on the devices should be given the process state. The filter operates in a

predict-update cycle. At a given time instant when a measurement (which is a vector of multiple scalars) comes in, and given the state at the time instant of a previous measurement, the filter predicts both the process state and the measurement corresponding to the predicted state. It then compares the predicted measurement with the actual measurement reported by the devices, The discrepancy between the actual and the predicted measurements is used to generate a correction for the filter's current estimate of the process state. Conceptually, the final updated state is a state that should map to a measurement that is a weighted average of the predicted measurement and the actual measurement. The weights are inversely proportional to the noise levels in the process model and the measurement model. The noise levels of the process and measurement models are described by a matrix $\mathbf{Q}$ and a matrix $\mathbf{R}$, respectively, and are also assumed to be known a priori.

The above is the general formulation of Kalman filtering. Of course, the appropriate dynamic and measurement models must be defined for the Kalman filter to be used for a specific application. Indeed, it is through the definition of the measurement model and the manner in which it is used in the filtering operation that M-Track is able to support asynchronous and heterogeneous cameras, and distinguishes itself from other Kalman filter based tracking systems.

For clarity of discussion, let us just consider the case where the target is just a single rigid object with M LED feature points. In this case, the process model is very straightforward: the process state is simply the position, orientation, and their derivatives, and the common constant-velocity model is used to predict the next state (see Section A.3 for the details). For the measurement model, we first need to define what should be included in the measurement vector. In the standard definition of Kalman filtering, the measurement vector should include the *complete* set of measurements from all devices for all feature points. Since each camera will report the 2D locations in its image plane for all of the LEDs, i. e. $\mathbf{z}_i = (u_{i1}, v_{i1}, \ldots, u_{iM}, v_{iM})$, the measurement vector would be the $\mathbf{z}_i$'s of all cameras concatenated together, i. e. $\mathbf{z} = (\mathbf{z}_1, \mathbf{z}_2, \ldots, \mathbf{z}_N)$ where N is the number of cameras. This means that the system would need to obtain data from all cameras before a new estimate of the target 3D position can be computed.

In a system with multiple synchronous cameras, the above definition of the measurement vector is appropriate, since measurements of the point from all of the cameras are

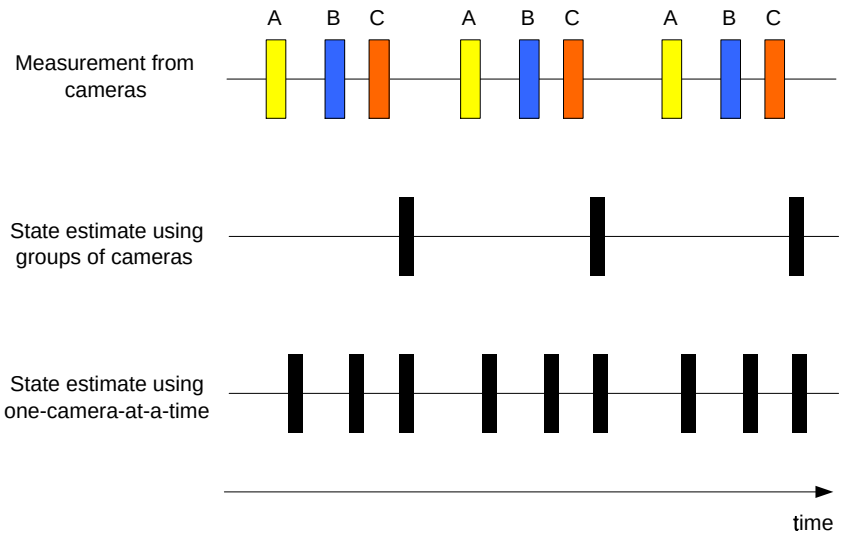


Figure 2.5: Timing diagrams of state estimates from three asynchronous cameras either by treating each set of measurements as simultaneous, or preserving the time information associated with each distinct measurement. latter approach results in higher state estimation rates and more accurate estimates.

made and available at the same time. Intuitively, the use of multiple camera measurements to compute a single position estimate also makes sense because any individual measurement offers only partial information. For example, at least two cameras are needed to determine the 3D position of a point, and more than two cameras produces an overdetermined system and provides further noise reduction. However, in a system with asynchronous cameras, the measurement from each individual camera comes *sequentially*. In this case, there are a couple of problems associated with the use of the measurement vector described above. One is that, since the system has to wait until measurements from all cameras are available before it does an state estimate (as shown in Figure 2.5), this results in a relatively low estimation rate and therefore potentially degraded estimation when the target has high dynamics. Another problem is that, since grouping measurements from different cameras into one measurement vector implies that they are made at the same time instant, estimation error arises when sequential measurements of a moving target are mathematically treated as if they are simultaneous.

Given the above concerns and inspired by the “single-constraint-at-a-time” (SCAAT)

principle proposed by Welch [88], the estimator in M-Track uses the Kalman filter in a "one-camera-at-a-time" manner. Similar to SCAAT, there is no explicit notion of the "complete" measurement vector; instead, the measurement vector at any one time is a "partial" vector that includes a small subset of the complete possible measurement set, and this subset is different at different time instants. In contrast to SCAAT, instead of purposely using one single device and source pair (i. e. one LED position for one camera) at a time to compute a state update, M-Track groups all 2D LED positions for the *same camera* to compute a state update. Figure 2.5 shows that this approach results in an increased state estimate rate over that attainable by grouping the measurements from all cameras. More important, measurements that are made asynchronously are indeed treated mathematically in the Kalman filter as separate non-simultaneous measurements, resulting in more accurate state estimation.

In order to do "one-camera-at-a-time" processing, the standard Kalman filtering cycle needs to be modified slightly to accommodate the partial measurement vector that comes in sequentially. Figure 2.6 illustrates the modified Kalman filtering cycle in M-Track. When the 2D locations are reported from a camera c, along with a timestamp, to the Kalman filter, the filter goes through the following steps (given the previous updated state and time of that update):

1. Compute the interval Δt between the current time and the time of the previous state update.

2. Predict the current state from the previous state, using the process model $\mathbf{A}(\Delta t)$ which is usually a function of Δt.

3. Predict the measurement of camera c which corresponds to the predicted state using the measurement model for camera c. Note that since the measurement model maps 3D target positions onto 2D image locations of a specific camera c, it is dependent on camera parameters such as its location, orientation, focal length, etc. Therefore, a different measurement model needs to be used for each camera, and is denoted as $h_c(\bullet)$. The measurement noise matrix is also dependent or parameterized based on the camera and is thus denoted as $\mathbf{R}_c$.

4. Compute the discrepancy between the predicted and actual measurements, and update the state estimate based on the discrepancy. The updated state and current measurement time are recorded and are priors for the next filter cycle.

As can be seen from Figure 2.6 and the above description, the main difference between the Kalman filter in M-Track and a standard Kalman filter is that the measurement model and prediction equations are different in every cycle, and at each cycle, only partial observation of the process is incorporated into the filter. However, over time the filter fuses the individual incomplete observations to provide a globally observable system. The Kalman filter inherently provides the means for this fusion because the knowledge of the process dynamics and the prior state allows the estimation of the state even in the case of an incomplete observation, since that observation is used to merely "correct" the prediction from the process model. Through the use of a camera-dependent measurement model $h_{\mathbf{c}}$ and measurement noise $\mathbf{R_c}$, we can accommodate not only cameras that take measurements asynchronously, but also cameras with heterogeneous parameters such as rate and resolution.

Besides providing faster state estimation rate and better accuracy, M-Track's ability to appropriately accommodate asynchronous and heterogeneous cameras also implies better flexibility for the overall system, because cameras are now treated individually. The occlusion, malfunction, or failure of one or a few of the cameras would therefore not cause failure of the whole system. Individual cameras can be activated, removed, or upgraded without disturbing system operation. No requirement on camera synchronization also makes the deployment of new systems or upgrade of existing systems much easier. Moreover, this formulation facilitates the incremental calibration of the cameras in the system (see details in Chapter 3), which makes the addition of new cameras or reconfiguration of existing cameras a much less labor-intensive process.

2.2.3.2 Support for tracking multiple independent points

In the previous subsection the target being tracked is modeled as a single rigid object. That implies that, even though there are multiple feature points on the target, the relative positions of these features stay fixed with respect to the local target coordinate frame. But

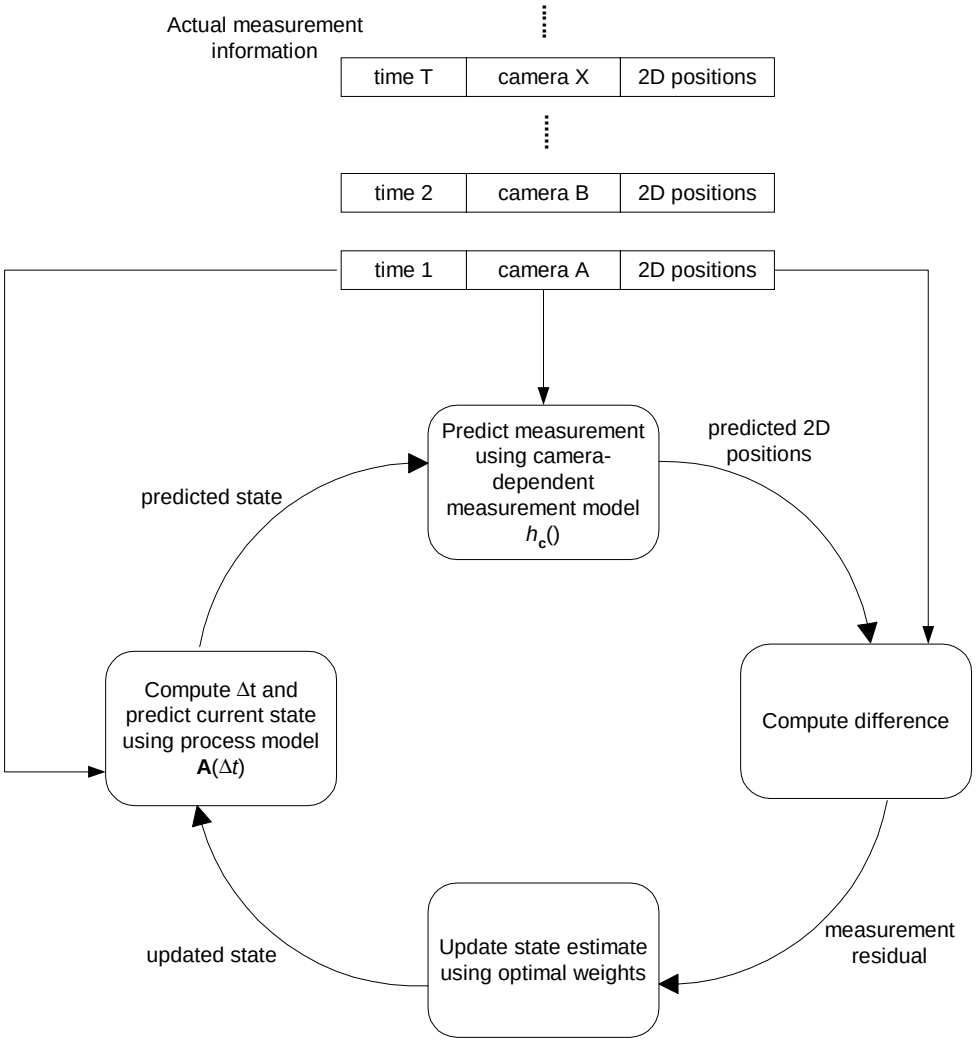


Figure 2.6: The one-camera-at-a-time Kalman filter loop running on the server.

this rigid model does not work for the cases in which the feature points move relative to each other. For example they could be totally independent LEDs held by different people; or when multiple LEDs are mounted on various parts of a human body, they move relative to each other. In these cases, a single position and orientation vector is not sufficient to describe the motion. In order to be able to track in these cases, we have chosen to model the target simply as multiple independent points moving in 3D, each point having a position and velocity vector associated with it. While points are not really moving independently in the full body tracking example, and the motion is modeled as that of an articulated figure as in much of the literature [12, 37, 69, 54, 26, 51, 25, 58, 72, 10, 24, 45], we still use the multiple-point model because the algorithm to track is simpler and faster, and adequate for real-time tracking. The focus of this thesis is to address the system-related issues associated with scaling a real-time tracking system to many cameras. We consider the employment of more complicated targets as an orthogonal issue and future work.

Now we describe how the central estimator is adapted to track the motion of multiple independent points. First note that, if we know a priori how many points there are, and assuming they don't disappear, tracking them is very straightforward. Each single point has a state vector (with position and velocity only) $\mathbf{x}_m = (\mathbf{p}_m, \hat{\mathbf{p}}_m)$, and the full state vector is the concatenation of all M points. We can basically use the filter algorithm described in Appendix A.3 for tracking a rigid object, except that there is no need to maintain orientation information.

However, in some applications the points can appear and disappear from the scene, and tracking in these situations is more challenging (see Section 2.3.2). At every filter cycle, we need a way to detect whether new points have appeared and/or old points have disappeared. We handle this in the data association step in the filter cycle.

As described in Appendix A.3, at the data association step we match a set of observed 2D dots with a set of predicted points. (In the next few paragraphs we use "dots" to imply they are actual observations and "points" to imply their are predicted from the internal model of the filter.) The multi-point case is different from the rigid body case in that we may not always find a match for all dots or points. A newly appeared dot will not have a corresponding point because the filter is not aware of it; and a predicted point may not have a corresponding dot because either that LED has been occluded temporarily or it has

completely disappeared from the scene. In order to detect these cases, we introduce the notion of *verify region* of a predicted point. More specifically, no prediction is perfect and thus every point has some error margin associated with it. The 2D region where that predicted point is considered "valid" is called the *verify region* of that point. If a dot is inside the verify region of a point it is considered "valid" and is a potential match for that point. This region is usually modeled as an ellipse that has a size dependent on the error covariance matrix $\mathbf{P}$ and measurement Jacobian $\mathbf{H}$ (see [5] for its mathematical expression). Intuitively we can think of it as the 2D projection onto the camera image plane of the 3D error ellipsoid of the feature point position in space. If there are multiple "valid" dots for a point, the dot that is closest in distance to the point is chosen to be the match. We repeat this process for every point and at the end are left with a bunch of unmatched dots and/or unmatched points.

An unmatched dot is a potential new point to track and the filter passes it back to the calling application. If the application confirms that it is actually a new point and initializes its position and velocity, the filter expands its internal state vector $\mathbf{x}$ to include the new $(\mathbf{p}, \dot{\mathbf{p}})$ elements. In this way, a new point is added to the model of the Kalman filter. An unmatched point can potentially be deleted. However, it could be due to temporary occlusion rather than disappearance from the scene. The filter simply passes it back to the application, and it is the application's responsibility to distinguish between the two cases. Usually the application sets a time-out threshold and a point that has not been seen (i.e. matched) for a period longer than the time-out threshold is considered gone, and its corresponding position and velocity elements are deleted from the Kalman state vector. Note also that, as a free by-product, points that have disappeared for less than the time-out threshold will be kept in the filter and their states will continue to be predicted by the dynamic model. What this means is that if a point is occluded temporarily, it will get updated when it is seen again, and no re-initialization is needed. This makes the system more robust.

2.2.4 Networking module

In this section we describe the communication between the 2D image processing program (which we call the "client") and the central estimator (the "server").

2.2.4.1 Communication protocol

The network topology between the clients and the server is logically a "star". Each of multiple clients processes a video stream from a camera and communicates with one central server. Information is passed between the client and server via TCP/IP packets over the Ethernet. Each client/camera has a unique name; when a new client makes a connection to the server, it needs to tell the server its name. The server maintains a list of current clients to which it is connected. When the server is running, new clients can join and old clients can disconnect, without restarting the server.

In normal tracking situations, the signal flow is one way from the client to the server. Packets sent from a client to the server contain 2D feature information such as the number of observed features, their locations, sizes, and the time when they are seen. (In the case of heterogeneous cameras, parameters of individual cameras also need to be sent from client to the server at system startup.) Although this is sufficient to enable the tracking functionality of the whole system, this simple protocol has limitations with respect to making the system sufficiently robust and practical, especially with many clients/cameras physically spread over a wide area. For example, sometimes the threshold for the 2D feature detection process needs to be individually adjusted for different cameras. It would be very time consuming to log onto various machines and do that one by one. More important, the client could also benefit from information only available at the server, such as the 3D velocity of the target, which could be used to direct a 2D feature search, such as for 2D position prediction as proposed in Section 2.2.2. Based on these considerations, we make the communication two-way and allow the server to send information back to the clients. In the current M-Track implementation, the server can send commands such as "restart", "increase/decrease threshold", "send me raw video rather than 2D features", etc. Consequently, the information that a client sends to the server can vary depending on the client's state and what the server requests. Figure 2.7 lists some of the commands or requests that

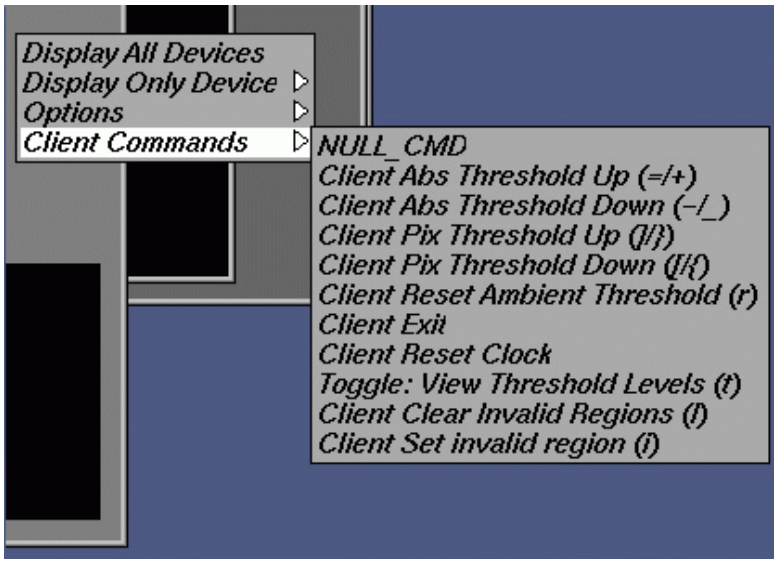


Figure 2.7: A snapshot of the "client command" menu of the server, which lists the commands that a server can send to a client.

a server can issue to a client. For a more specific example, see Figure 2.8. Note that part of this picture may look much like Figure 2.3, but the important thing to keep in mind is that this threshold information, which belongs to a client, is now available and adjustable on the server side. Also, as mentioned in Section 2.2.2, the 3D velocity estimates provided by the Kalman filter on the server could be sent to the client for predicting the 2D LED locations and reducing the latency in 2D feature detection.

2.2.4.2 Timing issues

As described in Section 2.2.1, we have made the conscious choice of not synchronizing all of the cameras. As a result, observations from different cameras are not necessarily made at the same time, but the dynamic model in the Kalman filter keeps them correlated in time. However, the use of the Kalman filter does require that all samples have a universal notion of time, e. g. the client and server programs need to have the same clock. This enables measurement samples to be accurately time-stamped. In addition, because communication between the clients and the server is TCP/IP based and on the local area network, packets from different clients might arrive at the server out of order. In this section we describe

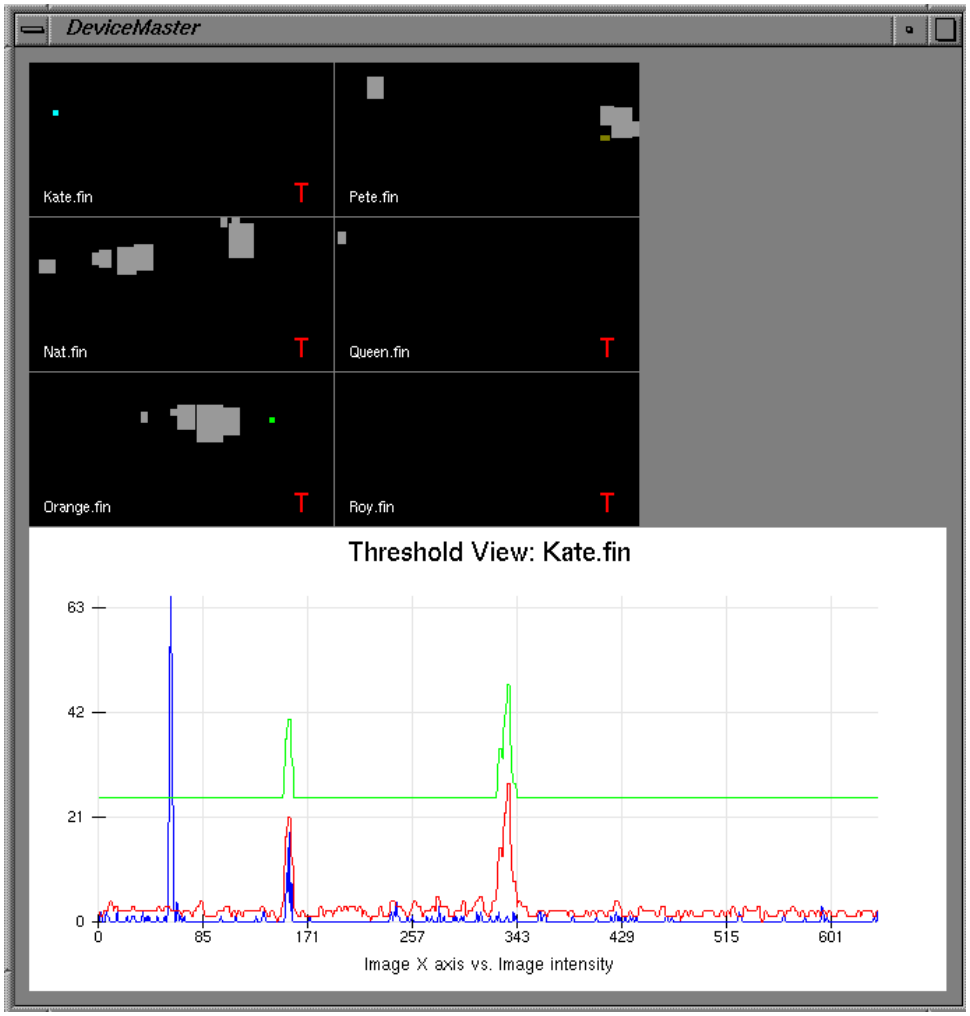
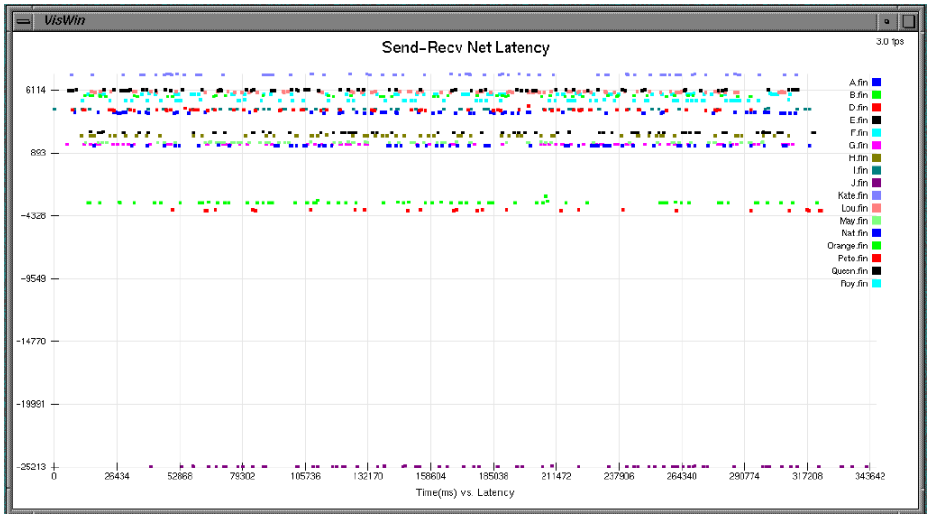


Figure 2.8: A snapshot of the server's visualization module, which updates and visualizes, in real-time, all kinds of information that the server has. In normal operating mode, each small subwindow of the top part of the window corresponds to a camera, with camera names shown as "Kate.fin", etc. Each subwindow shows its detected feature locations on the image by the small colored dots. The gray area is the non-active area of the screen, and the big square in the bottom half just shows the zoomed-in view of a particular camera. But the server can also request the client to send other kind of data as well. In this case, the threshold information on the client side represented by a scanline. The red line is the captured background; the green is the per-pixel threshold based given the background and thresholding algorithm; and the blue is the current video scanline. Note that the blue peak corresponds to the bright feature point that corresponds to the cyan dot shown in the sub-window named "Kate.fin".

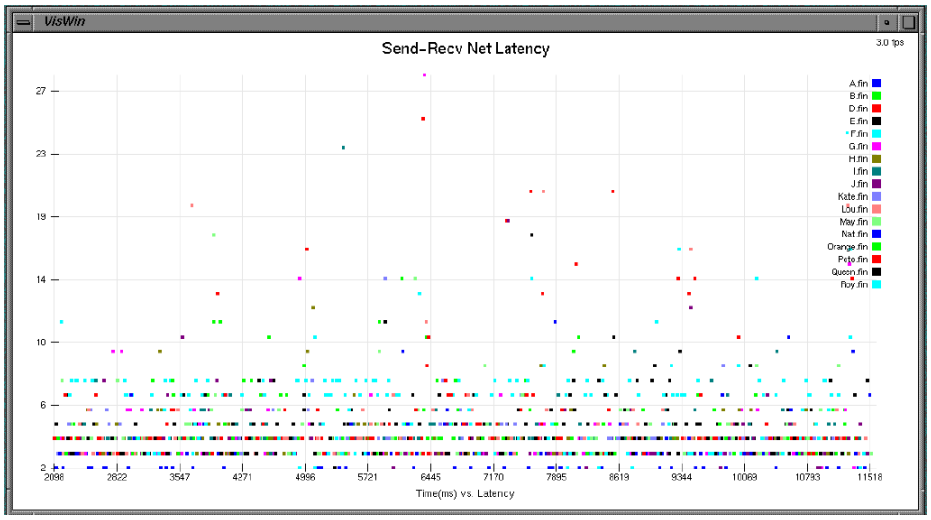
how we handle these two issues.

Keeping time universal on all computers Most of our clients run under UNIX and some of them run under WINDOWS. On both of these systems, some publicly available program implementing the Network Time Protocol (NTP) [67] can be run, ideally to adjust the clock on the computer with respect to some standard time server. However, our experiments and measurements show that even though the clock of most clients can be kept within 10ms of the others, some client clocks still drift much faster or slower than the rest of the client clocks. Since in the Kalman filter we only process the most recent packets and drop the "out-of-date" packets, a very slow client clock will cause the data from that client to never to be used, and a very fast clock will advance the notion of time in the filter too fast and cause data from other normal clients to seem old and get dropped. In both cases, data from certain cameras will never get used, an undesirable phenomenon that we call *starvation*. To correct the clock drift, we have a separate *time daemon* running on the server. At run time, each client periodically "asks" the server time daemon what time the server thinks it is, and based on that, estimates its time drift with respect to the server. This value of the drift is then added to subsequent timestamps on all of the observations that this client reports. We have verified in practice that this approach has resulted in higher utilization of client data and has prevented camera starvation. As an example, Figure 2.9 shows the time offsets of packets from various clients from the current server time, with or without the correction of clock drifts. It can be seen that the variance of time stamps are much reduced with the drift correction and there is no client starved due to a constant clock drift.

Handling of out-of-order packets Even if the clocks on all of the client computers are perfectly synchronized, packets from different clients can still arrive at the server out of order. This could be due to uneven network traffic delays, or could be due to a certain client computer being loaded and thus having slow processing. A client that is consistently slower (even slightly) than another can still starve, due to the fact that the server only processes the most "current" packet, where the notion of "current" is defined by the most recent timestamp of a client packet received by the server. There are several ways of handling these out-of-order packets. One simple solution is to have a reorder buffer. As illustrated



(a)



(b)

Figure 2.9: Effect of drift correction. The y-axis denotes the relative offset of a packet's timestamp given by the client from the current server time (a) without drift correction and (b) with drift correction. The x-axis denotes the time since the start of our experiment. Because the server always uses the most "recent" packet at any time instant, only the packets that are at the bottom of each graph are used by the server to estimate target motion. In (a) it can be seen that clients have consistant time drift from each other, and since one client's clock is always much ahead of the others, packets from that client would always get used and other clients would be starved. And in (b), after employing drift correction, no camera is consistantly ahead or behind others, and data from all cameras have a chance to get used.

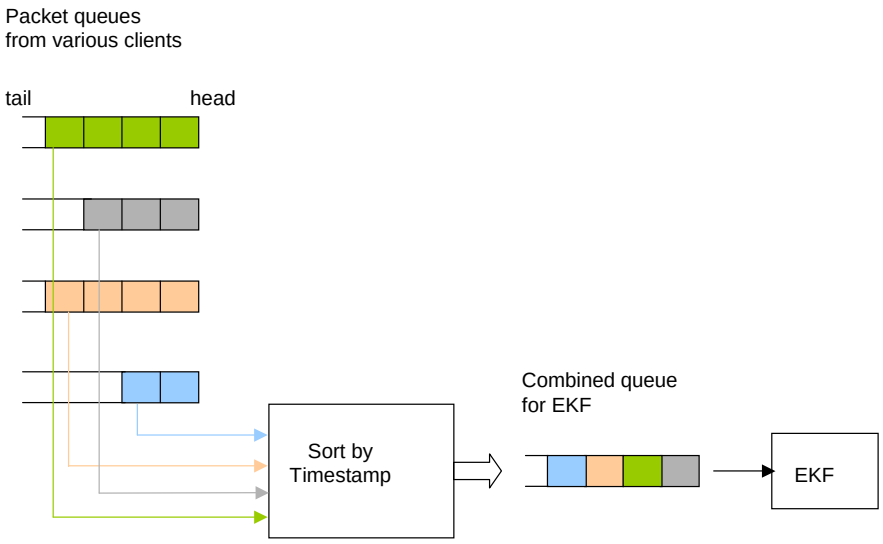


Figure 2.10: A reorder buffer can be utilized by the server to sort the packets by their timestamps from the clients in order to prevent camera starvation.

in Figure 2.10, the server reads the last packet from each of the client queues, sorts them by time, and feeds all k to the Kalman filter. This approach prevents the camera starvation problem and increases data utilization, but introduces additional latency. Since the packets are fed to the filter in a "batched" way, it takes the filter longer to process the whole batch. When the filter gets to the end of the batch, a significant amount of time has already passed and the data at the end of the batch is already "old", resulting in more inaccurate state estimates. In this situation there is usually already a new sample in the queue for that camera, and we would like to use the most up-to-date sample if possible. Thus, another alternative is, instead of reading and processing data from *all* available client queues every time, to randomly pick a camera and only read and process data from that camera, unless there is no data from that camera (because the target is not in its field of view) and we pick another one. On average we will use data from all cameras. Because only data from one camera is fed to the filter, it takes the filter significantly less time to process. The filter is therefore more likely to be able to keep up with the processing load. This approach has been verified in practice to sufficiently improve data utilization, avoid camera starvation, and result in less latency.

2.3 Applications

2.3.1 3D head tracking for the Interactive Mural

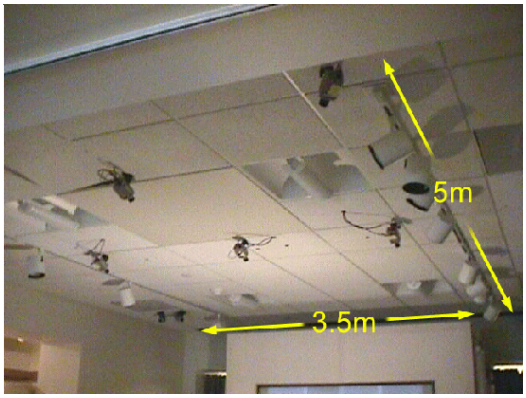
In this application we use M-Track to track the head position of a user to drive the rendering of a 3D graphics display system. The Stanford Interactive Mural is a large format display that is driven by a high performance graphics engine [33, 32]. It can display both 2D and 3D content in high resolution. In order for the user to have a more realistic and immersive feel of the 3D content, it is preferable to render the 3D model from the point of view of the user. To enable tracking of where the user is in 3D, the user needs to wear a hat with some LEDs attached, as in Figure 2.11. Ten cameras are mounted on the ceiling to cover the space in front of the Mural, as shown in Figure 2.12. The cameras are placed and set so that the area close to the Mural screen is covered with higher resolution than the area that is farther away from the screen. The reason for this is that a change in the user's position would require a much larger change on the display if the user is closer to the screen than if he/she is farther away. For example, the displayed contents are more sensitive to the user's position change in the near region. Therefore, the cameras are arranged to take this uneven sensitivity into account.

We model the user's head as a single rigid object with n feature points, and use the Kalman filter described in Section A.3 to track its position and orientation as user moves in front of the Mural. This information is then sent to a 3D rendering program that will render the 3D scene from the point of view of the reported viewer position, as shown in Figure 2.13. Each camera is sampling the scene at up to 60Hz and the latency between the digitization of the video to the output of the user's position is about 30ms. This application demonstrates the potential for such a real-time tracking system as an input technology for virtual reality.

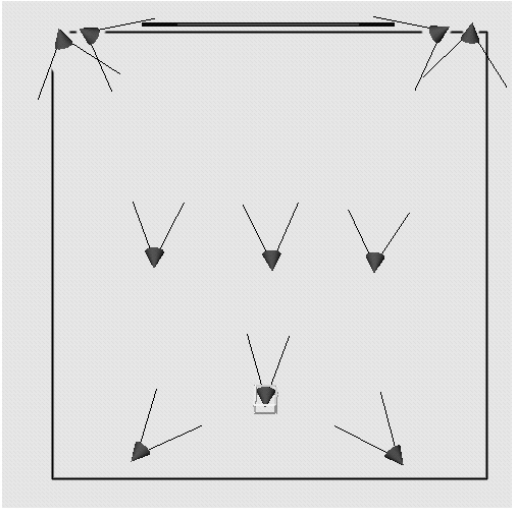
2.3.2 LumiPoint

LumiPoint is another M-Track based input system for the Interactive Mural. The high resolution of the Mural display (3796 pixels by 1436 pixels) enables the detailed display and

Figure 2.11: A video frame from a camera showing a person wearing a hat mounted with bright red LEDs.



(a)



(b)

Figure 2.12: Camera setup for head tracking for the Interactive Mural. (a) A picture of the physical setup on the ceiling in front of the Mural; (b) A plan view visualization of locations and orientations of the cameras.

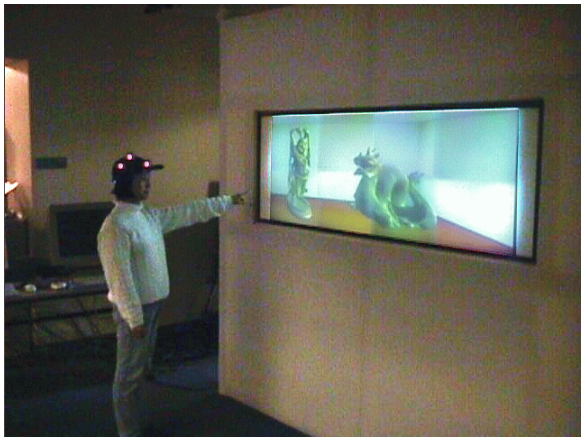
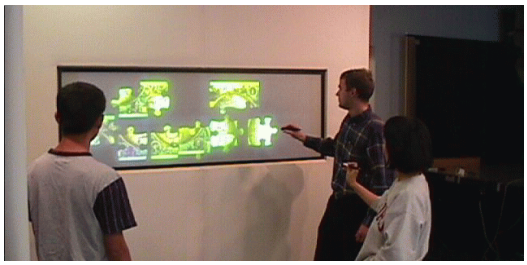


Figure 2.13: A user's head position is being tracked to drive the display of a "virtual museum" application on the Interactive Mural.

(a)



(b)

Figure 2.14: Colleagues collaborate on, and interact with, a wall size visualization system. In the right image, multiple users use a laser pointer as a pointing/writing device to jointly solve a jizzaw puzzle through the LumiPoint system, which can track multiple independent traces of the laser pointers

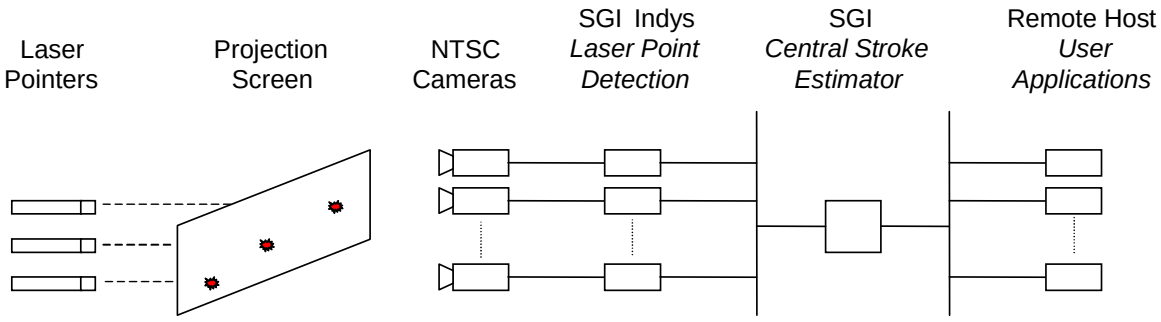


Figure 2.15: Laser pointers are used as input devices on a wall size display. An array of cameras and CPUs placed behind the screen detects the projected laser spots. Information from individual cameras is aggregated into a stream of strokes by the central estimator. Finally, filtered user input is provided to visualization applications.

exploration of complex data sets; the large physical display surface (6 ft. by 2 ft.) accessible by a group of people makes sharing and collaborative interaction with the data possible. LumiPoint is an input system that is designed specifically for multi-user collaboration on such a large format display. It allows any number of users to interact simultaneously with the large display, and scales well with display size, resolution and number of users.

In this system, each individual uses a laser pointer to directly manipulate data displayed on the screen. Cameras oriented towards the display surface observe all laser spots on the display directly. Each camera is connected to a CPU that digitizes the incoming video streams and finds all laser spot locations within the camera field of view at each time instant. This data is communicated to a central estimator that determines, based on time coherence and motion dynamics, whether the laser spot is the beginning, continuation or end of a stroke. A stroke is the continuous path of a laser spot on the display surface from the appearance of the spot to its disappearance. This information, together with an estimate of the position, velocity and acceleration of each active stroke, are available for use either directly by a visualization application or indirectly after interpretation by a gesture recognition module. An example of how several users can collaborate using laser pointers on the Mural is shown in Figure 2.14. The fact that a laser pointer is by definition a pointing device, and that it is usually shaped liked a pen, makes it a natural and intuitive device for interaction with a wall-like display.

The overall hardware and software architecture behind LumiPoint is basically the M-Track architecture, with multiple image-space detectors (one for each camera) and a central estimator, as shown in Figure 2.15. The only difference lies in the specific models and algorithms used in the central estimator.

First of all, compared to the 3D rigid object tracking example, the state vector now contains just the position and velocity of the laser spot on the 2D screen. Since each camera client effectively measures the 2D quantity directly[2], the matrix $\mathbf{H}$ in the measurement model in equation (A.2) is trivially the identity matrix, as opposed to the non-linear projective mapping in the 3D rigid object case.

The second issue that we need to address is the support of multiple users. By using the data association techniques presented in Section 2.2.3.2 for automatically adding and deleting points, the central estimator is able to report to the applications stroke-begin and stroke-end events, as well as the current locations of the continuing strokes, as shown in Figure 2.16. However, how do we distinguish and support multiple users?

Our early multi-user applications tended to assume that each user would be identified uniquely based on laser pointer color. We have both red and green laser pointers, and additionally experimented with making light pens from a variety of LED colors. While this is the most straightforward way to retrofit existing single user applications, many interactions do not actually require knowledge of which physical pointer is actively in use. Rather, the context within which the strokes are placed in the application is of primary importance. For example, there may be no need to identify which user selected an option from a menu or edited a document, only that the action occurred. Furthermore, some interface technologies are fundamentally context free. For instance, Bier et. al. [8] discussed click-through-tools. This interaction technique explicitly replaces the standard user tool palette with on screen transparent lenses that affect underlying data when a user clicks on them. Additionally, even when applications store context such as preferences on a per user basis, physical tags may not be the only way to disambiguate context. For instance, Rekimoto [70] uses temporal coherence of strokes to determine which of several identical user styluses has generated a stroke on the whiteboard in the Pick-and-Drop system.

[2]To put it more precisely, the image-space client reports the 2D location of the laser spot in the camera image plane, but the conversion from camera image coordinates to the screen is simply a linear mapping called homography (a 3x3 matrix multiplication), which is pre-measured.

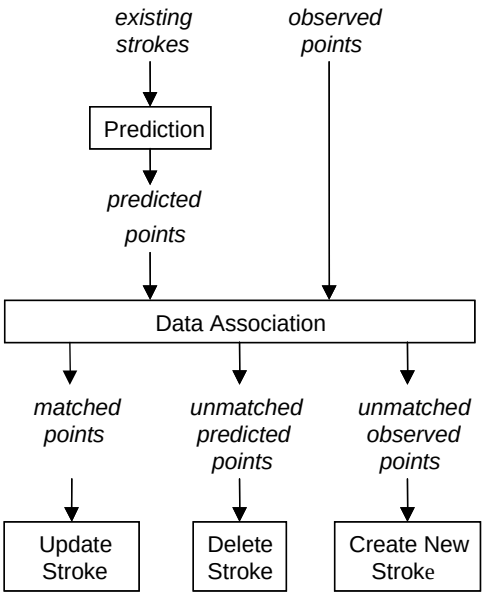


Figure 2.16: The central estimator associates laser points observed by the tracking subsystem with points predicted by existing Kalman systems (strokes). Matched points revise the relevant stroke state, while unmatched points cause strokes to be added or deleted.

Given the variety of user interaction methodologies available to applications, we abandoned the assumption that a physical identifier is required. Currently, in addition to laser color each event is tagged with an ID field, indicating to which stroke it belongs. Although the user application may not know from which physical laser pointer a stroke originates, user applications can resolve consistent click and drag sequences, even when multiple strokes of the same color are active in a single region of the work space. Currently, when multiple users interact using our system, the primary modality is with red laser pointers only. More details on the design, implementation and performance discussions can be found in [23].

To summarize, the LumiPoint application demonstrates the potential of M-Track to support multi-target tracking.

2.3.3 Simultaneous body and face motion capture

In many situations it is desirable to capture detailed motion embedded in a large volume. For example, in the entertainment industry, the captured body motion as well as facial motion of an actor/actress can be used to drive the animation of a virtual figure. However, due to the fundamental trade-off between the field of view and resolution of a camera, current camera-based motion capture systems can usually only operate in one scale. In other words, cameras are either spread out to capture the body motion; or they are set to be more close up to capture a person's facial motion. Moreover, while capturing the facial motion, the actor's head can only move within a very small region and in a very restricted way, so that cameras can see the whole face. This restrictive mechanism for facial capturing limits the possibility of acquiring realistic facial motion because certain motions will only occur when the actor can move and act freely in a larger volume. Recently there have been a few examples of *foveated*[3] systems in which one or multiple steerable pan/tilt/zoom cameras are guided to follow the moving object and zoomed in onto the interested portion of the object [66, 6, 9, 30, 20, 73]. However, the location of the moving object within the large working volume is either manual and relies on a human operator (such as the Eyevision system used in broadcasting the 2001 Superbowl [81], or is based on 2D information from another single wide-angle camera, which tends to often lose track of the target due to occlusion in the scene while the target is moving.

We have developed a new 3D model-based mixed scale motion recovery system, that can simultaneously track the motion of a human body in a wide area and capture the person's facial motion while he/she is moving. An overview block diagram of this system is shown in Figure 2.17. A multi-camera, wide-area subsystem tracks the LED features on the target's body and reports their 3D positions. Based on this information, the foveated camera control subsystem determines where the pan/tilt/zoom cameras should be pointing at and steers them as required. Video streams from these foveated cameras are then recorded and analyzed to recover the person's facial motion. The detailed design principles, issues, solutions and performance evaluation of this system are the subject of a separate document [22, 21]. What we would like to point out here is that the M-Track architecture is used to

[3]The word "foveated" comes from "fovea", which is a small rodless area of the retina that affords acute vision. It is used to describe tracking systems that provide a small but high-quality sub-working volume.

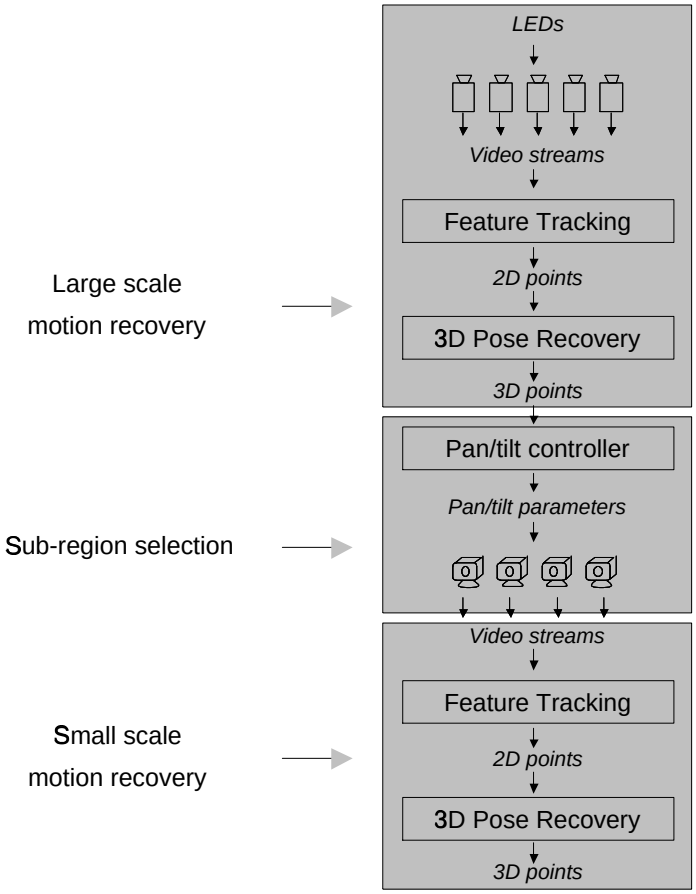


Figure 2.17: Overview of the mixed scale motion recovery system. M-Track is used for the large-scale motion recovery.

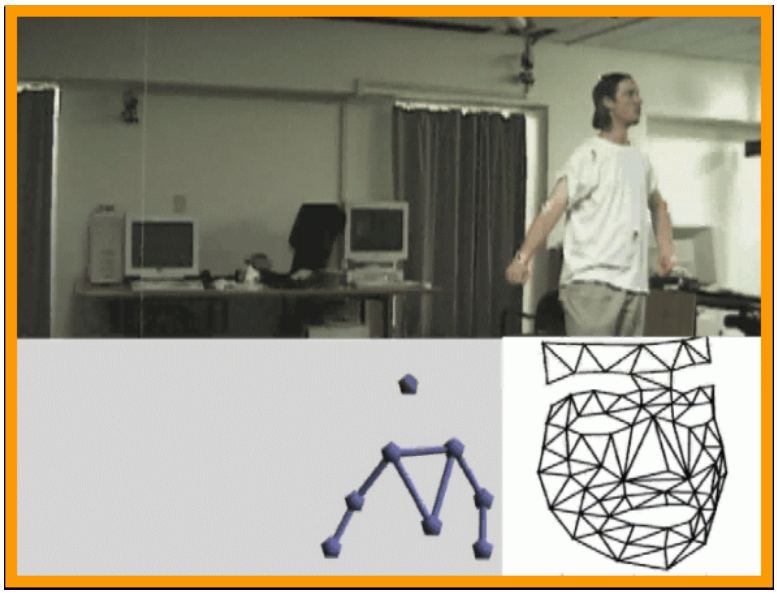


Figure 2.18: Simultaneous face and body motion recovery using our mixed scale system. M-Track provides the wide area tracking for body motion and the rough location of the person's head. Pan-tilt-zoom cameras are then guided to point to the the person's face and record the facial movement in higher resolution. Recorded video streams from multiple pan-tilt-zoom cameras are later analyzed to recover the facial motion of the person.

build the wide-area subsystem to track the various LED feature points on the person's body. The multiple cameras not only provide wide coverage range, but also reduce the likelihood of total loss of track of the target caused by occlusion. This is more robust than the 2D based foveated systems. Furthermore, using the Extended Kalman filter and its ability to track multiple points and maintain their time correspondence, we are able to "lock" onto a particular feature (say, the LED on the person's head), and have an automatic way to direct the guidable cameras. In addition, because we have a dynamic motion model in the central estimator of M-Track, the position and velocity estimate of the target is used by the foveated camera control unit to predict a future target location for the foveated cameras to look at, in order to compensate for the latency in physically steering the cameras to their desired configurations. Figure 2.18 shows an example output of this mixed scale system.

2.4 Discussion

M-Track is an scalable architecture specifically designed to do real-time motion tracking with a large number of asynchronous, possibly homogenous cameras. It uses many client CPUs to perform 2D mage processing distributedly, and uses a central server to integrate the 2D information from the clients and estimate the target motion. The central estimator, based on extended Kalman filtering, is not only able to incrementally assimilate asynchronous inputs, but also able to track multiple features and enable the continuous and automatic labeling of these features once they are labeled in the initial frame. The central estimator also allows the continuous tracking even when some feature points are temporarily occluded. Three end-to-end VR and motion capture applications, namely the head tracking, LumiPoint and simultaneous body/face capture, are implemented to demonstrate the effectiveness of the architecture. The number of cameras used in these three applications range from 8 to 26, all of them are asynchronous. In both LumiPoint and simultaneous body/face captures, there are multiple independent target points appearing and disappearing, either due to occlusion, or due to the user's conscious activation and deactivation. And M-Track can track all of them reasonably reliably. The fact that M-Track is able to track all targets accurately and robustly enough to enable the fulfillment of the final task demonstrates the validity and effectiveness of the architecture.

Chapter 3

Calibrating Distributed Camera Networks

3.1 Introduction

Many applications of tracking and observation require operation over a wide area, such as monitoring the traffic flow of vehicles in a parking structure or people in a building. In such cases, a single camera is unlikely to be sufficient. Rather, a network of interconnected cameras is required, each of which functions over only a small subset of the total area. In order to build such a system, one important issue that must be addressed is calibration of the cameras into the same global coordinate system. In this chapter we address camera calibration in a wide area sensing environment. Wide area system calibration is much more challenging than calibration of a single camera. In single camera calibration, the usual method involves placement of a carefully instrumented calibration target in the field of view. Based on correspondences between known 3D features on the target and their 2D locations in the image, calibration can be obtained. If multiple cameras are active in the same working volume, then each can be calibrated individually using an identical process.

The case of wide area calibration introduces a number of difficulties. Cameras each cover only a small subset of the total working volume. A calibration target can be moved so that each camera is calibrated separately. However, the usual method for global calibration across all cameras requires knowledge of all calibration object positions in a common

reference frame. This is difficult to obtain without expensive instrumentation. The synchronization of cameras assumed by the usual method poses an additional problem. In large systems with many heterogeneous cameras, it becomes difficult to ensure that all cameras record observations at exactly the same moment.

In this chapter we introduce a method that calibrates a system of asynchronous cameras into a single global coordinate system without requiring physical measurement of the position of a calibration object. A rough estimate of each camera's pose (i.e. location and orientation) is obtained using standard structure-from-motion techniques. The rough camera calibration can be used to track the path of a point moving through the entire working volume. This path defines a virtual calibration object, which can be used to improve the estimate of camera pose in the global coordinate space. Iterating the above process results in convergence to a precise estimate of camera pose as well as the point path defining the virtual calibration object. We evaluate our method by comparing it to traditional calibration techniques. Furthermore, we demonstrate its effectiveness in wide area settings by calibrating a multi-camera indoor tracking system where cameras cover disjoint viewing regions, a more challenging situation where traditional methods cannot be easily applied.

The rest of the chapter is organized as follows. Section 3.2 provides background and discusses previous work. Section 3.3 describes our proposed method and Section 3.4 gives experimental results for relevant application scenarios.. We summarize in Section 3.5. (A concise version of this chapter is published in [15]).

3.2 Background

3.2.1 Basis of traditional camera calibration

Camera model

Since one of the goals of computer vision is to perform metric measurements from images which are acquired using cameras, it is important to have quantitative models for these measurement devices. One model, called the pinhole camera model, is widely used in computer vision and its measurement mechanism can be easily modeled based on simple geometry.

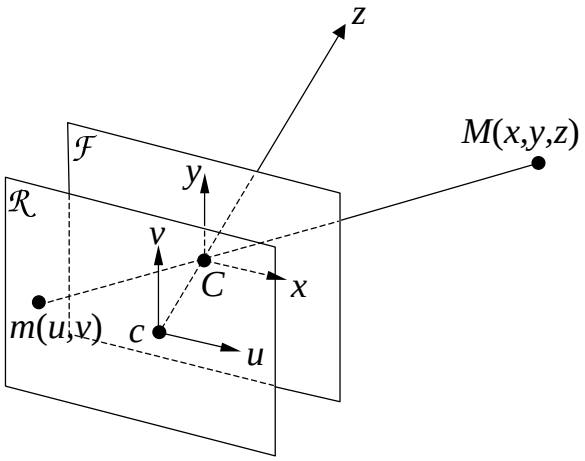


Figure 3.1: The pinhole camera model. The focal plane (x, y) is the plane that goes through camera center C and is parallel to the retinal (image) plane (u, v). The distance f between the two planes is called the focal length. An arbitary point M in 3D space with coordinates (x, y, z) is projected through C onto the retinal plane with image coordinates (u, v).

The pinhole camera model is depicted in Figure 3.1. It consists of a plane R called the *retinal plane* or *image plane*, in which the image is formed, and a point (pinhole) C, called the *optical center*, located at a distance f from the retinal plane. For an arbitrary point M in 3D space, its image m in the retinal plane is formed by the intersection of the line $< C, M >$ with the plane R. f is called the *focal length* of the optical system, and the plane going through the optical center C and parallel to R is called the *focal plane*.

The location of point M in 3D space can be described by its coordinates (x, y, z) relative to some 3D *world coordinate system*. The location of its image m in the image plane can be described by its coordinates (u, v) in some 2D coordinate system on the image plane. The camera measurement process can be viewed as a mapping from 3-D space coordinates (x, y, z) to 2D image coordinates (u, v), and can be described mathematically as

$$\begin{bmatrix} U \\ V \\ S \end{bmatrix} = \mathbf{P} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{3.1}$$

and where vector the $(U, V, S)^T$ is called the *homogeneous coordinates* of a 2D image point and the vector $(x, y, z, 1)^T$ is the homogeneous coordinates of a 3D point. The details and origins of homogeneous coordinates are beyond the scope of this brief tutorial. It suffices to know that they are intermediate quantities to use so that a linear relationship exists between them, as shown in Equation (3.1). The actual 2D image coordinates (u, v) are then computed from (U, V, S) using the relations

$$u = U/S, v = V/S \text{ if } S \neq 0 \tag{3.2}$$

In the above formula, $\mathbf{P}$ is a 3×4 matrix called the *prospective projection matrix* and can be viewed as the product of two matrices. Therefore Equation (3.1) can be written as

$$\begin{bmatrix} U \\ V \\ S \end{bmatrix} = \mathbf{HK} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{3.3}$$

where K is an 4×4 matrix describing the position and orientation of the camera with respect to the world coordinate system, and H is a 3×4 matrix with elements that are functions of camera parameters such as the focal length, center of the image plane, etc. These parameters do not depend on the position and orientation of the camera, and are thus called *intrinsic*; and by contrast, camera position and orientation describe how the local camera coordinate system relates to a global (external) coordinate system, and thus are called the *extrinsic* parameters. A camera can be considered as a system that depends on both the intrinsic and extrinsic parameters.

Camera calibration

Camera *calibration* is the process of estimating the intrinsic and extrinsic parameters of a camera, i. e. estimating matrices $\mathbf{H}$ and $\mathbf{K}$ and their associated parameters. Much previous work has found that intrinsic calibration is best performed on cameras individually since it is not dependent on the global coordinate system. Many calibration methods exist that are appropriate for a single camera, such as those proposed by Tsai [84] and Heikkila [36].

After the intrinsic parameters are determined, how to find the extrinsic parameters of each camera in a way that is globally consistent is the focus of the rest of this chapter. Therefore, it will be assumed that intrinsic calibration has been completed for all cameras.

Mathematically, extrinsic calibration is quite straightforward: according to Equation (3.3), and given that $\mathbf{H}$ is already known, as long as we know enough 3D points (x_i, y_i, z_i) and their corresponding image coordinates (u_i, v_i), we can solve for $\mathbf{K}$ using either linear or non-linear parameter estimation techniques.

However, in practice, the question is how to obtain the set of known 3D coordinates easily. Traditionally, this usually involves building a so-called *calibration object* that has (visually) detectable feature points and physically measuring the locations of the feature points relative to the local object frames in advance. To calibrate a camera, the calibration object is placed in front of the camera and a picture is taken. The 2D coordinates of the features points in the image are then located and correspondences to their 3D coordinates are found. Now that we have a set of 3D to 2D correspondences, regular parameter estimation techniques can proceed and the extrinsic parameters can be determined.

Note that, because the 3D coordinates of feature points are measured relative to the object coordinate frame, the solved camera position/orientation are also relative to the calibration object frame. Therefore, when there are multiple cameras, the calibration object must be placed at a location where all cameras can see it, and the solved camera poses are all relative to the object coordinate frame. Note also that, the larger the number of feature points is and the more spread out the feature points are in the working volume, the better the likelihood is of good calibration due to less need for data extrapolation, which is numerically more unstable.

Challenges of scaling to wide area calibration

The basic calibration techniques described above have parts that involve some manual work, such as building physical calibration objects and/or physically measuring 3D point locations. The amount of manual work may be acceptable when the number of cameras to be calibrated is small; however, it becomes unmanageable as the number of cameras grows. Moreover, when cameras cover a wide working area and all of the cameras need to be calibrated with respect to a *single* coordinate frame, there are additional difficulties, as

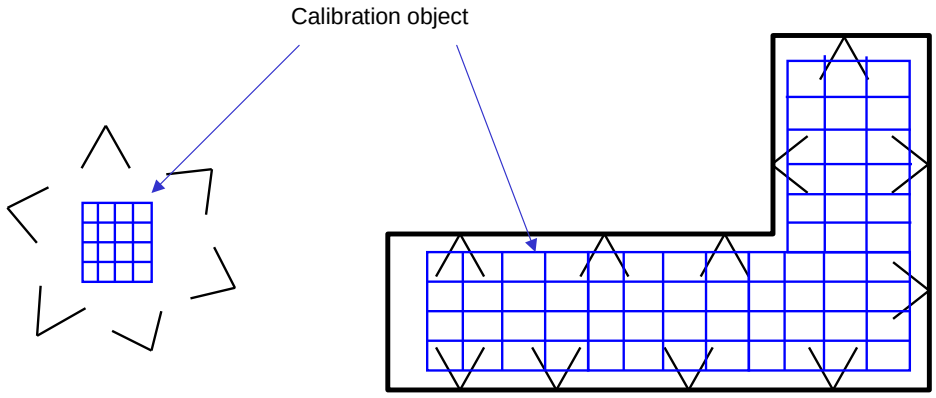


Figure 3.2: Camera settings with different levels of difficulty for calibration. Left: a simple multi-camera setting where it is easy to build and place a calibration object that all cameras can see; Right: An L-shaped camera setup is not so easy to calibrate using the traditional calibration procedure – either a large object has to be built, or adiitional relative positions of multiple smaller calibration objects need to be measured.

described below.

As mentioned before, calibrating multiple cameras with respect to one coordinate frame requires that the calibration object be placed so that it can be seen by all cameras. It is also desirable that the feature points of the object span as much of the working volume as possible. This is feasible for a camera setup as shown on the left in Figure 3.2. But for a complicated wide area setting such as the L-shaped corridor shown on the right in Figure 3.2, the construction of a large calibration object that all cameras in the working volume can see would be costly and cumbersome, and would not be portable when the shape and size of the working volume change. Multiple smaller calibration objects spread across the volume can be used one-by-one to calibrate subsets of the cameras, but to determine the global coordinates of each object, one would have to measure the relative positions of these objects. This can be a non-trivial and cumbersome process. One could also use a smaller calibration object and move it around, but similar to the multiple calibration object approach, one has to measure precisely the object movement to be able to put all cameras into one global coordinate frame. Therefore, more efficient calibration schemes need to be developed to handle wide area systems.

3.2.2 Previous work

There has been a great deal of research in the area of accurate camera calibration. Generally there are two main categories of methods.

One category of methods involves the physical measurement of 3D points. This is usually done with a calibration pattern/object whose feature locations are measured in advance and then imaged by the camera. Features are extracted from the image, and the best fit of intrinsic camera parameters and extrinsic camera pose is obtained. Tsai proposed a widely used model, but other more robust models are used as well [84, 36].

Rander and Kanade have a system of approximately 50 cameras arranged in a dome to observe a room sized space. In order to calibrate these cameras, a large planar calibration object is built and then moved precisely to several vertical locations, in effect creating a virtual calibration object that covers the room [68]. While this works well, it can be quite costly to ensure the precise movement of a calibration object, and it is not easily adaptable when the shape or size of the working volume changes.

The other category of methods usually don't require the physical measurement of 3D point feature locations. Rather, they try to jointly estimate the 3D feature locations as well as camera poses.

Azarbayejani and Pentland propose a method for calibrating the relative position of cameras [2]. An identifiable object is waved in front of a synchronized stereo pair of cameras, and the per-camera image location of the object at each time step is recorded. A standard structure from motion system is used to derive the relative pose of the two cameras. Their focus is not wide area tracking, and synchronous cameras with a common viewing volume are required.

Stein proposes a system of cameras to track vehicles in an outdoor environment [76]. By observing the motion of objects in video sequences from multiple cameras, an approximate camera pose and time offset can be recovered from several asynchronous cameras. Image features are used to refine the calibration estimate. This system requires a flat ground plane in all of the images and solves the homography relating objects on this 2D plane.

Gottschalk and Hughes propose a framework for auto-calibration in wide area spaces [31]. Head mounted sensors observe precisely time-division-multiplexed beacons mounted on

the ceiling of their UNC lab. Data gathered from the sensors can be used to estimate both the moving head location and orientation, and to refine the initially available position estimate of the beacons. Like the method described in this chapter, they also employ the principle of iterative calibration. Welch later proposed a refined estimation method [88]. However, the hardware setup and architecture in that work is quite different from the multi-camera environments that we consider.

The contribution of the work described in the chapter is a wide area calibration method that addresses several previously ignored difficulties. A large number of asynchronous cameras can be calibrated in a single consistent coordinate system. This can be achieved even when some cameras are arranged with non-overlapping working volumes and when no initial estimate of camera pose is available. In addition, the method requires no complex instrumentation, and is easily adaptable to working volumes of variable size and shape.

3.3 Proposed Method

An outline of our method is shown in Figure 3.3. After the separate calibration of intrinsic camera parameters, our method begins by obtaining 2D image correspondences. The pairwise relative pose between cameras can be found using structure from motion. Then, a unification process brings these pairwise relationships into a single global space. The rough estimate of global pose calculated by the preceding steps can be used to initialize the following iterative procedure. A 3D trace of an object moving through space is estimated using an extended Kalman filter (EKF). This trace can be used as a virtual calibration object by correlating it with camera observations. Using traditional camera calibration, a new set of camera pose estimates is obtained. Iteration produces a globally consistent camera calibration.

3.3.1 Initial extrinsic calibration

Pairwise calibration using structure from motion

To obtain a globally consistent extrinsic calibration of cameras, we start by searching for pairwise registration between nearby cameras in our system. To do this we borrow the

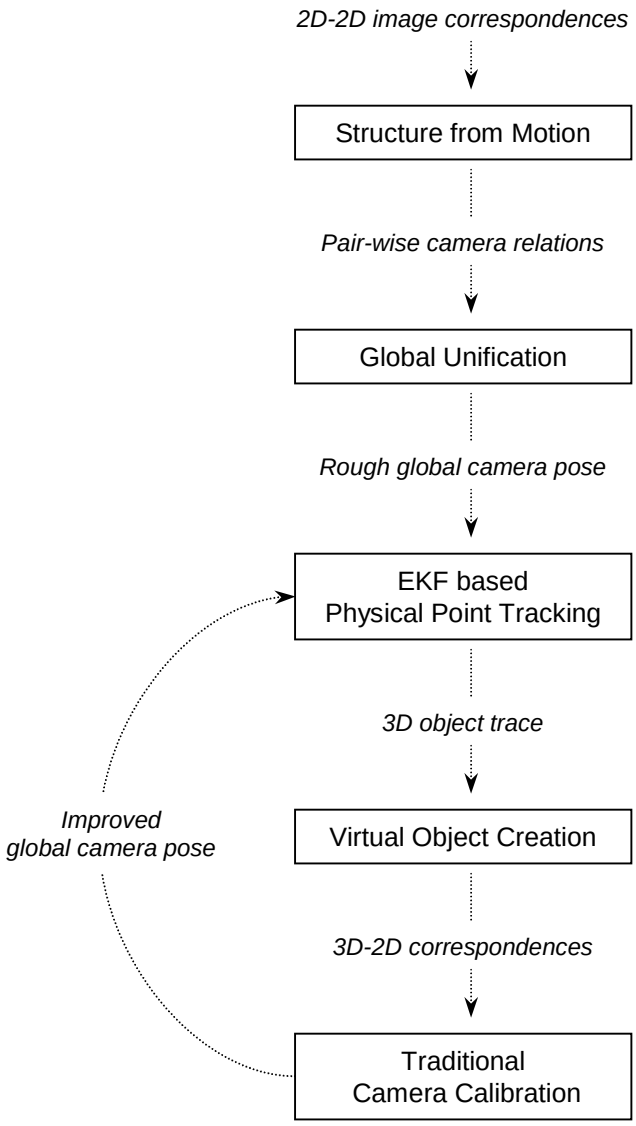


Figure 3.3: The main stages of our calibration method. Boxes represent computational stages. Italicized text shows data flow.

solution to the well-known structure from motion problem. In this problem, a camera is moved from location A to location B, and acquires an image of the same scene from each of the two locations. Due to the camera motion, the same feature point would appear at different 2D locations in the two images. A variety of algorithms exists to estimate the 3D locations of these feature points as well as the relative pose between the two cameras, given their corresponding 2D locations in this image pair [64, 63, 75, 62, 59, 35, 82]. Our calibration problem is similar in that we also have views of the same scene from cameras at different locations, and we are interested in computing the relative camera pose, though we have more than two cameras. However, given corresponding 2D image points in a pair of camera views, a structure from motion algorithm will recover the relative pose of a camera pair. Therefore, we use a publicly available structure from motion implementation from Zhang [93]. An easily identifiable object is moved so that over time it covers the working volume of our system. We use an LED or flashlight in a darkened room. Since each camera sees only a subset of the working area, not all cameras observe the object at any given location. At this stage, however, only pairwise registration is required. The corresponding 2D observations for all relevant pairs of cameras are recorded, and the relative pose is estimated for each pair of cameras with sufficient overlap in their respective fields of view.

It should be noted that since the cameras are not synchronized for simultaneous input, no pair of cameras will actually observe the point at exactly the same location. At this stage we make an approximation that will be refined in a later part of our algorithm. Since the object is known to move continuously, we discretize time into small intervals. We use 36ms, since this is approximately the time required for two NTSC video fields to be processed by our 60Hz cameras. Observations occurring during the same time interval are approximated as co-located both temporally and spatially. Given this approximation and the resulting set of pairwise image correspondences, we can employ structure from motion to obtain a set of pairwise camera registrations.

Global unification

The pairwise camera registration that has been obtained provides only the relative rotation and translation up to an unknown scale factor. For any given pair of cameras that have

sufficiently overlapping fields of view, say cameras A and C in Figure 3.4, structure-from-motion would find the direction of vector $\overrightarrow{AC}$, but not the length of it. If we denote this direction using $\hat{T}_{AC}$ (the "hat" indicates a unit vector), we know that camera C lies on the line in the direction of $\hat{T}_{AC}$ but don't know its exact location, i. e. we do not know the scale factor or length α_{AC}. Similarly, we may also know $\hat{T}_{AB}$, $\hat{T}_{BC}$, $\hat{T}_{CD}$, $\hat{T}_{AD}$, etc, but not the corresponding scale factors. Note also that these vectors are relative to different *local* coordinate frames, e. g. $\hat{T}_{AC}$ is with respect to the coordinate frame of camera A, and $\hat{T}_{CD}$ is with respect to the coordinate frame of camera C. The goal of the this step is not only to determine the locations of all of the cameras, but also to do so with respect to one single coordinate frame.

Some geometric analysis reveals that this goal is not hard to achieve. As an example, if we have pairwise registration for camera pairs (B, C) and (A, C), i. e. we know C should be on the line $\overrightarrow{AC}$ defined by vector $\hat{T}_{AC}$, and on line $\overrightarrow{BC}$ defined by vector $\hat{T}_{BC}$, and we already know the exact locations cameras A and B (i. e. α_{AB} is known), then the locations of camera C on the two lines, (or equivalently, the scale factors α_{AC} and α_{BC}) can be easily found by the intersection of the rays $\overrightarrow{AC}$ and $\overrightarrow{BC}$. In practice, due to imprecision and errors from the structure from motion procedure, the rays may not intersect. We use the point with the minimum distance to both rays as the approximate intersection point.

Once the scale factor α or β is computed, the global location and orientation of camera C can be derived based on the global location and orientation of either camera A or B. We use the following notation. The translation of camera A with respect to B's coordinate system is denoted by T_{BA} . The normalized vector from B to A is $\hat{T}_{BA}$. Similarly the rotation of A relative to B's coordinate system is R_{BA} . Using W to notate the world, or the global coordinate system, and arbitrarily picking camera A as a base, we have:

$$T_{WC} = T_{WA} + \alpha_{AC} \cdot R_{WA} \cdot \hat{T}_{AC} \quad (3.4)$$

$$R_{WC} = R_{WA} \cdot R_{AC} \quad (3.5)$$

Similarly, once the global position and orientation of camera C are found, we can determine the location of camera D by intersecting lines $\hat{\overrightarrow{AD}}$ and $\hat{\overrightarrow{CD}}$. Incrementally applying

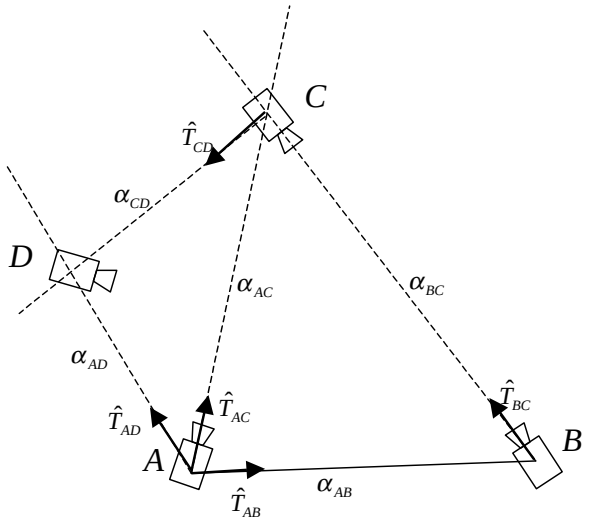


Figure 3.4: Global unification of all cameras. A set of globally calibrated cameras is shown connected by solid lines. The location of camera C can be calculated using the pairwise relationships with A and B, whose global poses are already known, i. e. $\hat{T}_{AB}$ and α_{AB} are known.

the above procedure locates all cameras in a single global coordinate system. Naturally this global coordinate system has an orientation based on the initial camera (A) and an arbitrary scale (α_{AB}). We take a few real world measurements in order to determine the transform to a physically meaningful coordinate system.

It should be noted that the pairwise registration obtained using structure from motion is not of equal quality in all cases. Camera pairs placed in degenerate positions are likely to cause errors, as are solutions determined from only a few image point correspondences. Using the residual error returned by structure from motion, we can rank the quality of pairwise information. This ranking can in turn be used to add cameras to the global set in preferential order, thus improving the estimate of global calibration.

Sources of error

Our initial global calibration process contains a number of approximations and sources of error. The discretization of time results in errors bounded by the velocity of the object and the discretization interval. The incremental method used to add cameras to the global

set is also a source of error. Since the global pose of each additional camera is dependent on the previously estimated global pose of other cameras, small errors made at each step can accumulate. Even though the errors made at each stage may be small, cameras added near the end of a large collection are likely to suffer from a great deal of accumulated error. However, the approximate calibration obtained is a good initial guess for the iterative method that is the core of our algorithm.

3.3.2 Iterative refinement

EKF based physical point tracking

Given a globally calibrated set of cameras, an object can be tracked continuously through the entire working volume. A number of techniques exist for integrating information from diverse sensors into a single estimate of object behavior [14, 42]. We chose to use an extended Kalman filter (EKF) because it is simple and efficient. As described in Chapter 2 and Appendix A.3, our EKF is configured with a constant velocity model and estimates the position and velocity of the tracked point. As an object moves through space, observations from cameras are used to update the EKF estimate. The resulting trace is a continuous estimate of object motion over time. Note that there is no requirement that cameras provide observations at simultaneous moments. The EKF parameters and internal dynamic model provide the temporal constraints normally derived from simultaneous observation. Details of appropriate EKF parameters and models are application dependent, but well known [14, 5, 13].

The initial estimate of camera pose using structure from motion requires that an identifiable object be moved so that the working space is covered. Since the requirements are the same for tracking, the data gathered earlier can be reused. Rather than discretizing observations into time intervals, an EKF processes the observations into a continuous 3D object path.

Virtual object creation

Tracking a 3D object over a wide area provides a method for obtaining a large virtual calibration object. A conventional calibration object has features distributed spatially. Ideally

these features are distributed in such a way that they are easily identifiable and cover the working volume. For large working volumes it is impractical to build a precisely measured physical object for the purposes of calibration. The virtual calibration object defined by the estimated 3D object path has features distributed temporally. Each temporal moment relates to a single position in space.

Even with correctly calibrated cameras, the object trace obtained previously will not be perfect. Inadvertent motion into a region observed by a single camera will leave the system under-constrained. Occlusions can cause complete loss of the object, and sudden acceleration will not fit our EKF model. In order to ensure an accurate virtual calibration object, we discard trace points that are seen by only one camera. In addition, we discard trace points for two seconds after time periods in which no camera observes the point. This provides a chance for the system to settle back into a more reliable state.

Since almost any location on the path can be used, a large number of calibration features can be constructed. If the path of the physical object through space traverses all portions of the desired working volume, then excellent coverage is obtained as well.

Recalibrating camera extrinsics

The virtual calibration object can be used to individually calibrate each camera with respect to the global coordinate system. A given camera reports a sequence of 2D image observations. The time of observation relates this 2D observation to a corresponding 3D feature point in the virtual calibration object. The resulting set of 2D to 3D correspondences can be used to find the external camera pose. As for calibrating the internal characteristics of our camera, we use a standard method [36].

Iterating the above stages improves the estimate of both camera pose and virtual object location. More accurate virtual objects provide better camera calibration, and better camera calibration allows the virtual object path to be determined more accurately. We have obtained convergence for highly over-constrained environments in five iterations, with more general wide area settings requiring up to forty iterations.

3.4 Results

Evaluating the effectiveness of a calibration method is not trivial. It is sometimes difficult to obtain ground truth data for the camera in question. In addition, since we have proposed a calibration method for use in wide area environments, general comparisons with existing techniques are impossible.

We first discuss an appropriate metric with which to evaluate our results. Next we consider calibration in a restricted setting, in which comparison with existing techniques is possible. Finally, we show that our proposed method functions as expected on a more general wide area calibration task.

3.4.1 Evaluation method

Camera calibration is often defined in terms of projection error. In a traditional calibration task, known 3D locations are projected onto the camera image plane. The distance from the projected point to the observed image location is known as the projection error. Given a set of correspondences, the best camera calibration is the one that minimizes this error.

Another formulation provides only 2D image correspondences between multiple cameras. In this case, in addition to camera parameters, the actual 3D point location is unknown. The best fit of these variables is often defined as a minimization of projection error. If cameras are poorly calibrated in relation to one another, then one expects projection error to be quite high.

Since the cameras used in our application are not synchronized, we slightly modify this method. An object is tracked using an EKF as previously described. At the time of each camera observation, the EKF provides a predicted object location. The predicted location can be projected onto the camera image plane. The distance between the predicted location and the observed location is recorded as projection error. Note that this error may be caused in part due to an inadequate EKF system dynamic model. In our case we use a constant velocity model; thus, any acceleration applied to the object will appear as projection error. However, this error will be present only during acceleration. Extended, consistent bias in the projection error can be attributed to poor camera calibration.

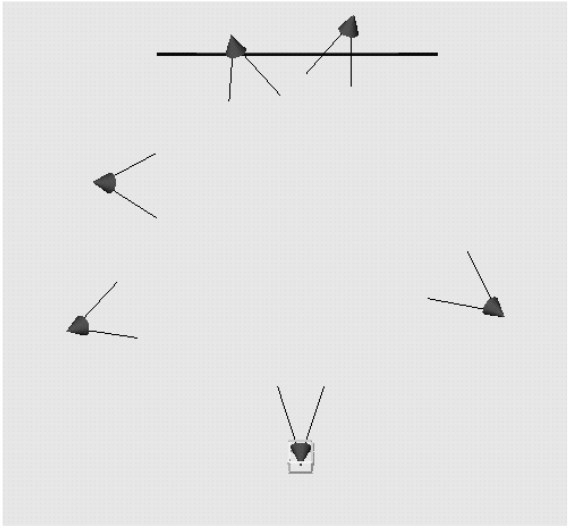


Figure 3.5: Restricted setting in which all cameras can see a common area. A configuration such as this allows comparison with existing calibration techniques.

3.4.2 Comparison with existing techniques

In order to compare our proposed calibration to existing methods, we have constructed an example in a more restricted setting. By arranging multiple cameras so that all may observe a single region of space, traditional target based calibration can be performed (Figure 3.5).

We calibrate the camera system using two methods, the one proposed in this paper, and one that uses a physical target. In order to isolate potential noise, the target based calibration uses the same LED feature as was used to build a virtual calibration object. This LED feature is now placed at the end of a Faro digitizing arm [27] that returns the position of the tip of the arm with sub-millimeter accuracy. By arranging for simultaneous triggering of the digitizing arm and camera, we can obtain correspondences between the global 3D location of the feature and the observed 2D image location. While only a few correspondences are theoretically needed, we use approximately 50 to obtain robust external calibration. By repeating this process for each camera a complete set of calibrations in a single coordinate space is obtained.

To evaluate our method fairly, we gathered a new object trace unrelated to any previous traces used during calibration. It is important not to reuse previous traces during evaluation,

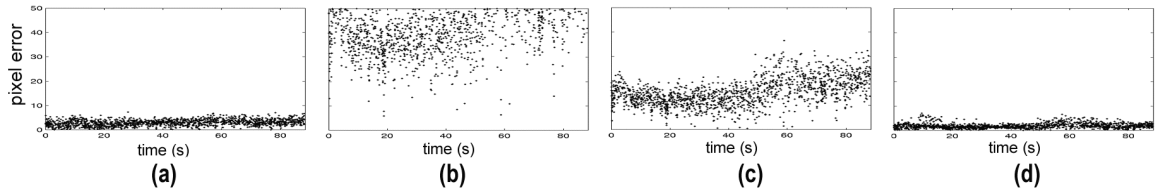
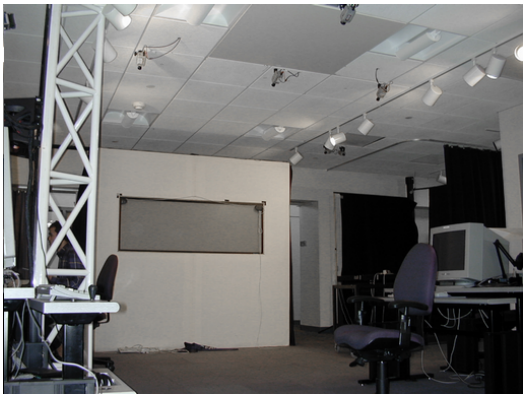


Figure 3.6: Comparison of projection error of a point trace using (a) traditional target based calibration, (b) structure from motion only, (c) one iteration of virtual object calibration, (d) five iterations of virtual object calibration. Note that a virtual calibration object performs as well as traditional calibration.

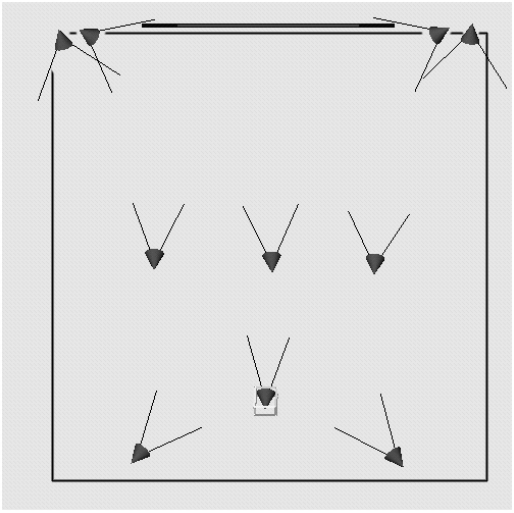
since we want to ensure against overfit solutions that match the input data set, but do not actually provide a calibrated system of cameras. Figure 3.6 contains a set of graphs showing projection error in the tracking process for a particular camera. (Data from other cameras in the system is similar.) Using the calibration obtained with a traditional physical target results in trace (a). The average pixel error in a mean squared sense over all cameras is relatively low, only 4.4 pixels. The rough global calibration obtained using structure from motion results in trace (b). Note that the mean error has greatly increased to 33.8 pixels. After building a virtual calibration object and recalibrating the cameras, we obtain trace (c). Using a virtual calibration object has reduced the mean error to 14.4 pixels. After five iterations of building virtual calibration objects and recalibrating the cameras, trace (d) results. The mean error has been further reduced to 3.9 pixels, approximately equal to the known reliable calibration obtained with a target. Some error remains, but as mentioned earlier this may be due to object acceleration or image feature extraction noise.

3.4.3 Example in a wide area setting

The wide area calibration technique described in this chapter can be generalized to a wide variety of applications, sensors, and environments. An understanding of a representative side area tracking setup in our lab may prove illustrative. This setup can support a variety of tracking applications, such as 3D head tracking for the Interactive Mural (see Section 2.3.1) and simultaneous body and face motion capture (see Section 2.3.3). The tracking system



(a)



(b)

Figure 3.7: (a) Ceiling mounted cameras are used to track users around a wide area environment in front of the large display. (b) Cameras are arranged so that observation of the entire space is possible, although no single camera observes the entire working volume.

has ten ceiling-mounted cameras oriented to observe a 4.0 x 4.5 meter area. Coverage extends from approximately a half meter to 2 meters from the floor. The wide-angle cameras in the corners cover the volume. In addition, since our application requires higher tracking resolution right in front of the display, a few more narrowly focused cameras are installed to increase the resolution in that area. Individual cameras observe only a portion of the volume. In aggregate, however, they cover the space. Figure 3.7 shows a photograph of the tracking space, and a plan view of camera placement. Note that in order to ensure correct estimates of observed object position, it is required that at least two cameras observe any given region in space. However, there is no single point that is observed by all cameras.

In the wide area setting described above, we can verify that the iteration process converges as it does for the more restricted setting described in Section 3.4.2. Figure 3.7 shows final camera placement, while Figure 3.8 shows camera locations after calibration using only structure from motion. Note that these locations are approximately correct. However, refinement is required. For example, the three cameras in the middle are expected to be collinear. Figure 3.8 also shows a plan view of the temporal path used to build a virtual calibration object. Note that the path traverses the field of view of all cameras, covering the

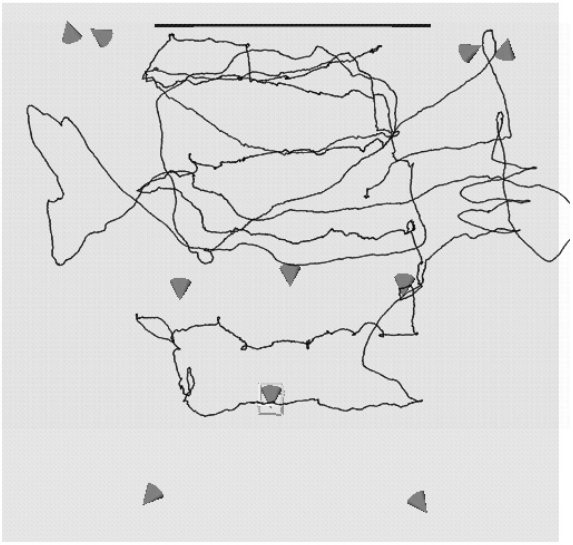


Figure 3.8: Top view of estimated camera positions after applying structure from motion only. The object path shown defines a virtual calibration object that can be used to improve camera calibration.

working volume better than a single physical calibration object.

As described previously, projection error is expected to decrease as quality of calibration improves. A new trace unrelated to traces used for calibration was captured. Figure 3.9a shows the projection error of a single camera after the coarse global calibration of all cameras using structure from motion. As before, the mean error across all cameras is quite large, about 16.3 pixels. Holes in the graph indicate time periods in which this particular camera did not observe the object, so no error measure is available. (Of course metrics are available for other cameras that do see the point during this time period). Figure 3.9b shows the improvement after three iterations. Forty iterations of building a virtual calibration object results in a low projection error, 1.3 pixels, shown in Figure 3.9c. Projection error lower than that obtained in the previous example can likely be attributed to improvements in intrinsic camera calibration. A plot of pixel error vs. iterations can be seen in Figure 3.10.

An intuitive explanation for the convergence of this algorithm is as follows. As shown in Figure 3.11, there is a geometric relation between the 3D position of a point, the camera

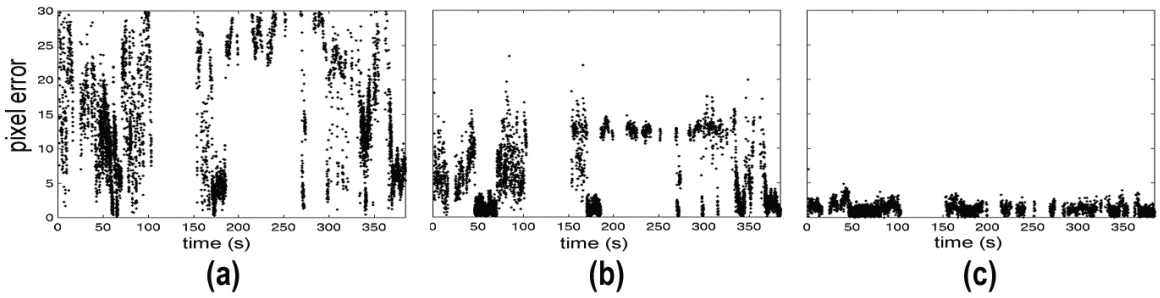


Figure 3.9: Projection error in a wide area calibration task. (a) Error after applying structure from motion. (b) Error after three iterations of calibration using a virtual calibration object. (c) Significantly reduced error after forty iterations using a virtual calibration object.

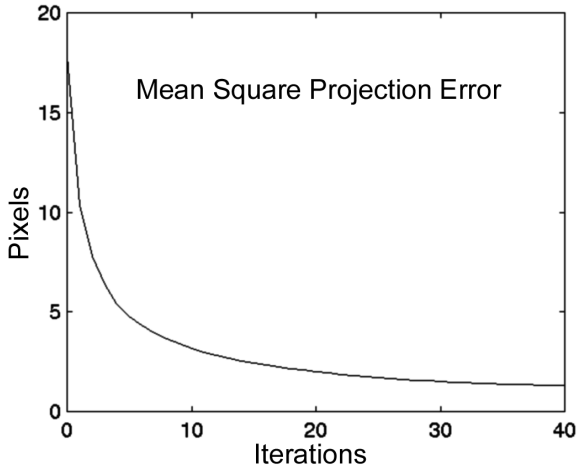


Figure 3.10: Projection error is greatly reduced after application of the iterative calibration process.

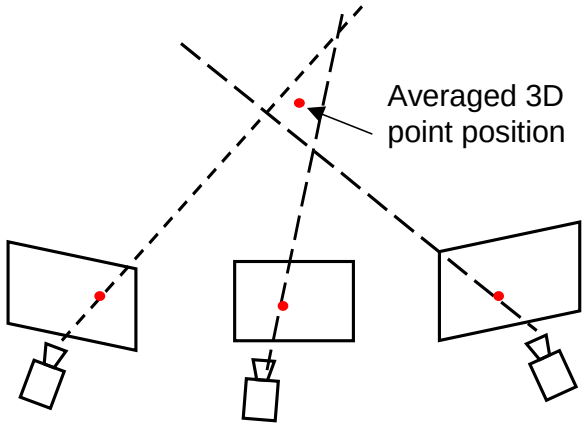


Figure 3.11: Intuition behind convergence. 3D point position is an average quantity based on multiple camera poses (and the 2D projections on the image plane). The variance of 3D position is less than the variance (error) of individual camera poses due to averaging. Less erroneous 3D positions in turn improve camera pose estimation in the recalibration. And thus convergence is conceivable.

pose, and the 2D projection of the point on the camera image plane. The 3D position of a point is essentially determined by the intersection of back-projected rays from each camera. When there are multiple cameras as in our system, the Kalman filter that assimilates measurements from all cameras acts as a low-pass filter, and the 3D position it generates is essentially an "average" based on data from all cameras. Since the variance of an average quantity is smaller than the variance of each individual quantity within the average, even though the initial camera poses are coarse and have larger error (variance), the 3D positions based on all of the coarse camera poses have less variance due to averaging. In turn, the improved 3D positions are used to recalibrate the cameras, which generates a better estimate of the camera poses. Using these improved camera poses, we can run the Kalman filter to track 3D position again, which again reduces variance. As the process iterates, it is not surprising that it converges. From the experiments we have run, we have observed emperically that, the more overlap there is among cameras' fields of view, the more accurate the initial camera poses (obtained from structure-from-motion) are, and the faster the iterative refinement precess converges. The number of iterations for convergence shown in Figure 3.10 is for one of the slower cases.

3.5 Conclusions

Calibration of cameras in a wide area environment introduces new challenges to the calibration process. Since individual cameras observe only a small fraction of the whole environment, determination of reliable global relationships is difficult. Typical previous approaches such as building calibration objects that span the observation space do not scale well to wide area environments.

We have introduced a method suitable for calibration of cameras under these difficult conditions. Intrinsic camera properties are calibrated using existing methods. Next pairwise extrinsic relationships are determined using standard structure from motion techniques. These pairwise relationships allow us to derive an approximate global calibration involving all cameras. The initial estimate of global relationships is not precise. However, it can be used to initialize an iterative calibration technique. Tracking a known physical object as it moves through the environment allows a virtual calibration object to be built. This virtual calibration object can be used as if it were a giant physical calibration object to improve the global camera calibration. Iterating the above process leads to our final camera alignment.

Chapter 4

Determining Optimal Camera Configurations

4.1 Introduction

In designing a vision-based tracking system it is important to define a metric to measure the ”quality” of a given camera configuration. Such a quality measure has several applications. By using the metric in an optimization process we can automate the camera placement process and do better than a human designer, especially as the tracking environment gets more complex and the number of cameras increases. Also, there are classes of applications where camera configurations change dynamically and some metric is needed to guide the automatic choice of best configuration. For example, in a multi-target tracking system with multiple pan-tilt cameras, it may be desirable to dynamically focus different subsets of cameras on each target. Other applications might require dynamic configuration due to bandwidth or processor power limitations. For instance, in a system with hundreds of cameras, only a subset of the cameras can be active. In these situations it is crucial to have a quality metric so that the camera configuration that enables the best tracking performance can be found.

In a motion capture system, multiple cameras observe a target moving around in a working volume. Features on the target are identified in each image. Triangulation or disparity

can be used to compute each feature's 3D position. In such a system, performance degradation can come from two major sources: (1) low resolution which results in poor feature identification; and (2) occlusion which results in failure to see the feature. Occlusion may be due to either the target itself or other objects in the scene. When not enough cameras see a feature, it is difficult or impossible to calculate its 3D position. In fact, the primary reason commercial motion capture systems consist of many cameras is to reduce occlusion, rather than to increase resolution or coverage. In order to achieve accurate and robust tracking, both occlusion and resolution must be considered. A quality metric for placing cameras should reflect the impact of both factors.

In this chapter we propose a quality metric that accounts for both the resolution and occlusion characteristics of a camera configuration. Its target application is the automatic placement and control of cameras for motion capture systems. It can be used both to design static camera arrangements, and dynamically focus subsets of cameras on different targets in a multi-target tracking system. The metric computes the uncertainty or error in the tracking system's ability to estimate the 3D positions of features. This uncertainty is caused by limited 2D image resolution, as well as occlusion due to the environment and/or the moving target itself. The error due to image resolution is computed by projecting the 2D image error of each camera into 3D space and measuring the size of the resulting 3D error volume. The uncertainty due to target occlusion is estimated by sampling the space of possible occluders and computing the probability that points are occluded.

The quality metric for multi-camera configurations includes a probabilistic occlusion model. This metric allows cameras to be placed more robustly than previous resolution-only metrics. It models the target self-occlusion behavior that can be commonly found in feature-based motion capture systems. In addition, the use of sampling in the metric computation allows for easy adaptation to tracking scenarios with disparate occlusion characteristics.

The rest of the chapter is organized as follows. Section 4.2 describes previous work related to camera placement. Section 4.3 describes the metric we propose, focusing specifically on how dynamic occlusion is modeled. Section 4.4 describes the experimental setup and results for the verification of our probabilistic occlusion model. Section 4.5 shows simulation results that illustrate the impact of resolution and occlusion on optimal camera

placement. These examples illustrate the usefulness of the metric, and provide insight on the location of ”good places” to put cameras for accurate and robust tracking. (A concise version of this chapter can be found in [16]).

4.2 Related work

The camera placement problem can be regarded as an extension to the well-known art-gallery problem [65]. Both problems have the goal of covering a space using a minimum number of cameras, and in both the solutions are greatly affected by the visibility relationship between the sensor and the target space. However, there are significant differences between these two problems. The art-gallery problem focuses on finding the theoretical lower-bounds on the number of guards for spaces of various (possibly very complex) shapes. Both the target space and the locations of guards are restricted to 2D. Additionally the visibility model is very simple. It assumes that the guard has a 360-degree field of view (FOV) and that there is no resolution degradation with viewing distance. The camera placement problem that we consider is a practical problem. The camera has a more complex model which includes 3D projection, limited FOV, and limited image resolution. Although the space to cover in practical camera placement is usually geometrically simple, there are usually constraints on allowable camera placement, such as ceilings and walls. In our work, the goal is not necessarily to find the absolute global optimum, but rather to enable the evaluation and comparison of a subset of potential solutions.

There is some previous work in automatic sensor planning in the area of robotic vision [78, 79, 83, 92], motion planning [50], and image-based modeling [29]. To a large degree, the previous work shares our view of camera placement as an optimization problem, and that an important step toward automatic solutions is the construction of a quality metric to evaluate various camera configurations. However, the problem domain and goals of the previous work are quite different from ours. Their target is usually a static object whose geometry is assumed to be known a priori, and the task is to find a viewpoint or a minimum number of viewpoints that exposes the features of interest on the target as much as possible. There is usually only one camera in the system and the quality metric includes the target geometry explicitly. The camera placement problem for motion capture differs

because the target is moving and its geometry and motion are not known a priori. In addition, there are multiple cameras working together in the system and the system performance is affected by their relative pose.

A few researchers have proposed uncertainty analysis for placing multiple cameras. Olague and Mohr [61] approximated the projective transformation of a camera using Taylor expansion, and used a scalar function of the covariance matrix as the uncertainty measure. Wu et al. [91] proposed a computational technique to estimate the 3D uncertainty volume by fitting an ellipsoid to the intersection of projected error pyramids. Both work consider limited image resolution as the only cause of 3D uncertainty. However, occlusion is frequently present in feature-based motion tracking systems and is sometimes the dominant source of error. The quality metric presented in this chapter estimates the 3D uncertainty caused by both occlusion and image resolution. This metric is able to model the dynamic and probabilistic characteristics of a moving target, as well as the mutual resolution compensation among multiple cameras.

4.3 Construction of the quality metric

In this section we define the ideal quality metric for a multi-camera configuration with respect to a working volume. Since it is based on the notion of *3D position uncertainty*, we first describe the possible causes of position uncertainty and how it is related to the layout of all of the cameras in the system. We then proceed to describe a general formulation to construct the metric, and using a top-down order, describe the various pieces needed in the metric. We describe specifically how to model the effect of limited camera image resolution and target occlusion on the position uncertainty of a point. We then extend the formulation to a volume and put all of pieces together to give the final formulation of the quality metric.

4.3.1 Causes of 3D position uncertainty

In order to determine the 3D position of a feature point on a target, one needs to know two pieces of information: the complete camera parameters and the 2D feature locations

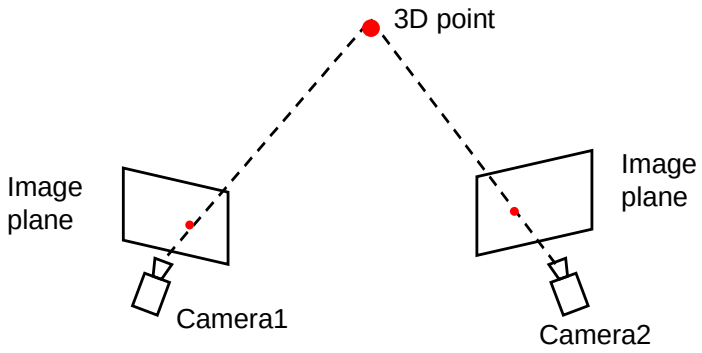


Figure 4.1: Determination of 3D position.

on the image plane. Camera parameters include intrinsic parameters and extrinsic parameters. These parameters together specify where a camera is located, how its image plane is oriented, and what the 2D location on the image plane should be given its 3D position. The 2D position of a feature is usually determined in two steps: classification and location calculation. In the classification step, pixels on the image are classified as either generated by the feature of interest, or not. For example, if the feature is a bright point, pixels that are brighter than a certain threshold are identified; if the feature is an edge, then those pixels with large neighboring gradient are identified.

As shown in Figure 4.1, given the knowledge of camera parameters and the 2D location of a feature seen by at least two cameras, the 3D position of the feature can be determined by the intersection of rays originating from the camera center and going through the 2D feature on the image plane. Of course, in practice, due to various errors and noises, these rays may not intersect at one point, and the 3D position is usually defined as the point that has the shortest distance to all back-projected rays.

Some practical systems have more sophisticated algorithms to determine target 3D position than the simple triangulation we just described above. For example, if multiple features on a target need to be tracked, features on the same rigid parts of the target have fixed distances between them and this information can be used to further constrain the position estimation; or some temporal model such as how fast the target can be moving can be used to correlate features in different frames. However, the determination of 3D feature point position still relies on the basic information we described above, which includes the camera

parameters and 2D image measurements.

Based on the above description, we can see that there are several factors that affect how accurately we can determine the 3D position of a feature point. The causes of *3D position uncertainty* include:

Camera parameters These include both the intrinsic parameters such as lens parameters and extrinsic parameters such as position and orientation. The process of determining these parameters is called camera calibration, and inacurrate calibration would cause in accurate 3D positions.

Image measurements Poor image measurements can result from a poor feature identification process. Moreover, even with a good identification algorithm, image measurements are subject to limited camera pixel resolution, and where the camera is placed relative to the target. For example, a target that is far away from a camera has worse resolution (pixel per unit object distance) than a target that is close the the camera.

Occlusion Occlusion is the phenomenon when the feature is blocked from a camera by some object(s) called *occluder(s)*. An occluder can be non-varying with time (i. e. *static*) such as a pillar or wall in the environment, or *dynamic* (i. e. time-varying) as the target moves. We classify dynamic occlusion into two types: self occlusion and inter-object occlusion. Self-occlusion refers to occlusion where the feature is occluded by the surface that it is attached to; inter-object occlusion refers to occlusion where the feature is occluded by other moving parts of a same target or another target. Figure 4.2 illustrates the different types of occlusion. When occlusion occurs and not enough cameras see a feature, it is difficult or impossible to calculate its 3D position. It is obvious that the occlusion behavior of a multi-camera tracking system is heavily affected by the relative position and orientation of all cameras.

Therefore, in order for a tracking system to have good accuracy in 3D position determination, it is important to have accurate camera parameter information, accurate image measurement information, and minimal target occlusion. The determination of camera parameters is the goal of camera calibration, which was discussed in Chapter 3. Accurate 2D image measurement depends on having a good feature detection algorithm, and also on the

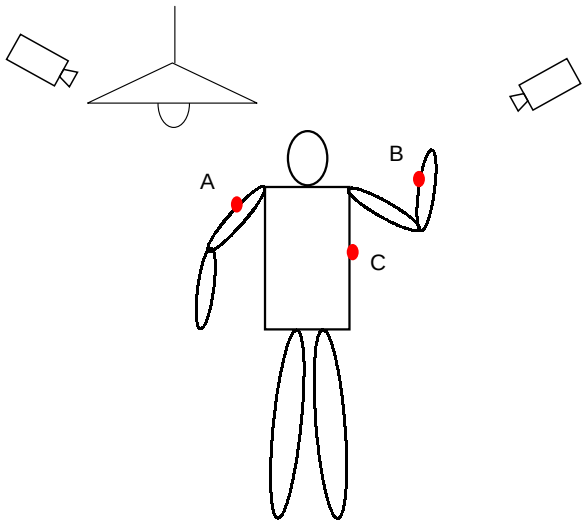


Figure 4.2: Various types of occlusion based on the occluder behavior. Static occlusion shown point A since it is blocked by a static object in the scene; dynamic self-occlusion shown for point B since it is blocked the surface it is atttached to; dynamic inter-object occlusion for point C since it is blocked by *other* parts of the target. Note when we talk about the possible occlusion that can happen to a point over a period of time, the points are *spatial* points in the working volume, not attached the target, and the occlusion types they can have depend on the configuration of the target. For example, point A can also have dynamic self-occlusion if the person's arm or other parts move to a different pose and block it from a camera. But the lamp will always be in between the left camera and point A.

location of the cameras relative to the target. The amount of occlusion is also significantly affected by where the cameras are located relative to the target.

4.3.2 Quality metric of a camera configuration: general definition

We define a *camera configuration* $\mathcal{C}$ as the collection of camera parameters of all cameras. That is, $\mathcal{C} = (\mathbf{C}_1, \mathbf{C}_2, \ldots, \mathbf{C}_N)$, where $\mathbf{C}_i$ is a vector containing the (intrinsic and extrinsic) parameters of the ith camera. (We will also use the word "layout" or "placement" interchangeably with "configuration" in this chapter, but keep in mind that this also includes camera intrinsic parameters such as field of view.). The quality metric of a camera configuration is defined as a scalar value q that measures the 3D position uncertainty, as a function of a camera configuration $\mathcal{C}$ with respect to a working volume V in which the target is moving, i. e.

$$q = f_V(\mathcal{C}) \tag{4.1}$$

Equation (4.1) basically says that position uncertainty is a function of camera layout. But what does this function look like, or in other words, how does camera layout affect position uncertainty? Based on Section 4.3.1 we know that the configuration of cameras not only affects the image resolution of the target, but also has a significant impact on the occlusion behavior of the target. Both image resolution and target occlusion affect how accurately and robustly 3D position is determined in a motion tracking system. We shall describe how to model their effects on the quality metric in the coming sections.

4.3.3 Uncertainty of a point with resolution only

As is well known in the vision community, one major source of error in determining 3D position is limited 2D image resolution. Most vision-based tracking systems perform some sort of triangulation on rays from two or more cameras. Some target feature is detected on the image and the ray defined by its 2D location and the camera center is back-projected into 3D space. The intersection point of rays from multiple cameras defined by the same feature is the 3D location of the feature, as described in Section 4.3.1. However, one can

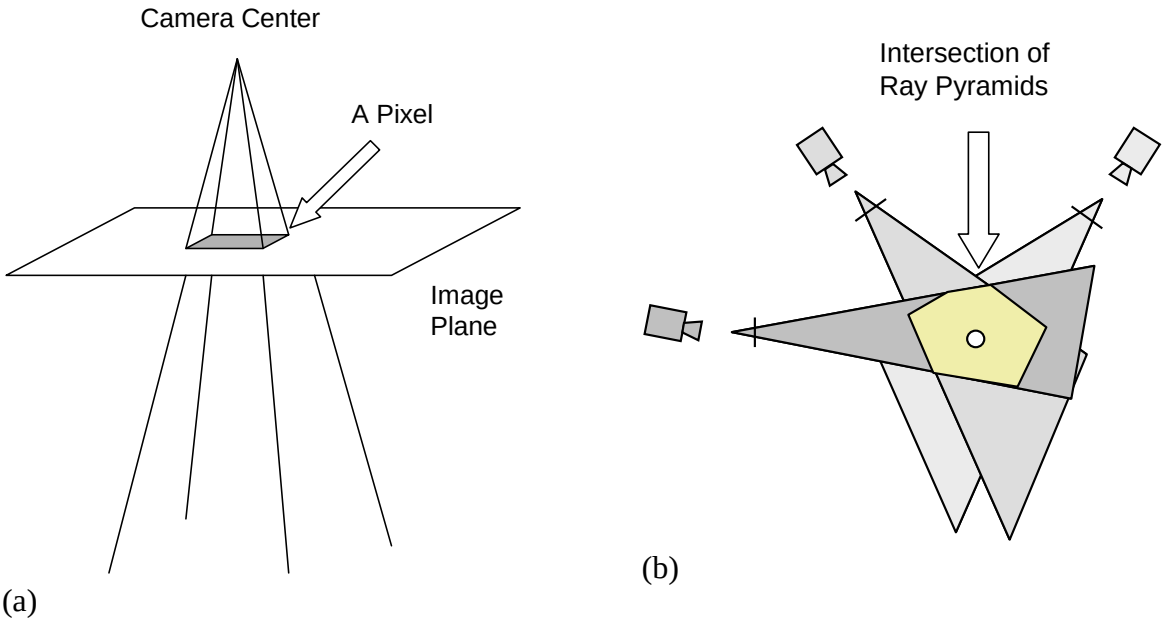


Figure 4.3: (a) The 3D uncertainty volume of a single camera due to limited 2D resolution on its image plane can be modeled as a pyramid of rays. (b) The 3D uncertainty volume of multiple cameras is the intersection of ray pyramids of all cameras.

only determine the 2D location of a feature on an image to a certain precision, as limited by the feature detection process and the image resolution. Thus each 2D observation from a camera gives rise to a cone of rays with some probability density rather than a single ray. This cone can be approximated as a pyramid of rays. As shown in Figure 4.3, the intersection of ray pyramids from multiple cameras form a 3D volume rather than a point. The actual target point has a high probability of being located anywhere in the error volume. The bigger the error volume, the higher the uncertainty in the point's actual position.

It should be noted that the size of the uncertainty volume is not isotropic: it varies with direction. In order to estimate the size of the volume, we measure its dimension in a set of sampled directions and then aggregate the error in all directions. To compute the 3D uncertainty for a given direction, we first compute the 3D error in that direction for each camera. When there are multiple cameras, the camera having better resolution will cut the error volume and make it smaller. Thus, the combined uncertainty of all the cameras is the minimum of all.

Conceptually, without being very specific or precise, $U_{res}(P)$, the 3D uncertainty at

point P due to limited image resolution, can be expressed by the following:

$$U_{\text{res}}(P, \emptyset) = \underset{\text{all } \mathbf{d}}{\text{norm}_k} \left(\min_{\text{all } \mathbf{C}} (L(P, \mathbf{d}, \mathbf{C})) \right) \tag{4.2}$$

where $L(P, \mathbf{d}, \mathbf{C})$ is the length in direction $\mathbf{d}$ of the 3D ray pyramid of camera $\mathbf{C}$ at point P, as shown in Figure 4.4. The function $\text{norm}_k()$ aggregates the uncertainty over all directions, and the choice of k is user-dependent. Usually the function $\text{norm}_k()$ can be $\max()$ (i. e. $k = \infty$) or mean-square-error (i. e. $k = 2$) depending on whether the user wants to evaluate the worst-case scenario or an average scenario. The empty set symbol $\emptyset$ denotes that this U_{res} is computed with no (dynamic) occluders in the scene, i. e. this measure takes into account uncertainty due to limited image resolution only. Static occlusion is directly taken into account by noting in the above formulation that only unoccluded cameras contribute to a reduction in 3D uncertainty at point P. This is a simplified case of the model described in the next subsection. In the next subsection we will consider the situation where there is dynamic occlusion, i. e. the set of dynamic occluders is not empty.

4.3.4 Uncertainty of a point with dynamic occlusion

Dynamic occlusion occurs when a target point is not visible from a camera due to occlusion by the moving target itself in the scene. The challenge to computing the error caused by this type of occlusion is that we do not know a priori where the target or occluder will be at any time, and the exact shape of the target, either. Without the knowledge of target shape and location it is impossible to arrange cameras such that the target is guaranteed to be visible.

We investigate this issue by assuming, for the moment, that we do know where the occluder is, as shown in Figure 4.5. As can be seen, point P is only visible from a subset of the cameras due to this occluder. As a result, the 3D uncertainty volume is formed by the intersection of the ray pyramids of only those unoccluded cameras, i. e.

$$U_{\text{res}}(P, \text{oc}_1) = \underset{\text{all } \mathbf{d}}{\text{norm}_k} \left(\min_{\text{all } \mathbf{C}} (L(P, \mathbf{d}, \mathbf{C}) v(P, \mathbf{C}, \text{oc}_1)) \right) \tag{4.3}$$

where visibility factor $v = \infty$ if P is occluded from camera C given occluder oc_1 and

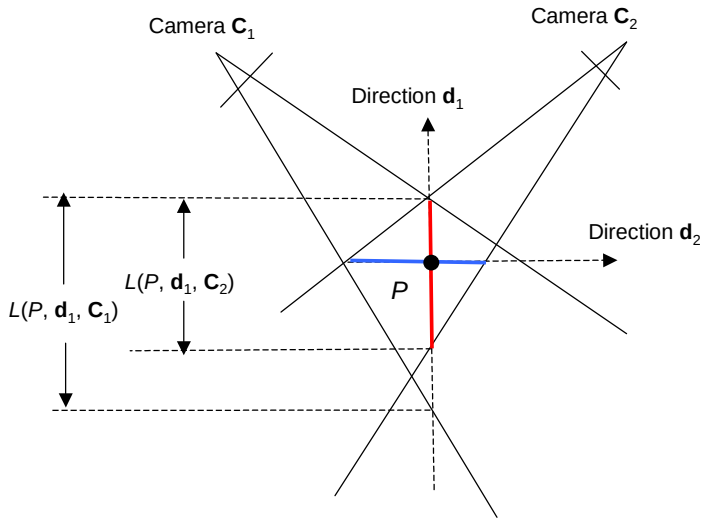


Figure 4.4: Estimating the size of the uncertainty volume of a point. $L(P, \mathbf{d}_i, \mathbf{C}_j)$ is the length in direction $\mathbf{d}_i$ of the 3D ray pyramid of camera $\mathbf{C}_j$ at point P. For each direction, the best (minimum) L out of all cameras is picked as the estimated size of the uncertainty volume for that direction, because multiple cameras compensate for each other's uncertainties in determining point P's position, as shown by the red line for direction $\mathbf{d}_1$ and the blue line for direction $\mathbf{d}_2$. The overall uncertainty for the point considering all directions is defined to be either the worst (maximum) or some average of the uncertainty of all directions.

$v = 1$ if P is visible. Therefore, if a camera cannot see P, its ray pyramid size L will have a huge weight v and will *not* be counted in the $\min(\bullet)$ evaluation that combines the effect of all cameras. A different occluder oc_2 (with different position, orientation or shape) would lead to a different uncertainty value $U_{\mathrm{res}}(P, \mathrm{oc}_2)$, and so on. If we wish to evaluate the uncertainty for a whole tracking session and we know precisely all the positions and orientations of the occluder in the tracking session, we could evaluate the visibility factor for each occluder and aggregate the uncertainty values for all occluders to get an estimate of the overall position uncertainty at point P for a given camera configuration, i. e.

$$U_{\mathrm{res}}(P) = \sum_{\mathrm{oc}_i \in \mathcal{O}}^{*} U_{\mathrm{res}}(P, \mathrm{oc}_i) \tag{4.4}$$

where $\mathcal{O}$ denotes the set of occluders over all positions and orientations, and $\sum^{*}$ denotes some general aggregation function that maps an ensemble of values $\{U_{\mathrm{res}}(P, \mathrm{oc}_1), \ldots, U_{\mathrm{res}}(P, \mathrm{oc}_N)\}$ onto a single number. (This aggregation could be a simple summation, average, weighted average, or more complex mapping depending on the application and user specification. We will discuss how to choose this general function later.) However, this method will not be very useful for designing a real tracking system. In reality, the path that a target takes could vary greatly and camera configurations optimized for one specific path may be poor for other paths. So the next question is, how can we find a camera configuration that avoids occlusion for all possible paths?

Instead of simulating a precise path when calculating occlusion, we can take a probabilistic approach. Though we may not know exactly where the "occluders" will be, we may have some idea as to how likely it is that they will be at certain positions and orientations, i. e. if we have the probability distribution for the occluder, we can generate possible occluders by drawing samples from the distribution. For each possible occluder, we calculate how many cameras are obstructed from observing the target point P. The results across occluders that are sampled from the distribution are aggregated into a single estimate of occlusion characteristics. The more often occlusion occurs and the more cameras that get occluded, the greater the 3D uncertainty. Sampling the space of all possible occluders gives us an occlusion metric for a particular camera configuration. In other words, we can still use Equation (4.4), but instead of having the occluder positions and orientations drawn

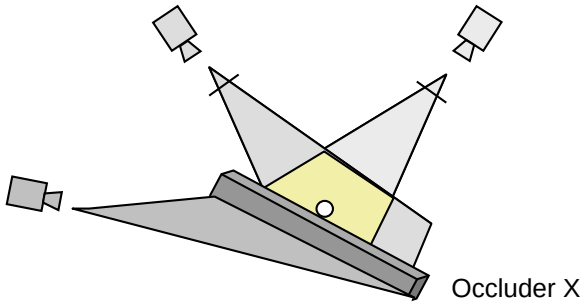


Figure 4.5: Effect of an occluder to the 3D uncertainty volume. The point can be seen by cameras B and C but not camera A. Therefore A does not contribute to reducing the uncertainty volume of the point. In general, the 3D uncertainty volume at point P of multiple cameras when there is an occluder present is the intersection of ray pyramids of the subset of cameras that are *visible* or *unoccluded* from the point.

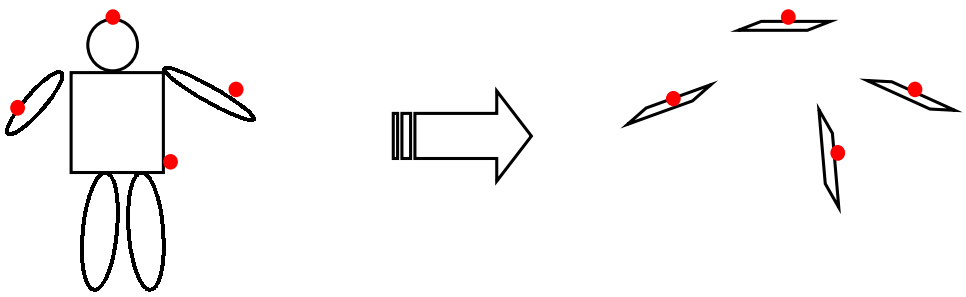


Figure 4.6: Approximation of an occluder: dynamic self occlusion is modeled as a plane attached to the 3D point of interest.

from an a priori known path, they are drawn from some known probability distribution.

The probabilistic description addresses the problem of having no precise knowledge of the occluder path, but another challenge that we face is that we do not know the precise shape of the occluder. We address this problem by looking at the various occlusion phenomena in motion capture, determining what occurs most commonly and building a model that describes the occlusion phenomena sufficiently but is still computationally manageable. As described in Section 4.3.1, we see that there are two types of dynamic occlusion, i. e. occlusion due to the moving target, in a typical point-feature based motion capture system. One is self-occlusion, where the feature point is occluded by the very surface it

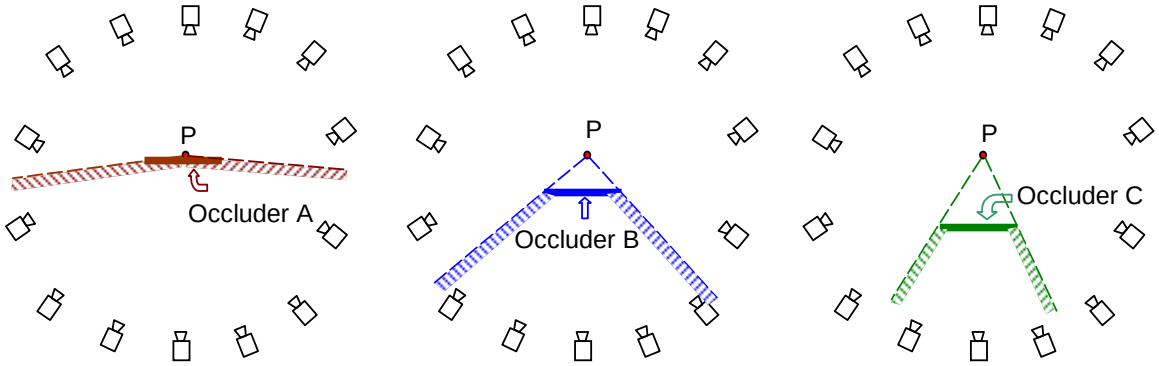


Figure 4.7: Worst-case occluder. The occluded regions (shown shaded) of three occluders that are located at various distances from a point P. The shape of the occluded region is dependent on both the size and orientation of an occluder. But for the same occluder, the closer it is to the target point, the more cameras that will be occluded. A conservative estimate of the occluded region can be made by assuming that the occluder is very close to point P. In this case an entire hemisphere of camera locations will not be able to see the point. Notice the durality of this figures vs. Figure 4.5. Here the shaded region is the *occluded* region where cameras cannot see the point, whereas in Figure 4.5 the shaded region is the *visible* where the cameras can see.

is attached to; another type is called inter-object occlusion, where the feature is occluded by other parts of the target or other target(s). Our assumption is that the major source of occlusion for a point is the surface to which it attaches. We use a plane to approximate the surface, as shown in Figure 4.6. We believe that this is a reasonable model because, first, in current motion capture systems, points are attached to a surface that can be viewed as a plane locally, and self-occlusion does happen all the time; second, since a plane always occludes a whole hemisphere of cameras, effects of other possible (inter-object) occluders in that hemisphere have already been accounted for. Thus this approximation also models partially the effect of inter-object occlusion. Another benefit of using this model is that, because the plane going through point P gives the worst-case occlusion with respect to P of all other plane positions (see Figure 4.7), only planes going through P need to be considered in occlusion calculations to give a conservative estimate of the impact of occlusion. That means that only orientation is needed to parameterize these planes. This leads to a simpler computation and sampling process in the overall uncertainty estimation, as

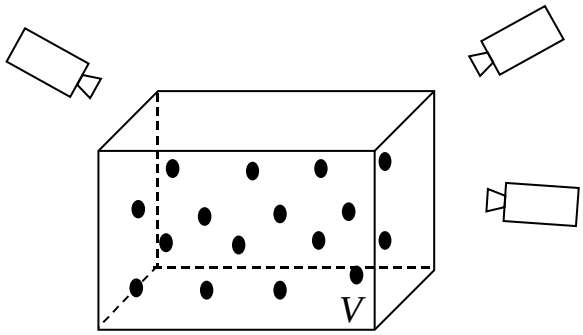


Figure 4.8: The quality metric with respect to a whole 3D volume is defined to be some norm of the per-point 3D uncertainty of all of the sampled points in the volume.

Equation (4.4) can be further simplified to

$$U(P) = \sum_{\text{plane}_i \in \Gamma(P)}^{*} U_{\text{res}}(P, \text{plane}_i) \tag{4.5}$$

where $\Gamma(P)$ is the set of planes going through point P with all possible orientations and the orientations (normals) of the plane are drawn from a distribution.

4.3.5 Uncertainty of a volume

Equation (4.5) estimates the 3D positional uncertainty of a point P. Given that, it is not difficult to extend this notion to a whole volume. The uncertainty of a whole volume V can be considered to be some norm of the uncertainty values of all points P in that volume (see Figure 4.8), and that is defined to be the "quality" with respect to V given a camera configuration:

$$q = \underset{P \in V}{\text{norm}_k}\, U(P) \tag{4.6}$$

where $U(P)$ is the uncertainty for a point P as defined in Equation (4.5), and norm_k denotes the kth norm of a set. There are an infinite number of points in a volume; therefore, in actual computation, we evaluate a discrete number of points by drawing them from a distribution that describes how likely a target is to be at point P within volume V.

4.3.6 Non-uniform distributions

Note that, in order to evaluate Equations (4.5) and (4.6), the probability distribution for occluder orientations and the probability distribution for target positions need to be known a priori. What distributions should we use? The answer is that it is application specific and it depends on how much information about the target motion and the tracking session is known. If one has no other information, a uniform distribution can be used. If one has more information, more specific distributions should be used. For example, as shown in Figure 4.9, in an VR application where a person's head position is tracked to drive a display, more accurate tracking is needed when the person is near the display since a slight movement close to the screen would cause a huge movement of the contents on the display. Another example is that when the feature point is at the top of a person's head and we know that the head will not face downward, we can limit the occluder orientation distribution to include only those directions pointing upwards. This shows that one benefit of using sampling in our quality metric formulation is that it easily allows application-specific constraints and/or requirements to be included in the quality evaluation.

4.3.7 Summary of our proposed metric

Based on the analysis of the previous section, we summarize the final version of our proposed quality metric as:

$$q = \underset{P \in V}{\text{norm}_k} \, \text{Prob}(P) U(P) \tag{4.7}$$

$$= \underset{P \in V}{\text{norm}_k} \, \text{Prob}(P) \left[\sum_{\text{plane}_i \in \Gamma(P)}^{*} \text{Prob}(\text{plane}_i) U_{\text{res}}(P, \text{plane}_i) \right] \tag{4.8}$$

where $\text{Prob}(X)$ denotes the probability distribution of a random variable X and $\Gamma(P)$ is the set of planes going through point P of all possible orientations. Both the points in the working volume and the occluder (plane) orientation are drawn from some a priori distribution.

The assumptions that we have made are that all feature points are independent and there

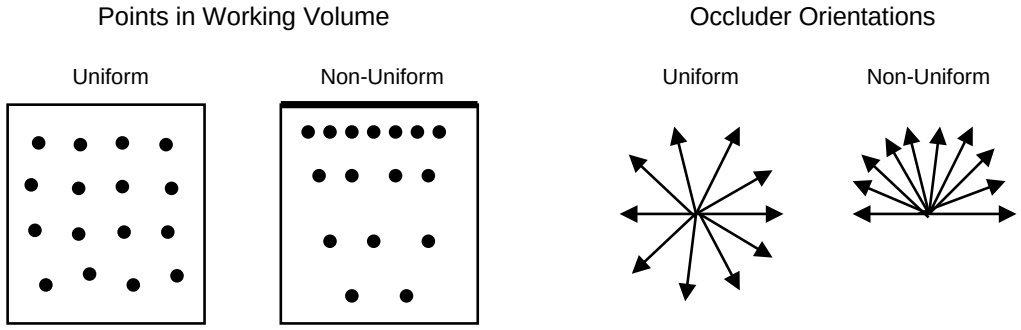


Figure 4.9: Examples of using non-uniform distributions in the computation of the quality metric. Left: uniform and non-uniform spatial point sampling in the an immersive room, a user's position is use to drive a display on one side of the wall (marked by a thicker black line). In order for the display program to have uniform response in terms of screen pixels, more accurate tracking is required when the user is close to the display because a slight movement close to the screen would cause a huge movement of the contents on the screen. Therefore, regions near the screen is sampled more frequently to give them a heavy weighting in the quality metric. Right: uniform and non-uniform sampling in occluder directions. An example scenario where the non-uniform distribution is used for tracking a feature point on the top of a person's head, and we know that occluder (the top of the head) will not face downward.

is only one occluder at a time; thus inter-object occlusion is ignored.

4.3.8 Practical issues

The quality metric defined above is the theoretical quantity that we would like to use to measure camera configurations. However, as can be seen from its formulation, in practice there are a few components in the definition which are up to the system designer to specify, such as which norm to use to combine the per-point uncertainty value, and which general function to use to aggregate the effect of multiple occluders. The decision is very application dependent. For example, for the choice of the k-norm with which to combine the spatial samples, if one wants to evaluate the worst-case, one should use the infinity-norm, which is the max() function. If one wants to evaluate the average quality, 1-norm (mean-error) or 2-norm (mean-squared-error) can be used. If one wants to combine this metric with some generic optimizer to obtain an optimal solution, it should be noted that the infinity-norm results in flat regions on the function landscape. Therefore, optimizers which rely on gradients in the landscape will not work optimally. We have found that a 2-norm works well in practice.

The choice of the general function $\sum^*$ to aggregate the effect of all possible occluders is even more flexible and application dependent. In other words, the function chosen should reflect what the designer "wants" or what is "desirable" based on the application. For example, a straight-forward and physically intuitive choice is again some sort of k-norm, indicating the final quality is an "average" based on all occluders. In fact, 2-norm will also be used for $\sum^*$ in the simulations presented later in this chapter. But this aggregation function can take on a more complex form than a simple k-norm, and is up to the system designer. The main contribution of this work is to provide the "components" of the quality metric, e. g. the resolution based uncertainty $L(P, \mathbf{d}, \mathbf{C})$ and the visibility test result $v(P, \mathbf{C}, \mathrm{oc})$ in Equation (4.3), and a framework to put them together. But the system designer can choose the specific ways to merge these components into a final quality metric.

For example, the way that Equation (4.3) is defined implies that if a point is being seen by no camera or only one camera, the size of the uncertainty volume is infinity because

there is one direction that has infinite uncertainty, i. e. instances where at least two cameras see a point will get finite (and probably small) uncertainty values. This agrees with physical principles, but for certain applications there may be other reasons where it is desirable to have at least three cameras see a point. In this case the system designer can completely ignore the definition of U_{res} in Equation (4.3), and simply count the number of instances in which 0, 1, 2, 3, etc. cameras can see the point. A very "bad" score can then be given for the cases where fewer than three cameras see the point; the final metric can be the individual scores for the number of unoccluded cameras weighted by their frequency of occurences. Another example is that, in Equation (4.3), the resolution-only based the uncertainty $L(P, \mathbf{d}, \mathbf{C})$ and $v(P, \mathbf{C}, \text{oc})$ are multiplied together. This has a physical interpretation that the overall uncertainty a point is the resolution-only based uncertainty *conditioned* on it being seen. In other words, once the point is occluded, v is infinity and the overall uncertainty is infinity, no matter how small the resolution-based uncertainty is. What this metric means is that, quantitatively, having occlusion is much more severe than bad resolution, which is physically correct. However, if in certain applications the system designer wants to guarantee certain minimum resolution and occlusion is a secondary concern, what he probably can do is to evaluate candidate configurations using the resolution-only metric, then out of the configurations that satisfy his resolution requirement, he can evaluate them again using our metric with the occlusion model and pick the least occlusion prone configuration. How to configure the quality metric to satisfy non-physically based requirements by certain applications is non-trivial and requires more future research, and this will be discussed in Chapter 5.

4.4 Experimental verification

Even though we formulated of our quality metric based on physical principles, we have made some assumptions and simplifications during the course of development, and therefore, we would like verify that the simplified model still reasonably explains reality. In order to do that, we set up 256 different camera configurations using up to 8 cameras. We selectively disable subsets of the 8 cameras, so that we have one setup with all cameras activated, 8 setups with 7 cameras activated, each with a different camera disabled, and

28 (i. e. 8 choose 2) setups with 6 camera activated, etc. Therefore the total number of different setups is 256. This way we only need to calibrate all 8 cameras once (using the method described in Chapter 3) but are able to obtain 256 different camera configurations to compare.

For each configuration, we attached a bright LED on various parts of a human body such as the head, shoulder, knee, etc. We let the person move around in a working volume and by processing the video streams from the various cameras, we are able to count at each sampled time instant, how many cameras can see the bright LED feature point. Aggregating the number of "visible" cameras of all of the frames of all of the sessions (with the LED placed on different body parts) yields a histogram for a given camera configuration, showing how often how many cameras can see a point attached to a body. Such a histogram is shown in Figure 4.10. This histogram gives us some idea of the characteristics of the occlusion for that particular configuration in motion tracking. Next we determine the camera parameters through our multi-camera calibration scheme and feed those as input to our quality metric. Given this configuration, we can also use the occlusion model in our proposed quality metric to predict how often a random point in the volume is seen by 0, 1, 2, 3, etc. cameras, and generate a predicted histogram for the number of visible cameras.We used a uniform distribution for both the occluder orientations and the target positions in the working volume. A good model should predict a histogram that is similar to what is obtained from the experiment. In order to see the "similarity" quantitatively, we use the common regression analysis technique [57].

In order to perform the regression analysis, one needs to come up with a single quantity for each experimentally obtained histogram and predicted histogram. Since it takes at least two cameras to perform triangulation to determine the 3D position of a point, it is important for 3D tracking systems to have at least two cameras seeing the feature point as often as possible. Therefore, by summing up the number of occurences of 2 through 8 visible cameras on the histogram and then dividing by the total number of cases, we are able to obtain an estimate of the probability of at least two cameras seeing a point for a given camera configuration. We compute this probability for each of the 256 camera configurations both from the experimentally obtained data and through our simulation, and the experimental and predicted probabilities are plotted against each other in Figure 4.11. A perfect model

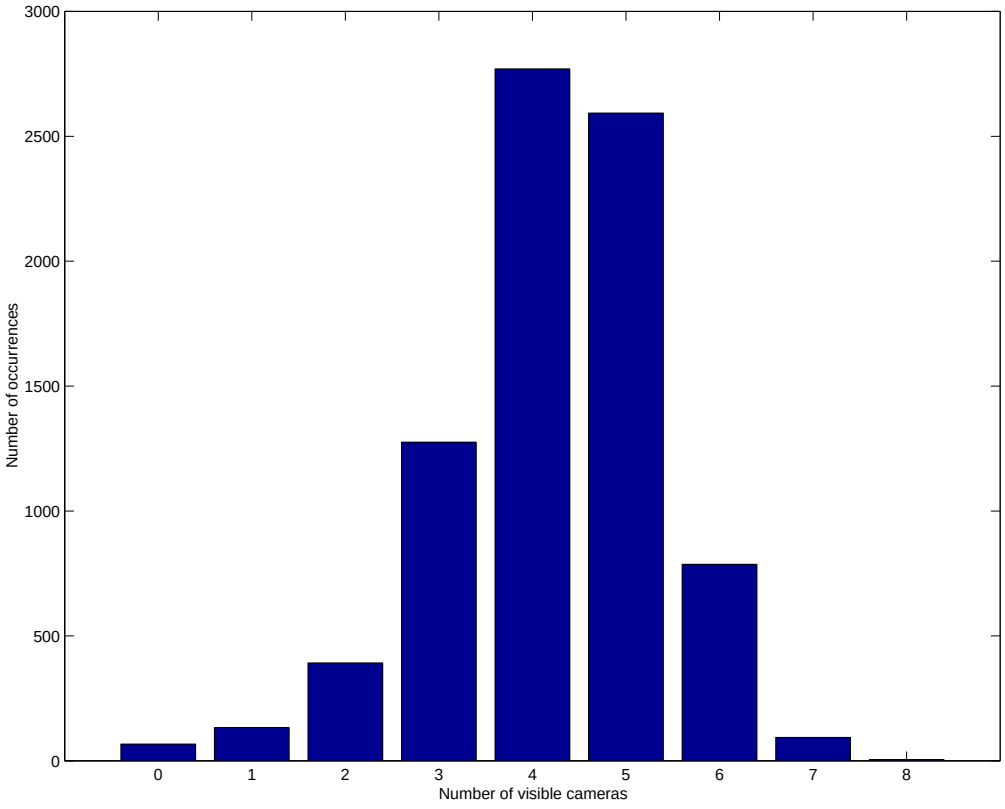


Figure 4.10: Histogram of the number of visible cameras of an 8-camera configuration, plotted from data obtained from 12 motion capture sessions.

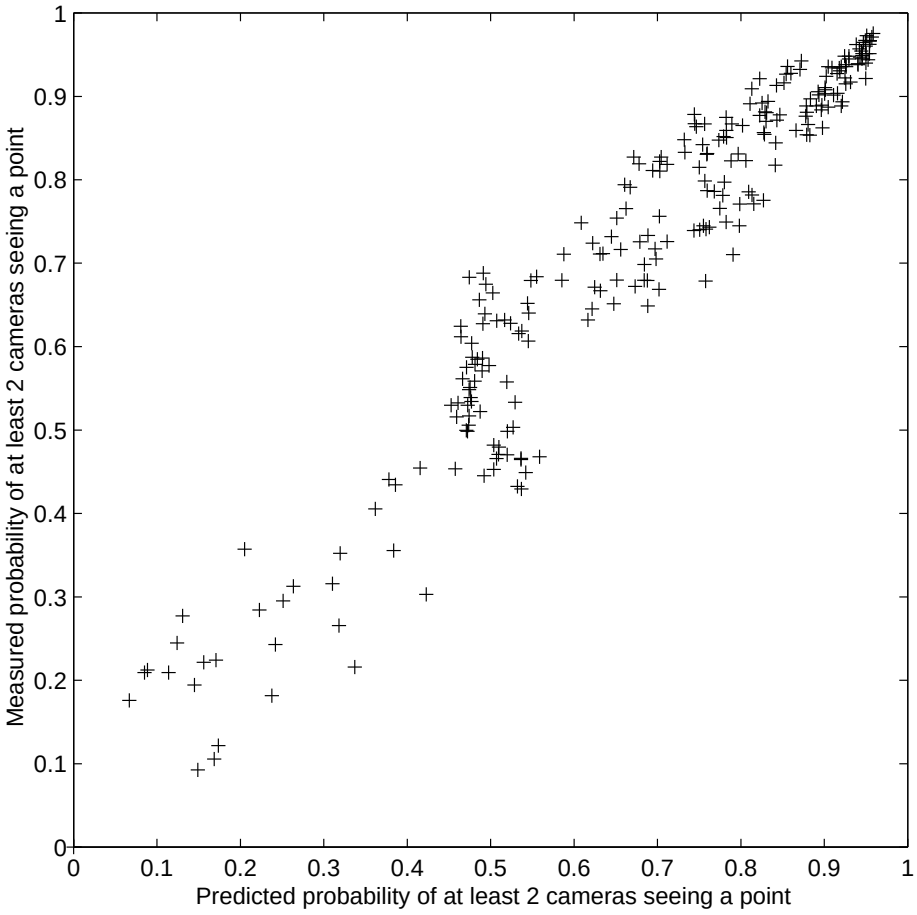


Figure 4.11: Correlation between the experimental and the model-predicted data. Each point represents a specific camera configuration. The x-axis is the probability predicted by our dynamic occlusion model, and the y-axis is the measured probability aggregated from all 12 experimental data sets.

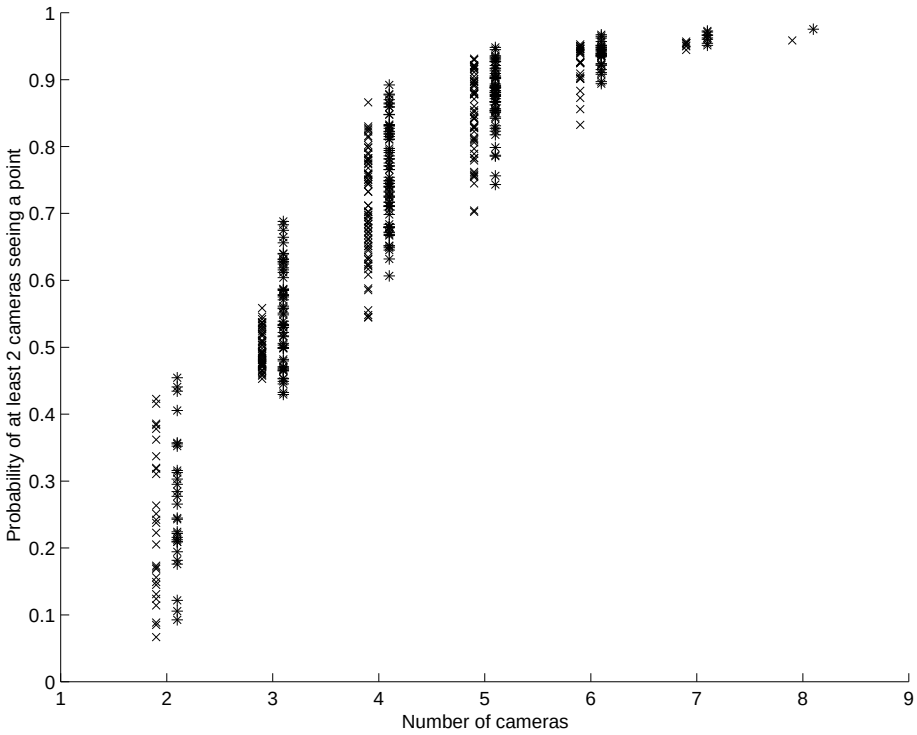


Figure 4.12: The predicted probability (shown by 'x's) and experimentally acquired probability (shown by '*'s) vs. the number of cameras in the configurations.

would generate a straight line from $(0, 0)$ to $(1, 1)$, showing perfect correlation. But the prediction from our model is quite good as well. The correlation coefficient between the experiment and prediction is 0.968, quite close to 1. Moreover, the correlation between the experimental data itself is 0.98, which means that there are variances in the data itself, and therefore that a correlation of 0.98 is the upper bound of how well the experimental data can be predicted. Given that, we conclude that the simplified occlusion model in our quality metric predicts the actual occlusion behavior quite adequately. The small variances that this model did not account for are probably due to the fact that we used a uniform distribution for occluder orientation and target position, which is not really true. A more specific and accurate distribution is expected to improve the correlation, and is the subject of future research.

The same data in Figure 4.11 can also be shown in another form in Figure 4.12, in

which the predicted and experimentally acquired probabilities are plotted again the number of cameras in the configuration. This graph shows that the more cameras there are in a system, the less often occlusion occurs.

4.5 Camera placement examples: impact of dynamic occlusion

The major way that we envision these metrics being used is in conjuction with some optimization process. The metric's role is to tell whether one configuration is better than another. However, it is even more interesting and useful to obtain some general intuition and guidelines as to what kind of camera configurations are "better" than others by analyzing various camera configurations. This is a more challenging task, since reaching general conclusions requires a lot of analysis, synthesis, and verification, and it is also not clear whether such a general trend exists. However, based on the resolution-only metric, previous researchers in computer vision have performed some analysis on a few simple configurations with 2-4 cameras and have developed some intuition as to what kind of camera setups are preferable over others [91, 60] in terms of uncertainty due to limited image resolution only. In this section we seek to do some thing similar. We consider several simple camera configurations, and evaluate them using both our quality metric, and the resolution-only metric. The objective of doing this is, first, to understand the interaction of resolution and occlusion, and whether they have similar or different requirements on camera placement patterns. Second, our previous intuition on camera placement was based on minimizing resolution-only based uncertainty, but since occlusion must also be considered in placing cameras for motion capture tasks, it is important for us to build intuition based on our new metric, that takes account of the impact of both resolution and (dynamic) occlusion on position uncertainty. More specifically, in the following examples, two different metrics are used for evaluation. One is the resolution-only metric, which is based on Equation (4.7) with $U(P)$ defined by Equation (4.2). Another metric is the resolution-occlusion combined metric, as defined by Equation (4.8). For both metrics, 2-norm is used for all aggregation functions. For the combined metric, the probability distributions used for dynamic occluder

directions and spatial point sampling are uniform distributions.

The first example shows the impact of occlusion on a camera placement pattern. Cameras are constrained to move on an outer sphere, and both have a fixed field of view (FOV) that is enough to cover the target sphere, as shown on the left of Figure 4.13. Because of symmetry, the parameter that really independently affects the quality is the relative angle θ between the two cameras. We plot the quality for θs from 0 to 180 degrees, with and without considering occlusion. If we consider resolution only, the best configuration occurs when the two cameras are oriented perpendicular to each other, since the ”good” direction of the 3D error of one camera cancels the ”bad” direction of the other. However, when the combined metric with dynamic occlusion is used, the best configuration is with the two cameras facing opposite to each other, because this increases the probability that there is at least one camera seeing the target space for occluders with arbitrary orientations. Note that in general, the result as to what is a ”preferable” pattern is closely tied to how the metric is defined, so any result should be interpreted in this context. Since we use 2-norm when aggregating the uncertainty values in various directions in Equation (4.2) as well as in Equation (4.3), it is a mean-squared-error. Consequently, when a point is not seen by any camera, the uncertainty in all directions is infinity (a huge number in practice), so the overall U_{res} is huge. When a point is seen by one camera, only one direction (the line connecting the target point and camera center) has a very large uncertainty value; the uncertainty values in the other directions are finite and relatively small, therefore the overall (average) uncertainty value for this point is small. What this means is that by using 2-norm, a point seen by one camera is better than a point seen by no cameras. Based on this assumption, when dynamic occlusion is considered, two cameras that look opposite to each other are preferred. If in an application a point seen by one camera is no better than one seen by no cameras, then the metric needs to be configured to reflect that by using a $\max()$ function in Equations (4.2) and (4.3).

The second example illustrates the impact of occlusion‘ on the number of cameras needed to cover a certain space. As in the first example, the task is to cover a spherical space; cameras are constrained to stay on an outer sphere with fixed FOV. We used an evolutionary algorithm based optimizer [85] to find the best quality achievable for a given number of cameras. Figure 4.14 plots the best quality vs. the number of cameras. It can

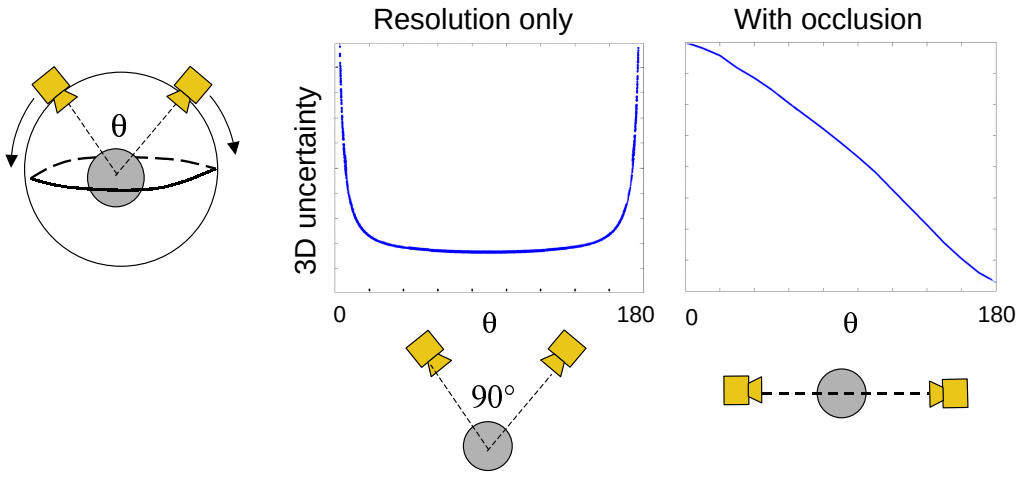
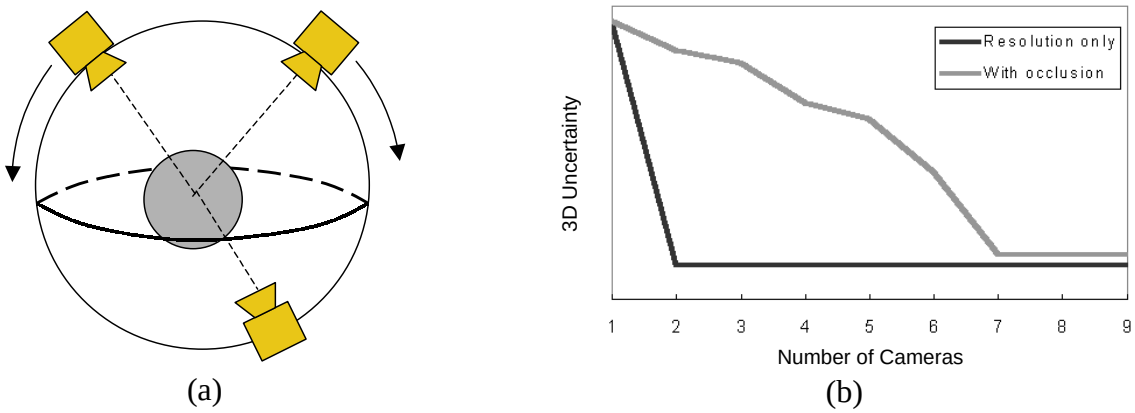


Figure 4.13: An example of placing two cameras. Left: Two cameras constrained to move on an outer sphere trying to cover a inner spherical target space; the angle with which they intersect the sphere center is θ. The occlusion and resolution metrics are minimized at different angles. Center: 3D uncertainty vs. θ considering resolution only. Right: 3D uncertainty vs. θ considering occlusion only. (θ is from 0 to 180 degrees.)

been seen that, if resolution is the only consideration, the best quality doesn't improve much after the space is covered by two cameras. However, if dynamic occlusion is considered, adding more cameras continues to improve the quality.

The next example demonstrates the impact of occlusion on the setting of camera focal length (or equivalently, the FOV). Two typical extreme configurations are considered (Figure 4.15). One configuration has two clusters of cameras. Each cluster consists of 4 narrow-angle cameras, each covering only one-quarter of the target space. The two clusters are placed so that they are 90 degrees apart on the outer sphere. This configuration is analogous to using two perpendicular high-resolution cameras. The other configuration consists of 8 cameras spread evenly around the sphere. In this configuration each camera's FOV is wide enough to cover the whole sphere. If no occlusion is considered, the narrow-angle solution gives the best overall resolution. If occlusion is considered, the wide-angle configuration is better since it is more robust. In the wide angle configuration each point is observed by eight cameras rather than two. This example shows that resolution improvement and occlusion reduction have conflicting requirements on the FOV of cameras.



(a) (b)

Figure 4.14: Quality vs. number of cameras. (a) N cameras constrained to move on an outer sphere trying to cover a inner spherical target space; (b) 3D uncertainty vs. number of cameras. Error due to resolution only is nearly minimized by only two cameras. Many more cameras are needed to ensure robustness against occlusion. Also note that the exact number of cameras needed to reduce the 3D uncertainty to a resonably small value depends on the relative radius of the spheres, so the ”7” on the graph is not a magic number. The larger the inner sphere compared to the outer sphere, the more cameras are needed because it is harder (or less likely) to ensure that a point on the surface of the inner sphere is seen by at least two cameras.

Thus, given limited camera resources, trade-offs have to be made between the importance of resolution and robustness depending on the specific application.

In the next example we demonstrate how one can apply the proposed metric in a more practical setting to automate the camera placement process, and the impact of dynamic occlusion.

We combine the metric with the evolutionary-algorithm-based optimizer in order to automatically find a placement solution. This optimizer is chosen because it is robust when function landscapes are discontinuous and also because of its ability to allow constraints on the variables that we want to optimize.

First we try to place three cameras of fixed field of view to cover a square floor, considering resolution only. The cameras are constrained to be on the ceiling. As shown in Figure 4.16(a), one of the configurations that the optimizer generated that gives good overall resolution puts two cameras far enough apart to just cover the whole floor, with the relative orientation between them approximately 90 degrees. What’s interesting is the pose of the 3rd camera. In the solution shown, it is placed on the ceiling looking straight down and

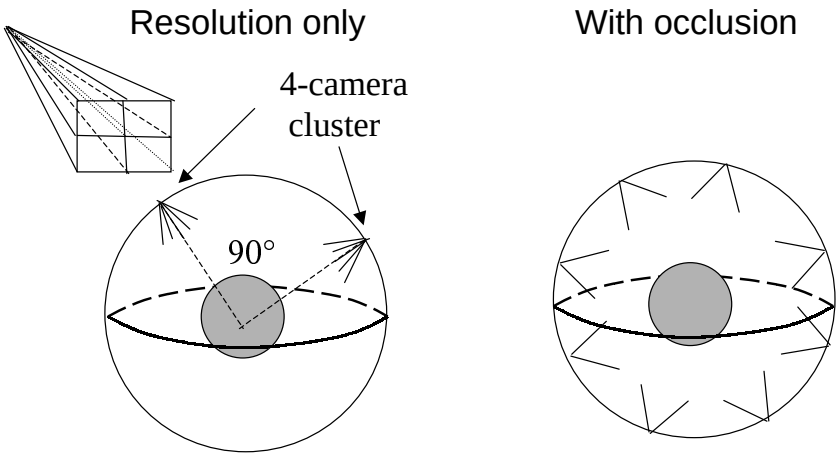
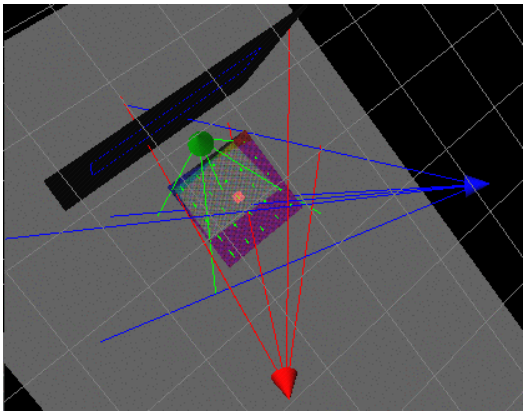


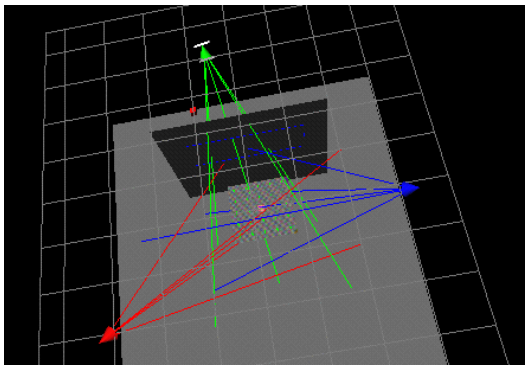
Figure 4.15: Effect of different FOVs. Left: Two camera clusters seperated by 90 degrees. Each cluster consists of 4 narrow angle cameras, equivalent to a single high resolution camera. Right: All cameras are wide-angle and evenly spread around the sphere in 3D. The diagram is a visualization only; the cameras are not coplanar. The resolution only metric prefers the former case, while the metric with occlusion considered prefers the latter.

covering the ”back” region relative to the other two cameras. This makes sense because we know that the uncertainty region for each camera is an ”ellipsoid” with the worst (longest) direction being the line connecting the target point and camera center. When two cameras are pointing in approximately perpendicular directions, the two bad directions cancel each other, and the resulting uncertainty volume for a point is like a sphere with more-or-less even uncertainty in all directions. However, the value of this uncertainty is larger for points that are farther away from the two cameras. Therefore, the region that is farther from the two cameras (red and blue ones in Figure 4.16) has worse 3D uncertainty due to resolution. To compensate, the 3rd camera is placed closest to that region to provide higher resolution.

In contrast, if our goal is to optimize for the least probability of occlusion, by applying the metric with a high occlusion weighting, the placement that the optimizer found is shown in Figure 4.16(b). As can be seen, this arrangement has all three cameras backed off so that each of them covers the whole square, i. e. every point on the floor is covered with all three cameras. This matches our intuition that the more cameras there are that see the target, the less often occlusion might happen.



(a)



(b)

Figure 4.16: An example of placing three cameras. Optimal placement patterns of three cameras constrained to be on the ceiling trying to cover a square floor. Colored ”shadows” are drawn and superimposed on the floor to show the cameras’ coverage. (a) in terms of resolution only. The purple part on the floor is the part covered by both the red and the blue camera. (b) in terms of the combined metric with dynamic occlusion. The cameras are backed off and evenly distributed around the patch. Every point on the patch is covered by all three cameras.

It is interesting to note that we have asked people in our lab to propose camera configurations. No one proposed a design similar to the optimizer’s optimal resolution placement; however, once everyone saw the computer generated design, all agreed that it makes sense. This illustrates the necessity of a quantitative approach to camera placement.

4.6 Conclusions

In this chapter we have proposed a metric to evaluate the quality of a multi-camera configuration, in terms of both resolution and occlusion. It includes a probabilistic occlusion model that reflects the dynamic self-occlusion behavior of the target which is commonly found in feature-based motion tracking systems. The computation of the model is sample-based, making it easily adaptable to applications with various occlusion characteristics. In addition, a metric that takes into account both resolution and occlusion allows the previously ad hoc process of placing multiple cameras to become an automated quantitative process.

The inclusion of the probabilistic based occlusion model makes it especially useful for designing motion capture systems and other occlusion-dominant tracking systems. In simple situations, it should do as well as a human designer's intuitive solution. In more complex situations, it enables the automatic generation of solutions that may take a human designer a long time to find. Moreover, it enables the finding of optimal camera configurations when the placement solution has to be automatically generated. We have verified the validity of the occlusion model by experimental data. We also have demonstrated how this metric can help us in understanding the design trade-offs of camera placement. Additionally, camera configurations can be designed automatically or semi-automatically for tracking systems with various requirement and constraints.

Chapter 5

Summary and Future Work

5.1 Summary of contributions

The non-intrusive nature of cameras makes them a very attractive choice for sensors for motion tracking, which has numerous applications in surveillance, entertainment and medicine. Recently, as the cost of imaging sensors as well as processors decrease, large scale tracking systems utilizing many cameras are emerging because they allow for a larger working volume, higher tracking accuracy, more robustness, and greater flexibility. However, the added complexity of such systems makes it difficult to calibrate all of the cameras into a single global coordinate frame in a scalable fashion. It also introduces the problem of how to optimally place (or configure) cameras to maximize the tracking performance. Another question is how the hardware and software architecture should be designed to accommodate the large number of cameras in a scalable and flexible manner.

The original contributions of this dissertation are:

First, we have developed a novel wide-area multi-camera calibration scheme. A system of many (possibly asynchronous) cameras can be calibrated with respect to a global coordinate system in minutes by simply waving a bright light in the working volume observed by the cameras. This scheme does not require physical measurement of 3D positions and thus is easy and quick to carry out; the method is adaptable to working volumes of any size and shape, and scalable to the number of cameras. Moreover, it can easily calibrate camera setups where not all cameras have an overlapping field of view, a situation that traditional

methods cannot not handle.

Second, we have proposed a quantitative metric for evaluating the tracking quality given a multi-camera placement configuration. Even though occlusion of tracked objects has a huge impact on motion tracking, previous work only uses the 3D uncertainty caused by limited image resolution of cameras to evaluate quality. Our metric considers both image resolution and the likelihood of target occlusion. In the formulation of the metric, we have proposed a novel probabilistic model that estimates the dynamic self-occlusion of targets and have verified its validity through experimental data. Using this metric, we have analyzed various camera placement patterns and shown the impact on camera placement requirements when considering either only resolution or both resolution and occlusion.

In addition, we have designed and implemented M-Track, a scalable tracking architecture for real-time motion tracking with tens of cameras. Its one-processor-per-camera design enables the parallel processing of high-bandwidth image data. The employment of a central estimator based on extended Kalman filtering allows the smooth and asynchronous integration of information from camera-processor pairs. Since cameras are not required to be synchronized, this architecture results in easier deployment, and enables the employment of heterogeneous sensors including cameras with different resolutions and frame rates. The central estimator also enables the tracking of multiple features and the continuous and automatic labeling of these features once they are labeled in the initial frame. In addition, it allows the continuous tracking even when some feature points are temporarily occluded. Three end-to-end VR and motion capture applications, namely LumiPoint, head tracking, and simultaneous body/face capture, are built upon this architecture to demonstrate the validity and usefulness of the system.

5.2 Future work

Many-camera systems have only recently been commercialized, and they can only be afforded by large, well-funded institutions in the government or the entertainment industry due to high-end requirements on necessary components and high operating costs (complexity and manual labor). Many-camera systems with commodity parts have just become feasible recently; the architecture and capabilities of these systems are still an active research

area because many open problems remain to be solved. This dissertation has addressed issues in camera calibration, placement and system design. Certainly more can be done to elaborate on the work reported in this dissertation, and even more can be done to address other issues in many-camera tracking that have not been the topic of this dissertation.

Calibration

Several avenues can be pursued to enhance our proposed multi-camera calibration scheme. One is to enable dynamic auto-calibration. Currently, cameras are calibrated *statically* in the sense that they are calibrated once after they are installed, and if afterwards some of their poses change (e.g. when bumped), a human operator needs to notice that the system is now miscalibrated and repeat our proposed calibration process. It the system needs to be running all the time, it is desirable for the system to *dynamically* "notice" a mis-calibration from the inconsistency among cameras and automatically re-calibrate the mis-aligned camera(s) based on observations obtained thus far. Another enhancement is to enable scalable intrinsic calibration. Our current methods only deal with the scalable calibration of extrinsic parameters. Although extrinsic parameters tend to change more frequently then intrinsic ones since any movement of cameras would result in a change, intrinsic parameter calibration is still carried out on a per-camera basis, and can be tiresome when there are a lot of cameras. It would be desirable to develop a scalable scheme that calibrates both extrinsic and intrinsic parameters.

Placement

As mentioned in Section 4.2, there has been relatively little work on camera placement issues for motion tracking applications. The quality metric described in this dissertation serves as a first step toward the final goal of optimal camera placement. A number of avenues can be pursued to carry this work further.

First of all, it would be very useful to build a user-friendly interactive camera planning system that utilizes this metric. The main contribution of our metric is to provide an approximate model to predict dynamic occlusion (i. e. the probabilities of target being seen

by 0, 1, 2, . . . , cameras) and to demonstrate the impact of considering occlusion on camera placement in addition to image resolution. We have seen that resolution and occlusion sometimes will result in conflicting requirements on camera placement patterns. We have, in our examples, lumped together the effect of the two using the notion of 3D position uncertainty, which is based on physical principles. However, in practical camera placement, users might have other ways to specify their requirements, and sometimes the requirements are not physically based. For example, "I want 3D resolution of at least x when a point does get seen, and only when that requirement is satisfied do I care about having less occlusion". Or "I want to maximize the probability of a feature point being see by at least three cameras rather than two, and I want the score for points being seen only by two cameras much worse than those seen by three". Statements like these reflect relative weightings between resolution and occlusion that are different from the physically-based one implied in the 3D uncertainty formulation. Therefore the final single-valued metric will also be different. It is important for a practical camera planning tool to provide a suitable interface for users to specify their particular needs and relative weighting, and then to "translate"these into a single metric. This metric will still consist of various components developed in this dissertation, such as the various occlusion probabilities, but they will just be combined in a user-defined function rather than the physically-oriented 3D uncertainty formulation.

Second, the performance prediction model in our current metric assumes that all feature points are independent. However, feature points are actually correlated in quite a few practical systems, especially the one employing more sophisticated algorithms. For example, points are temporally correlated in a system where occluded points are interpolated from their positions in previous and later frames; points are spatially correlated if a rigid body or articulated figure model is imposed on points seen in the same frame. In both of these cases, the occlusion of single point would not cause as disastrous of a result as in a system with multiple independent points. Therefore, the performance penalty due to that occlusion should not be as high. More elaborate models should be developed to handle these situations.

Another useful direction of research is to obtain more precise probabilistic models for human motion. As can be seen from the formulation of the quality metric in Section 4.3,

more accurate prediction of the occlusion behavior would require more specific probabilistic models for target motion in terms of their directional and spatial distributions.

Moreover, it would be desirable to develop quality metrics that evaluate other aspects of the tracking system performance besides 3D positional accuracy, as a function of camera configuration parameters in addition to position and orientations. For example, frame rate can be another camera parameter that can be included in a camera configuration, and latency is an important performance measure for many interactive systems. It would be desirable to develop a comprehensive model to allow designers to evaluate various configurations and to choose the best one that minimizes latency.

System design and general tracking

One goal for the system described in this dissertation is to provide a working prototype to support end-to-end applications and a test bed for further research in multi-camera tracking. Because of this, in terms of system design, more focus was given to developing a working end-to-end system that is adequate to support our required applications and flexible enough to be enhanced later. Our architecture choice only represents a point in the design space and further exploration on alternative designs should be carried out to find more efficient system solutions. Specifically, the many-to-one star topology between the clients and server would not be able to scale well when the number of cameras increases much from the current level, and a fully distributed solution, where all camera-processor pairs can communicate with each other, would be very desirable if we are interested in building systems with hundreds of cameras. Another promising direction would be hybrid tracking systems that include other types of sensors (such as acoustic or magnetic ones) besides optical cameras. The hope is that the different types of sensors can complement each other in terms of latency, range, rate, etc. A fully-distributed, wide area sensor *network* would be our ultimate goal.

Appendix A

Kalman Filtering: Theory and Implementation

In this appendix we first give background information on the discrete Kalman filter and its non-linear variation, the extended Kalman filter. Then we proceed to describe the detailed models and filter implementations in M-Track. Lastly we describe how we set the filter parameters. More details on Kalman filtering can be found in [14] and [5].

A.1 The discrete Kalman filter

Since R. E. Kalman published his paper in 1960 [47] describing a recursive solution to estimate the state of a linear discrete-time random process, the set of recursive equations he presented, the Kalman filter, has been the topic of extensive research. Extensions and variants have been proposed that have been widely used in applications ranging from the Global Positioning System to financial market prediction.

The filter operates in a cycle of prediction and adjustment. It predicts the process *state* (the parameters that we are interested in) based on a model of the process dynamics, and then adjusts this prediction based on the discrepancies between the predicted state and some actual *measurement* of the target. The measurement can be a direct measurement of the states, or an indirect measurement of quantities that are affected by the states. Both the dynamic prediction process and the measurement process are assumed to be corrupted

by some additive Gaussian noise with known variance. The filter maintains the first two statistical variants of the process state: the mean and the variance. The final estimate of the state is usually some weighted average of the predicted state and the measured state, where the weights reflect (inversely) how much *uncertainty* there is in the dynamic model and in the measurement model. The Kalman filter is optimal in the sense that it minimizes the expected mean squared error between the estimated and the "true" states, if certain conditions on the process, models and measurements are met. Though in practice these conditions are seldom met, the Kalman filter has been shown to be able to generate robust and satisfactory results in many less-than-ideal circumstances.

A.1.1 Mathematical models

The Kalman filter must have two mathematical models for prediction. The first is a dynamic model that describes how the current state is evolved from a previous state, i. e.

$$\mathbf{x}_{k+1} = \mathbf{A}_k \mathbf{x}_k + \mathbf{w}_k \tag{A.1}$$

The second is called a measurement model which describes how the measurement is derived from the current state, i. e.

$$\mathbf{z}_k = \mathbf{H}_k \mathbf{x}_k + \mathbf{v}_k \tag{A.2}$$

The notation and terms used for the vectors and matrices in Equations (A.1) and (A.2) are specified as the following:

$\mathbf{x}_k$**:** $(n \times 1)$ state vector at time t_k

$\mathbf{A}_k$**:** $(n \times n)$ matrix relating $\mathbf{x}_k$ to $\mathbf{x}_{k+1}$ in the absence of an external force or disturbance (or *state transition matrix* if each element of $\mathbf{x}_k$ is a sample of a continuous process, see Appendix A.3.2).

$\mathbf{w}_k$**:** $(n \times 1)$ process noise vector. Each element of $\mathbf{w}_k$ is assumed to be a white sequence with known covariance. A *white sequence* is defined as a sequence of zero-mean, uncorrelated random variables, e. g. the discrete analog of a continuous-time random

process. That is, the random variables within a given sequence (vector) have zero means and are mutually uncorrelated.

$\mathbf{z}_k$**:** $(m \times 1)$ measurement vector at time t_k

$\mathbf{H}_k$**:** $(m \times n)$ matrix describing the noiseless relationship between the measurement and the state vector at time t_k

$\mathbf{v}_k$**:** $(m \times 1)$ measurement noise vector. Each element of $\mathbf{v}_k$ is assumed to be a white sequence with known covariance and is uncorrelated with the elements of $\mathbf{w}_k$

The Kalman filter models the uncertainties in the dynamic prediction and the measurement processes as two independent, additive white noises. More precisely, the elements of both the process noise vector $\mathbf{w}_k$ and the measurement noise vector $\mathbf{v}_k$ are assumed to be white sequences. Moreover, the sequences within $\mathbf{v}_k$ are assumed to have zero cross-correlation with the corresponding sequences within $\mathbf{w}_k$. The covariance matrices of vectors $\mathbf{w}_k$ and $\mathbf{v}_k$ are assumed to be known as:

$$E[\mathbf{w}_k \mathbf{w}_i^T] = \begin{cases} \mathbf{Q}_k, & i = k \\ 0, & i \neq k \end{cases} \tag{A.3}$$

$$E[\mathbf{v}_k \mathbf{v}_i^T] = \begin{cases} \mathbf{R}_k, & i = k \\ 0, & i \neq k \end{cases} \tag{A.4}$$

$$E[\mathbf{w}_k \mathbf{v}_i^T] = 0, \quad \text{for all } k \text{ and } i \tag{A.5}$$

A.1.2 Derivations of the filter equations

Before the filter can begin to operate and estimate a process state from some measurement, two more assumptions must be made. First, we need to assume that at this point we already have an *a priori* estimate of the process state, $\hat{\mathbf{x}}_k^-$, based on our knowledge about the process prior to t_k. (The "hat" denotes estimate.) We also assume that we have some idea of the accuracy of this estimate. That is, we define the estimation error to be the difference between the "real" state and the estimated state:

$$\mathbf{e}_k^- = \mathbf{x}_k - \hat{\mathbf{x}}_k^- \tag{A.6}$$

and we define its associated error covariance matrix to be

$$\mathbf{P}_k^- = E[\mathbf{e}_k^- \mathbf{e}_k^{-T}] = E[(\mathbf{x}_k - \hat{\mathbf{x}}_k^-)(\mathbf{x}_k - \hat{\mathbf{x}}_k^-)^T] \tag{A.7}$$

This covariance matrix reflects the filter's own estimate of its state uncertainty. Note also that any change in the state estimate $\hat{\mathbf{x}}$ would result in a change in $\mathbf{e}$, which in turn would change the estimation covariance $\mathbf{P}$. Thus, $\mathbf{P}$ needs to be updated every time $\mathbf{x}$ is updated.

Given the prior estimate $\hat{\mathbf{x}}_k^-$, we now seek to use the measurement $\mathbf{z}_k$ to improve the prior estimate. Specifically, we choose to let the updated (*a posteriori*) estimate $\hat{\mathbf{x}}_k$ be a linear blending of the prior estimate $\hat{\mathbf{x}}_k^-$ and the difference between an actual measurement $\mathbf{z}_k$ and a predicted measurement $\mathbf{H}_k\hat{\mathbf{x}}_k^-$ as described by the equation

$$\hat{\mathbf{x}}_k = \hat{\mathbf{x}}_k^- + \mathbf{K}_k(\mathbf{z}_k - \mathbf{H}_k\hat{\mathbf{x}}_k^-) \tag{A.8}$$

where $\mathbf{K}_k$ is defined as the blending factor or *gain* that minimizes the error covariance matrix $\mathbf{P}_k$ associated with the updated estimate $\hat{\mathbf{x}}_k$

$$\mathbf{P}_k = E[\mathbf{e}_k \mathbf{e}_k^T] = E[(\mathbf{x}_k - \hat{\mathbf{x}}_k)(\mathbf{x}_k - \hat{\mathbf{x}}_k)^T] \tag{A.9}$$

(The justification for choosing the specific form of Equation (A.8) is beyond the scope of this short tutorial, and can be found in Section 5.8 of [14]). This minimization is achieved in several steps. First we substitute Equation (A.2) into Equation (A.8); next we substitute the resulting expression for $\hat{\mathbf{x}}_k$ into Equation (A.9). Then we take the derivative of the trace[1] of matrix $\mathbf{P}_k$ with respect to $\mathbf{K}_k$, set the derivative to zero, and solve for $\mathbf{K}_k$. The resulting optimal gain is

$$\mathbf{K}_k = \mathbf{P}_k^- \mathbf{H}_k^T (\mathbf{H}_k \mathbf{P}_k^- \mathbf{H}_k^T + \mathbf{R}_k)^{-1} \tag{A.10}$$

This particular $\mathbf{K}_k$ that minimizes the mean-square estimation error is called the *Kalman gain*. It is interesting to note from Equation (A.10) that, as the prior estimate error $\mathbf{P}_k$

[1]The *trace* of a matrix is its major diagonal.

approaches zero, the gain $\mathbf{K}_k$ approaches zero and as a result the updated estimate $\hat{\mathbf{x}}_k$ approaches the prior estimate $\hat{\mathbf{x}}_k^-$. On the other hand, as the measurement error $\mathbf{R}_k$ approaches zero, $\mathbf{K}_k$ approaches $\mathbf{H}_k^{-1}$ and consequently, the updated estimate $\hat{\mathbf{x}}_k$ approaches $\mathbf{H}_k^{-1}\mathbf{z}_k$, which can be interpreted as the state that is mapped back from the actual measurement using the inverse measurement model. The smaller the measurement error $\mathbf{R}_k$ is, the more "believable" the actual measurement is and the more the actual measurement contributes to the updated state estimate; on the other hand, the smaller the prior (predicted) estimate error $\mathbf{P}_k^-$ is, the more we believe the predicted state and the more the predicted state contributes to the final state estimate.

We now have a way to obtain an improved estimate of the state with the additional information provided by the measurement at time t_k, using of Equation (A.8) with $\mathbf{K}_k$ set to the Kalman gain. This requires the knowledge of $\mathbf{x}_k^-$ and $\mathbf{P}_k^-$, and similarly we need to know $\mathbf{x}_{k+1}^-$ and $\mathbf{P}_{k+1}^-$ at the next step in order to process a new measurement at t_{k+1}. The prior state for the next step $\mathbf{x}_{k+1}^-$ can be easily predicted through the use of the process model given by Equation (A.1), and the associated error covariance $\mathbf{P}_{k+1}^-$ can be evaluated by plugging $\mathbf{x}_{k+1}^-$ into Equation (A.7) and performing the indicated expectation.

To summarize, the Kalman filter is a set of equations that assimilate discrete measurements into optimal state estimates in a recursive fashion. The filter operates in a prediction-adjustment cycle. The equations in the prediction stage project forward in time the current state to obtain the a priori estimate for the next time step, and the equations in the adjustment stage incorporate measurements at the next time step into the a priori estimate to obtain an improved a posteriori estimate. The complete set of equations for the prediction stage and adjustment stage are listed below[2].

Prediction equations:

$$\hat{\mathbf{x}}_k^- = \mathbf{A}_{k-1}\hat{\mathbf{x}}_{k-1} \tag{A.11}$$

$$\mathbf{P}_k^- = \mathbf{A}_{k-1}\mathbf{P}_{k-1}\mathbf{A}_{k-1}^T + \mathbf{Q}_{k-1} \tag{A.12}$$

$$\hat{\mathbf{z}}_k = \mathbf{H}_k\hat{\mathbf{x}}_k^- \tag{A.13}$$

[2]All of the Kalman filter equations can have several mathematically equivalent forms. We will just list one form here.

Adjustment equations:

$$\mathbf{K}_k = \mathbf{P}_k^- \mathbf{H}_k^T (\mathbf{H}_k \mathbf{P}_k^- \mathbf{H}_k^T + \mathbf{R}_k)^{-1} \tag{A.14}$$

$$\hat{\mathbf{x}}_k = \hat{\mathbf{x}}_k^- + \mathbf{K}_k (\mathbf{z}_k - \mathbf{H}_k \hat{\mathbf{x}}_k^-) \tag{A.15}$$

$$\mathbf{P}_k = (\mathbf{I} - \mathbf{K}_k \mathbf{H}_k) \mathbf{P}_k^- \tag{A.16}$$

A.2 The extended Kalman filter

The original Kalman filter assumes linear dynamics and linear measurement models. However, many practical applications have non-linear dynamics and/or non-linear measurement relationships, and it is important to have a way of linearizing the problem. This can be handled by an extension of the original, called the *extended Kalman filter* (EKF). An EKF linearizes the model(s) about a trajectory that is continually updated with the state estimates resulting from the measurements. Though the EKF is sub-optimal due to the linearization, it has been successfully used in numerous applications with non-linear models in the past and continues to be one of the most important and widely-used extensions of the Kalman filter. As with the discrete Kalman filter, the EKF also operates in a prediction-adjustment cycle. The filter equations for the prediction stage and adjustment stage are listed below:

Prediction equations:

$$\hat{\mathbf{x}}_k^- = f(\hat{\mathbf{x}}_{k-1}) \tag{A.17}$$

$$\mathbf{P}_k^- = \mathbf{A}_{k-1} \mathbf{P}_{k-1} \mathbf{A}_{k-1}^T + \mathbf{Q}_{k-1} \tag{A.18}$$

$$\hat{\mathbf{z}}_k = h(\hat{\mathbf{x}}_k^-) \tag{A.19}$$

Adjustment equations:

$$\mathbf{K}_k = \mathbf{P}_k^- \mathbf{H}_k^T (\mathbf{H}_k \mathbf{P}_k^- \mathbf{H}_k^T + \mathbf{R}_k)^{-1} \tag{A.20}$$

$$\hat{\mathbf{x}}_k = \hat{\mathbf{x}}_k^- + \mathbf{K}_k(\mathbf{z}_k - h(\hat{\mathbf{x}}_k^-)) \tag{A.21}$$

$$\mathbf{P}_k = (\mathbf{I} - \mathbf{K}_k \mathbf{H}_k)\mathbf{P}_k^- \tag{A.22}$$

The basic operation of the EKF is very similar to that of the linear discrete Kalman filter, except for a few differences.

1. A non-linear function $f(\bullet)$ relates the state at time t_{k-1} to the state at time t_k, and another non-linear function $h(\bullet)$ relates the state to the measurement.

2. The definitions of $\mathbf{A}_k$ and $\mathbf{H}_k$ are different. They are now *Jacobian* matrices. $\mathbf{A}_k$ is the Jacobian matrix of partial derivatives of $f(\bullet)$ with respect to $\mathbf{x}_k$, and $\mathbf{H}_k$ is the Jacobian matrix of partial derivatives of $h(\bullet)$ with respect to $\mathbf{x}_k$. That is, the element at the ith row and jth column of the matrices are given by

$$\mathbf{A}_k[i,j] = \frac{\partial}{\partial \mathbf{x}_k[j]} f[i](\mathbf{x}_k) \tag{A.23}$$

$$\mathbf{H}_k[i,j] = \frac{\partial}{\partial \mathbf{x}_k[j]} h[i](\mathbf{x}_k) \tag{A.24}$$

Intuitively, Jacobian matrix $\mathbf{A}_k$ represents how much change occurs in the next state given a unit amount of change in the current state, i. e. the sensitivity of the next state to the current state; similarly, Jacobian matrix $\mathbf{H}_k$ describes the sensitivity of the measurement to a unit amount of change in the current state.

A.3 The Kalman filter implementation in M-Track

In this section we describe the (extended) Kalman filter implemented in M-Track. We shall describe the detailed state vector, dynamic model, measurement vector, measurement model, and the associated noise covariance matrices. We shall summarize the complete filter cycle including an additional step called *data association* that was not present in the theoretical filter cycle but is unavoidable and necessary in practice. We shall also describe how filter parameters are determined.

A.3.1 The Kalman filter cycle in M-Track

The state vector

In the current M-Track implementation, there are three types of targets with different associated Kalman *state vectors*. The target can be a single point, in which case we only need the position and its derivatives (velocity, acceleration, etc.) to describe its motion. The target can also be a rigid object with multiple LEDs mounted. In this case we need both the position and orientation (as well as their derivatives) to fully describe the motion. There is also the case where the target is a full articulated human body with multiple LEDs mounted. Even though it is possible to represent its motion using one absolute position together with a set of relative angles between linkages with known lengths (for example, see [12]), we choose to treat this case as multiple independent single points, each with a position vector. This is mostly for implementation simplicity because using this representation we can leverage our implementation of single point tracking and extend that to multiple instances[3]. Since a single point is just a degenerate case of a rigid body (with no need for orientation), in this subsection we will describe the Kalman models and parameters used to track a single rigid body. The approach that we took to extend the single target algorithm to track multiple targets is described in Section 2.2.3.2 of the main text.

Since the position and orientation (and their derivatives) completely describe the motion of a rigid body, we could just include them directly in the Kalman filter state vector

[3]Since we are mostly interested in tracking architecture issues in this work, we consider articulated figure modeling an orthogonal issue. Interested readers can find related work in [12, 37, 69, 54, 26, 51, 25, 58, 72, 10, 24, 45].

$\mathbf{x}$. The 3D Cartesian coordinates $\mathbf{p} = (x, y, z)$ and the Euler angles $\mathbf{r} = (\phi, \theta, \psi)$ are common representations for position and orientation, respectively. However, to avoid the non-linearities and singularities associated with Euler angles, in actual implementation we maintain the orientation as a quaternion *externally* to the Kalman filter, as in [13, 1, 88]. More specifically, in the internal filter state vector $\mathbf{x}$, while we maintain the target position directly using its Cartesian coordinates $\mathbf{p}$, for orientation we maintain the *change* in Euler angles or the *incremental* orientation using small Euler angles $\Delta\mathbf{r} = (\Delta\phi, \Delta\theta, \Delta\psi)$ around the (x, y, z) axis. At the end of each filter cycle, the updated incremental orientations $(\Delta\phi, \Delta\theta, \Delta\psi)$, which represent the interframe rotation, are "added onto" the external quaternion $\mathbf{q} = (q_w, q_x, q_y, q_z)$ to obtain the updated global orientation, and then zeroed before the next filter cycle as shown in Equation (A.41). Thus, the incremental Euler angles are linearized for the EKF, centered about zero. These incremental Euler angles do not overparameterize rotation and are approximately independent, and therefore can be used for reliable linearization. In addition, the internal state vector also includes the first derivatives of target position and orientation, i. e. the velocities in 3D and angular velocities about the (x,y,z) axis. (The angular velocities are included in the state vector directly because they behave more like independent vectors and do not exhibit the non-linearities of the Euler angles themselves). Therefore, we have

$$\mathbf{x} = (\mathbf{p}, \dot{\mathbf{p}}, \Delta\mathbf{r}, \dot{\mathbf{r}})^T = (x, y, z, \dot{x}, \dot{y}, \dot{z}, \Delta\phi, \Delta\theta, \Delta\psi, \dot{\phi}, \dot{\theta}, \dot{\psi})^T \tag{A.25}$$

as the complete EKF state vector for a rigid body. The global orientation

$$\mathbf{q} = (q_w, q_x, q_y, q_z) \tag{A.26}$$

is maintained externally and used in the linearization at each filter step.

The dynamic model

The dynamic model in a Kalman filter describes how the process to be estimated varies with time when no external force is exerted. In M-Track, the process to be estimated is the motion of the target, where the target is either human-held objects or an actual human.

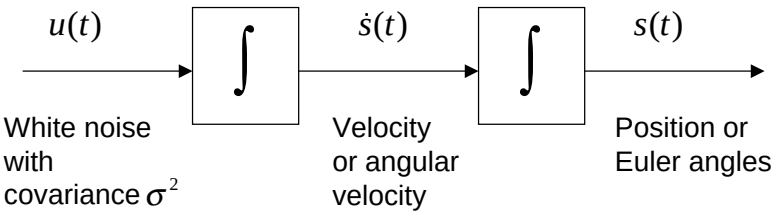


Figure A.1: Continuous-time constant velocity dynamic model. For each element s in the position and orientation state vector $\{x, y, z, \phi, \theta, \psi\}$, u is the corresponding white noise that drives that element. The linear and angular accelerations are thus modeled as zero-mean white noise. As a result, the linear and angular velocities are modeled as random walks (integrated random noise), and position and orientation are integrated random walks.

Modeling precisely the dynamics of human motion is a challenging on-going research problem in fields including bio-mechanics and medicine. In Kalman filtering, we are looking for models that are simple enough to be implementable, yet still describe the physical phenomenon with reasonable accuracy. In M-Track, we have chosen a simple *constant-velocity* model to describe the dynamics of target position and orientation, similar to previous work [13, 1, 88, 11] describing the tracking of human motion. While employing a different dynamic model might improve the tracking performance for certain situations, such *modeling* issues are not the focus of this thesis, and interested readers can find introductory materials in Chapter 10 of [14].

The constant velocity model assumes that the velocity of the target will stay constant but is subject to additive zero-mean random noise, e. g. the acceleration of the target is assumed to be zero-mean white noise. The velocity, as the result of the integration of the acceleration, is a random walk, and the position is an integrated random walk. This model is also used for the orientation. Angular acceleration is modeled as zero-mean white noise, and consequently, angular velocity is a random walk and orientation is an integrated random walk. This is shown in block-diagram form in Figure A.1.

As seen in Equation (A.47) in Appendix A.3.2, for this constant velocity model, the state transition matrix relating the element pair $(x, \dot{x})$ at two time instants that are Δt apart are given by $\begin{bmatrix} 1 & \Delta t \\ 0 & 1 \end{bmatrix}$, and this relationship holds for the remaining elements of the state vector shown in Equation (A.25). Thus, after aligning the state elements with their

corresponding derivatives, we obtain the complete state transition matrix $\mathbf{A}(\Delta t)$ for the full 12-member state vector $\mathbf{x}$ in Equation (A.25) as

$$\mathbf{A}(\Delta t) = \begin{pmatrix} \mathbf{I}_3 & \Delta t\mathbf{I}_3 & \mathbf{0}_3 & \mathbf{0}_3 \\ \mathbf{0}_3 & \mathbf{I}_3 & \mathbf{0}_3 & \mathbf{0}_3 \\ \mathbf{0}_3 & \mathbf{0}_3 & \mathbf{I}_3 & \Delta t\mathbf{I}_3 \\ \mathbf{0}_3 & \mathbf{0}_3 & \mathbf{0}_3 & \mathbf{I}_3 \end{pmatrix} \tag{A.27}$$

where $\mathbf{I}_3$ is the 3x3 identity matrix and $\mathbf{0}_3$ is the 3x3 zero matrix.

The process noise

The process noise vector $\mathbf{w}_k$ in Equation (A.1) models the uncertainty in the target motion. The elements of this vector are white sequences that model the responses of the corresponding elements of the state vector $\mathbf{x}$ to the white noise source u (i.e. acceleration) over time Δt. Even though the individual white sequences within $\mathbf{w}_k$ are uncorrelated in time, there are correlations between the white sequences within $\mathbf{w}_k$ at a given time instant t_k which are given by the process noise covariance matrix $\mathbf{Q}_k$. More specifically, in our discrete implementation, because each white sequence is assumed to be time stationary, $\mathbf{Q}_k$ is a function of the sample duration Δt only. Assuming the variance of the positional acceleration is $\sigma_{\mathbf{p}}^2$ in each of the (x, y, z) directions, and the variance of the rotational acceleration is $\sigma_{\mathbf{r}}^2$ in each of the (ϕ, θ, ψ) directions, $\mathbf{Q}(\Delta t)$ can be evaluated analytically from $\mathbf{A}(\Delta t)$ as shown in Appendix A.3.2, and is given by

$$\mathbf{Q}(\Delta t) = \begin{pmatrix} \frac{(\Delta t)^3}{3}\sigma_{\mathbf{p}}^2\mathbf{I}_3 & \frac{(\Delta t)^2}{2}\sigma_{\mathbf{p}}^2\mathbf{I}_3 & \mathbf{0}_3 & \mathbf{0}_3 \\ \frac{(\Delta t)^2}{2}\sigma_{\mathbf{p}}^2\mathbf{I}_3 & (\Delta t)\sigma_{\mathbf{p}}^2\mathbf{I}_3 & \mathbf{0}_3 & \mathbf{0}_3 \\ \mathbf{0}_3 & \mathbf{0}_3 & \frac{(\Delta t)^3}{3}\sigma_{\mathbf{r}}^2\mathbf{I}_3 & \frac{(\Delta t)^2}{2}\sigma_{\mathbf{r}}^2\mathbf{I}_3 \\ \mathbf{0}_3 & \mathbf{0}_3 & \frac{(\Delta t)^2}{2}\sigma_{\mathbf{r}}^2\mathbf{I}_3 & (\Delta t)\sigma_{\mathbf{r}}^2\mathbf{I}_3 \end{pmatrix} \tag{A.28}$$

Note that the off-diagonal elements of $\mathbf{Q}(\Delta t)$ model the instantaneous correlations between a position/angle element and its respective derivative. The actual values for $\sigma_{\mathbf{p}}^2$ and $\sigma_{\mathbf{r}}^2$ are determined by empirical experiments, which will also be described in Appendix A.3.2.

The measurement model

The measurement model is used to predict the noise-free sensor response given the filter's current state estimate as in Equations (A.25) and (A.26). In M-Track, since the measurements at any time are the 2D image coordinates of the LEDs on the target as observed by a particular camera, this mapping depends also on the parameters of the camera, as well as the position of the LED relative to the target coordinate frame. If there are a total of M point features (LEDs) on the target, the 2D coordinates $\hat{\mathbf{z}}_{c,m}$ of the mth LED on the image plane of camera c is

$$\hat{\mathbf{z}}_{c,m} = h_c(\mathbf{x}, \mathbf{q}, \mathbf{c}, \mathrm{l}_m) + \mathbf{v} \tag{A.29}$$

where $\mathbf{c}$ denotes the parameters of camera c and l_m denotes the 3D coordinates of the mth LED relative to the target local coordinate frame. The function $h(\bullet)$ computes the 2D projection of the mth LED from the complete state information $\mathbf{x}_t$ and $\mathbf{q}_t$, as well as from camera parameters and local LED coordinates. The detailed expression for $h(\bullet)$ is quite involved, but can be computed through a series of 3D and 2D coordinate transformations and geometric projections [55, 28]. Since $h(\bullet)$ is non-linear, the extended Kalman filter is used and a *measurement Jacobian* matrix $\mathbf{H}_{c,m}$ is computed according to Equation (A.24). This Jacobian describes the sensitivity of the measurement to changes in the Kalman states. Note that the matrix $\mathbf{H}_{c,m}$ has dimension $2 \times N$, and that there is a different $\mathbf{H}_{c,m}$ for each different camera and LED pair.

The measurement noise

The measurement noise reflects the uncertainty in the actual reported 2D image coordinates due to random error such as imperfect feature identification, noise in the video signal, etc. The noise on the mth LED is represented by the 2×2 measurement noise covariance matrix $\mathbf{R}_m$. Assuming that our 2D feature location process is independent on the two axes of the image plane, $\mathbf{R}_m$ is a diagonal matrix

$$\mathbf{R}_m = \begin{pmatrix} \sigma_{xi}^2 & 0 \\ 0 & \sigma_{yi}^2 \end{pmatrix} \tag{A.30}$$

where σ_{xi}^2 and σ_{yi}^2 are the variances of our 2D measurement in the x and y directions of the image plane, respectively. Again, the values of σ_{xi}^2 and σ_{yi}^2 are determined empirically (see Appendix A.3.2), and can vary among cameras if they are heterogeneous.

Measurement data association

The full *predicted* measurement vector is the concatenation of all LED observations $\hat{\mathbf{z}}_m$, $1 \leq m \leq M$. However, in actuality, a camera may not see all of the LEDs due to occlusion. Therefore, the *actual* measurement vector $\mathbf{z}$ usually has fewer elements than $\hat{\mathbf{z}}$. More important, in order to compute the measurement discrepancy or *residuals* in Equation (A.8), we need to determine, for each actual observed 2D coordinate, which one of the M LEDs has caused it. This problem of finding correspondences between the observations and predictions, known as the *data association* problem, has been an intense topic of research in the tracking community for years [5, 18], and there is not yet a completely satisfying and robust algorithm. We have adopted a modified version of the nearest neighbor algorithm presented in [5]. For each observed LED dot, we search through all predicted 2D locations $\hat{\mathbf{z}}_m$, $1 \leq m \leq M$, and associate the nearest predicted location with this observation. Once we find the associated predicted dots for all actual observed dots, we can construct the $2m \times 1$ predicted measurement vector $\hat{\mathbf{z}}$ by stacking up the appropriate individual $\hat{\mathbf{z}}_m$s associated with the actual measurement. The measurement noise covariance matrix $\mathbf{R}$ and the Jacobian matrix $\mathbf{H}$ for the m observed dots are formed in a similar fashion by stacking up the appropriate $\mathbf{R}_m$s and $\mathbf{H}_m$s, as illustrated in Figure A.2. Since the $\hat{\mathbf{z}}$, $\mathbf{R}$, and $\mathbf{H}$ associated with the set of LEDs that are actually seen are usually submatrices of the full forms associated with all of the LEDs, we use a subscript "sub" to distinguish them from their full counterparts. Obviously, the subset of the feature points being seen varies from frame to frame and camera to camera. Therefore, the sizes of $\hat{\mathbf{z}}_{\text{sub}}$, $\mathbf{R}_{\text{sub}}$, and $\mathbf{H}_{\text{sub}}$ vary from one filter cycle to another.

The initial states and state error covariance

As described in the previous section, the Kalman filter maintains and updates a $n \times n$ covariance matrix $\mathbf{P}$ that reflects the filter's estimate of the uncertainty in the states. In

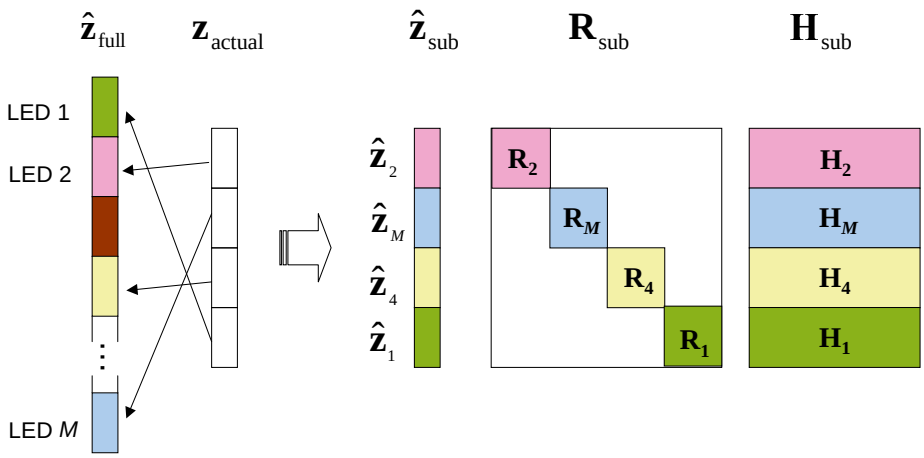


Figure A.2: Matching the set of actually observed 2D dots $\mathbf{z}_{\text{actual}}$ with the set of all predicted locations $\hat{\mathbf{z}}_{\text{full}}$. Usually only a subset of the predicted points are observed. The resulting prediction vector $\hat{\mathbf{z}}_{\text{sub}}$ that is used in calculating $\Delta\mathbf{z}$ later only contains entries corresponding to those observed points. Consequently, the $\mathbf{R}_{\text{sub}}$ and $\mathbf{H}_{\text{sub}}$ matrices used in computing the Kalman gain are also sub-matrices corresponding to only those observed points. Note that the length of vector $\Delta\mathbf{z}$ is $2m$, where m is the number of LEDs actually observed by a camera for that frame. For the same reason, the size of $\mathbf{R}_{\text{sub}}$ is $(2m \times 2m)$, with m (2×2) matrices $\mathbf{R}_m$ on its diagonal, and the size of $\mathbf{H}_{\text{sub}}$ is $(n \times 2m)$, where n is the length of the state vector.

order to bootstrap, an initial value of $\mathbf{P}$ needs to be given to the filter, along with the initial values of the states. We start the tracker by putting the target object in some canonical location and orientation, and feed the canonical position and orientation to the filter as the initial states. In practice, the target object does not have to be precisely positioned or oriented, and only needs to be close to the canonical location and orientation. We let $\mathbf{P}$ be a diagonal matrix with its diagonal elements set to be some large number, signifying that we don't know much about the actual position and orientation of the target. The elements of $\mathbf{P}$ become smaller as more measurements are incorporated into the filter, resulting in smaller uncertainties in the state estimates.

Summary of the complete filter cycle

The steps involved in a complete filter cycle with the specific models and parameters in M-Track is summarized below.

At the beginning of each filter cycle, the filter already has the knowledge of the prior internal state estimate $\hat{\mathbf{x}}_{k-1}$, its associated error covariance $\mathbf{P}_{k-1}$, and the prior external orientation $\hat{\mathbf{q}}_{k-1}$, for the time instant which was Δt before the current measurement. Now, to integrate the current measurement $\mathbf{z}$ from camera c with size $2m$ for m observed LEDs, the filter cycles through the following steps (the subscript k is dropped for all quantities corresponding to the current time instant for conciseness):

1. Predict the state and error covariance.

$$\hat{\mathbf{x}}^- = \mathbf{A}(\Delta t)\hat{\mathbf{x}}_{k-1} \tag{A.31}$$

$$\mathbf{P}^- = \mathbf{A}(\Delta t)\mathbf{P}_{k-1}\mathbf{A}^T(\Delta t) + \mathbf{Q}(\Delta t) \tag{A.32}$$

   where $\mathbf{A}(\Delta t)$ and $\mathbf{Q}(\Delta t)$ are given in equations (A.27) and (A.28), respectively.

2. Predict the measurement and compute the measurement Jacobian for all M feature points (LEDs) on the target.

$$\hat{\mathbf{z}}_m = h_{c,m}(\hat{\mathbf{x}}^-, \hat{\mathbf{q}}, \mathbf{c}, \mathbf{l}_m) \tag{A.33}$$

$$\mathbf{H}_m = \frac{\partial}{\partial \mathbf{x}}(\hat{\mathbf{x}}^-, \hat{\mathbf{q}}, \mathbf{c}, \mathbf{l}_m) \tag{A.34}$$

for $1 \leq m \leq M$. Note that there is a different $\mathbf{z}_m$ and $\mathbf{H}_m$ for every feature point.

3. Perform data association: for each observed LED dot find its associated predicted location. Construct the corresponding prediction vector $\hat{\mathbf{z}}_{\text{sub}}$ (of length $2m$), measurement noise matrix $\mathbf{R}_{\text{sub}}$ (of size $2m \times 2m$) and Jacobian matrix $\mathbf{H}_{\text{sub}}$ (of size $2m \times n$) by constructing the respective individual $\hat{\mathbf{z}}_m$s, $\mathbf{R}_m$s and $\mathbf{H}_m$s.

4. Compute the Kalman gain.

$$\mathbf{K} = \mathbf{P}^{-}\mathbf{H}^{T}(\mathbf{H}\mathbf{P}^{-}\mathbf{H}^{T} + \mathbf{R})^{-1} \tag{A.35}$$

5. Compute the residual between the actual measurement $\mathbf{z}$ and the associated predictions $\hat{\mathbf{z}}_{\text{sub}}$.

$$\Delta\mathbf{z} = \mathbf{z} - \hat{\mathbf{z}}_{\text{sub}} \tag{A.36}$$

6. Update the predicted state estimate and error covariance based on the measurement residual and the Kalman gain.

$$\hat{\mathbf{x}} = \hat{\mathbf{x}}^{-} + \mathbf{K}\Delta\mathbf{z} \tag{A.37}$$

$$\mathbf{P} = (\mathbf{I} - \mathbf{K}\mathbf{H}_{\text{sub}})\mathbf{P}^{-} \tag{A.38}$$

7. Update the external orientation by factoring in the incremental rotation $(\Delta\phi, \Delta\theta, \Delta\psi)$ in the state vector. The incremental rotation, expressed in quaternion form, is given by

$$\Delta\hat{\mathbf{q}} = (\sqrt{1-\epsilon}, \Delta\phi/2, \Delta\theta/2, \Delta\psi/2) \tag{A.39}$$

where $\epsilon = (\Delta\phi^2 + \Delta\theta^2 + \Delta\psi^2)/4$. This incremental rotation is then composed with the previous global quaternion to obtain the current global quaternion

$$\hat{\mathbf{q}} = \hat{\mathbf{q}}_{k-1} \otimes \Delta\hat{\mathbf{q}} \tag{A.40}$$

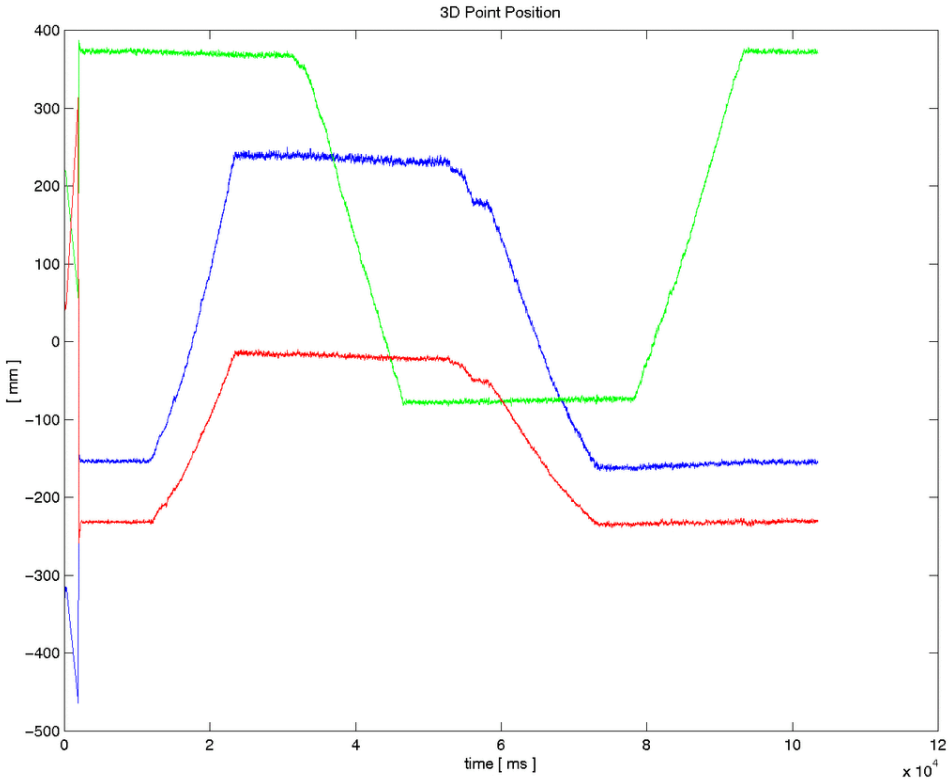


Figure A.3: An example state trace generated by the central estimator, showing the x (blue), y (green) and z (red) positions of a target point moving along a rectangular path in a plane almost parallel to the y-axis.

where $\otimes$ denotes a quaternion multiplication ([40, 41]).

8. Set the incremental rotation elements $(\Delta\phi, \Delta\theta, \Delta\psi)$ in the state vector $\mathbf{x}$ back to zero.

$$\Delta\phi = \Delta\theta = \Delta\psi = 0 \tag{A.41}$$

And $\hat{\mathbf{x}}$, $\mathbf{P}$ and $\hat{\mathbf{q}}$ become the priors for the next cycle when a new measurement comes in.

As an example, Figure A.3 shows the (x, y, z) position plot of a target moving in a rectangular path.

A.3.2 Determination of filter parameters

The Kalman filter requires that the process noise covariance matrix $\mathbf{Q}_k$ and the measurement noise covariance matrix $\mathbf{R}_k$, defined in Equations (A.3) and (A.4), be known prior to operation of the filter. In this section we briefly describe how to set the value of these matrices.

If the discrete model of Equation (A.1) comes from a sampled continuous-time system, which is very common in practice, the covariance matrix associated with $\mathbf{w}_k$ can be evaluated analytically from their continuous models. We begin with a a continuous process described by

$$\dot{\mathbf{x}} = \mathbf{F}\mathbf{x} + \mathbf{G}\mathbf{u} \tag{A.42}$$

where $\mathbf{u}$ is a vector forcing function whose elements are white noise, and $\mathbf{F}$ and $\mathbf{G}$ are the matrices relating the derivative of the state to the state itself and the external force, respectively . We consider the samples of this process at discrete times $t_0, t_1, \ldots, t_k, \ldots$. State space methods [46] can be used to solve Equation (A.42) and relate the sample of $\mathbf{x}$ at time t_{k+1} with the sample at t_k:

$$\mathbf{x}(t_{k+1}) = \mathbf{A}(t_{k+1}, t_k)\mathbf{x}(t_k) + \int_{t_k}^{t_{k+1}} \mathbf{A}(t_{k+1}, \tau)\mathbf{G}(\tau)\mathbf{u}(\tau)d\tau \tag{A.43}$$

where $\mathbf{A}_k$ can be evaluated from the continuous model by using the inverse Laplace (donated by $\mathcal{L}$) transform

$$\mathbf{A}(t_{k+1}, t_k) = \left[\mathcal{L}^{-1}[(s\mathbf{I} - \mathbf{F})^{-1}]\right]_{t=t_k}^{t_{k+1}} \tag{A.44}$$

Equation (A.43) can be equivalently written using a more concise notation,

$$\mathbf{x}_{k+1} = \mathbf{A}_k\mathbf{x}_k + \mathbf{w}_k$$

It can be clearly seen that this is the same as Equation (A.1), where $\mathbf{A}_k$ is the state transition matrix from t_k to t_{k+1}, and $\mathbf{w}_k$ is the driven response at t_{k+1} due to the white noise input

$\mathbf{u}_k$ during the (t_k, t_{k+1}) period. We can therefore analytically evaluate the $\mathbf{Q}_k$ matrix as

$$\begin{aligned}
\mathbf{Q}_k &= E[\mathbf{w}_k \mathbf{w}_k^T] \\
&= E\left\{ \left[\int_{t_k}^{t_{k+1}} \mathbf{A}(t_{k+1}, \xi)\mathbf{G}(\xi)\mathbf{u}(\xi)d\xi \right] \left[\int_{t_k}^{t_{k+1}} \mathbf{A}(t_{k+1}, \eta)\mathbf{G}(\eta)\mathbf{u}(\eta)d\eta \right]^T \right\} \\
&= \int_{t_k}^{t_{k+1}} \int_{t_k}^{t_{k+1}} \mathbf{A}(t_{k+1}, \xi)\mathbf{G}(\xi)E[\mathbf{u}(\xi)\mathbf{u}^T(\eta)]\mathbf{G}^T(\eta)\mathbf{A}^T(t_{k+1}, \eta)d\xi d\eta \qquad \text{(A.45)}
\end{aligned}$$

The matrix $E[\mathbf{u}(\xi)\mathbf{u}^T(\eta)]$ is basically a matrix of Dirac delta functions because it is the covariance matrix of the source white noise, and the amplitudes of these delta functions are presumably known from the continuous model.

As an example, for the constant-velocity process used in our M-Track system (shown in Figure A.1), the continuous model for an element, say x, in the position and orientation state vector $\{x, y, z, \phi, \theta, \psi\}$, in this case is

$$\begin{bmatrix} \dot{x} \\ \ddot{x} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} x \\ \dot{x} \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u(t) \qquad \text{(A.46)}$$

We can determine the corresponding discrete model (i. e. $\mathbf{A}_k$ and $\mathbf{Q}_k$) for a sampling interval of Δt. In fact, we can drop the k subscript because matrices $\mathbf{A}$ and $\mathbf{Q}$ are only dependent on Δt. First, the transition matrix can be evaluated simply as

$$\begin{aligned}
\mathbf{A}(\Delta t) &= \left[\mathcal{L}^{-1}[(s\mathbf{I} - \mathbf{F})^{-1}] \right]_{t=\Delta t} \\
&= \mathcal{L}^{-1} \begin{bmatrix} s & -1 \\ 0 & s \end{bmatrix}^{-1} = \mathcal{L}^{-1} \begin{bmatrix} \frac{1}{s} & \frac{1}{s^2} \\ 0 & \frac{1}{s} \end{bmatrix} \\
&= \begin{bmatrix} 1 & \Delta t \\ 0 & 1 \end{bmatrix} \qquad \text{(A.47)}
\end{aligned}$$

Next, given that we now know $\mathbf{A}_k$, and that $E[u(\xi)u(\eta)] = \sigma^2\delta(\xi - \eta)$, the covariance

matrix $\mathbf{Q}(\Delta t)$ can now be evaluated using Equation (A.45):

$$\begin{aligned}
\mathbf{Q}(\Delta t) &= \int_0^{\Delta t}\int_0^{\Delta t} \begin{bmatrix} 1 & \xi \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} \sigma^2\delta(\xi-\eta) \begin{bmatrix} 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ \eta & 1 \end{bmatrix} d\xi d\eta \\
&= \sigma^2 \int_0^{\Delta t}\int_0^{\Delta t} \begin{bmatrix} \xi\eta\delta(\xi-\eta) & \xi\delta(\xi-\eta) \\ \eta\delta(\xi-\eta) & \delta(\xi-\eta) \end{bmatrix} d\xi d\eta \\
&= \sigma^2 \begin{bmatrix} \frac{(\Delta t)^3}{3} & \frac{(\Delta t)^2}{2} \\ \frac{(\Delta t)^2}{2} & \Delta t \end{bmatrix}
\end{aligned} \tag{A.48}$$

The above analytical solution helps us to reduce the parameters to set for $\mathbf{Q}(\Delta t)$ to only setting the covariance of the noise source, σ^2. In this example, it represents the variance of the acceleration of the target motion. If the target is moving slowly, a smaller value of σ^2 should be used; if the dynamics of the target change rapidly, a larger σ^2 should be used. The exact value is determined by an offline tuning process. We record the measurements of some target moving with a known trajectory. We start with some initial guess of σ^2 and run the Kalman filter offline with this value to estimate the target trajectory. Next we calculate the error between the estimated trajectory and the known trajectory. We then vary the value of σ^2 and run the Kalman filter again to obtain a different estimated trajectory. After repeating this for several times, the one value of σ^2 that results in the minimum error is selected for use in the actual filter. This of course, assumes that the dynamics of the target used in filter parameter turning is similar to that of the target in actual tracking.

The setting of the measurement noise covariance matrix $\mathbf{R}$ for 2D feature location is based on rational analysis rather than empirical trial-and-error. Since it is a 2 by 2 matrix whose elements represent the covariance of the uncertainty in the x, y locations in the 2D image plane, we can determine these values by simple analysis. Based on our 2D feature detection process, we can assume that the location of the x and y coordinates of a feature are uncorrelated. That means that the off-diagonal elements of $\mathbf{R}$ are zeros. For the diagonal elements, again based on the precision of our cameras and our feature detection algorithm, we conclude that that we can locate a feature location within k_x pixels in the x direction and k_y pixels in the y direction. Therefore, the two diagonal values are set to be k_x^2 and k_y^2, respectively.

Bibliography

[1] A. Azarbayejani and A. P. Pentland. Recursive estimation of motion, structure, and focal length. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17(6):562–75, 1995.

[2] Ali Azarbayejani and Alex Pentland. Real-time self-calibrating sterea person tracking using 3-d shape estimation from blob features. In *Proceedings of the 13th International Conference on Pattern Recognition*, pages 627–32, 1996.

[3] R. Azuma and G. Bishop. Improving static and dynamic registration in an optical see-through hmd. In *21st International SIGGRAPH Conference, 24-29 July 1994, Orlando, FL, USA*, pages p.197–204. New York, NY, USA : ACM, 1994, 1994.

[4] R. Azuma and G. Bishop. A frequency-domain analysis of head-motion prediction. In R. Cook, editor, *SIGGRAPH '95, 6-11 Aug. 1995, Los Angeles, CA, USA*, pages p.401–8, 1995.

[5] Y. Bar-Shalom and T. E. Fortmann. *Tracking and Data Association*. Academic Press, 1988.

[6] J. Batista, P. Peixoto, and H. Araujo. Real-time active visual surveillance by integrating peripheral motion detection with foveated tracking. In *1998 IEEE Workshop on Visual Surveillance, 2 Jan. 1998, Bombay, India*, pages p.18–25, 1998.

[7] A.M. Baumberg and D.C. Hogg. An efficient method for contour tracking using active shape models. In *1994 IEEE Workshop on Motion of Non-rigid and Articulated Objects, 11-12 Nov. 1994, Austin, TX, USA*, pages p.194–9, 1994.

[8] E. A. Bier and S. Free. MMM: A user interface architecture for shared editors on a single screen. In *Proceedings of the ACM Symposium on User Interface Software and Technology*, pages 79–86, 1991.

[9] S. Birchfield. Elliptical head tracking using intensity gradients and color histograms. In *1998 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 23-25 June 1998, Santa Barbara, CA, USA*, pages p.232–7. Los Alamitos, CA, USA : IEEE Comput. Soc, 1998.

[10] M.J. Black and A.D. Jepson. Eigentracking: robust matching and tracking of articulated objects using a view-based representation. *International Journal of Computer Vision*, 26(1):63 – 84, 1998.

[11] C. Bregler. Learning and recognizing human dynamics in video sequences. In *Proceedings. 1997 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (Cat. No.97CB36082)*, pages 568–74. IEEE Comput. Soc, Los Alamitos, CA, USA, 1997.

[12] C. Bregler and J. Malik. Tracking people with twists and exponential maps. In *Proceedings. 1998 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (Cat. No.98CB36231)*, pages 8–15. IEEE Comput. Soc, Los Alamitos, CA, USA, 1998.

[13] Ted J. Broida and Rama Chellappa. Estimation of object motion parameters from noisy images. *IEEE transactions on pattern recognition and machine intellegence*, PAMI-8(1):90–99, 1986.

[14] Robert B. Brown and Patrick Y.C. Hwang. *Introduction to random signals and applied Kalman filtering*. J. Wiley, New York, 1992.

[15] X. Chen, J. Davis, and P. Slusallek. Wide area camera calibration using virtual calibration objects. In *IEEE Conference on Computer Vision and Pattern Recognition. CVPR 2000, 13-15 June 2000, Hilton Head Island, SC, USA*, pages p.520–7 vol.2, 2000.

[16] Xing Chen and James Davis. Camera placement considering occlusion for robust motion capture. Technical Report CS-TR-2000-07, Computer Science Dept., Stanford University, 2000.

[17] Motion Analysis Cooperation. *http://www.motionanalysis.com/*.

[18] I.J. Cox. A review of statistical data association techniques for motion correspondence. *International Journal of Computer Vision*, 10(1):53 – 66, February 1993.

[19] T. Darrell, P. Maes, B. Blumberg, and A.P. Pentland. A novel environment for situated vision and behavior. In *Workshop on Visual Behaviors, 19 June 1994, Seattle, WA, USA*, pages p.68–72, 1994.

[20] T. Darrell, B. Moghaddam, and A.P. Pentland. Active face tracking and pose estimation in an interactive room. In *IEEE Conference on Computer Vision and Pattern Recognition, 18-20 June 1996, San Francisco, CA, USA*, pages p.67–72. Los Alamitos, CA, USA : IEEE Comput. Soc. Press, 1996.

[21] James Davis. *Mixed Scale Motion Recovery*. PhD thesis, Stanford University, 2002.

[22] James Davis and Xing Chen. Mixed scale motion recovery using guidable cameras. Technical Report CS-TR-2000-08, Computer Science Department, Stanford University, December 2000.

[23] James Davis and Xing Chen. Lumipoint: Multi-user laser-based interaction on large tiled displays. *Displays*, 2002.

[24] Q. Delamarre and O. Faugeras. 3d articulated models and multiview tracking with physical forces. *Computer Vision and Image Understanding*, 81(3):328 – 57, March 2001.

[25] S.L. Dockstader and A.M. Tekalp. Tracking multiple objects in the presence of articulated and occluded motion. In *Workshop on Human Motion, 7-8 Dec. 2000, Los Alamitos, CA, USA*, pages p.88–95. Los Alamitos, CA, USA : IEEE Comput. Soc, 2000, 2000.

[26] T. Drummond and R. Cipolla. Real-time tracking of highly articulated structures in the presence of noisy measurements. In *Eighth IEEE International Conference on Computer Vision, 7-14 July 2001, Vancouver, BC, Canada*, pages p.315–20 vol.2, 2001.

[27] FaroArm. http://www.faro.com.

[28] Olivier Faugeras. *Three-Dimensional Computer Vision, A Geometric Viewpoint*. MIT Press, 1993.

[29] S. Fleishman, D. Cohen-Or, and D. Lischinski. Automatic camera placement for image-based modeling. *Computer Graphics Forum*, 19(2):101 – 10, June 2000.

[30] S.N. Fry, M. Bichsel, P. Muller, and D. Robert. Tracking of flying insects using pan-tilt cameras. *Journal of Neuroscience Methods*, 101(1):59 – 67, August 2000.

[31] Stefan Gottschalk and John F. Hughes. Autocalibration for virtual environments tracking hardware. In *ACM SIGGRAPH '93*, pages 65–72, 1993.

[32] François Guimbretière, Maureen Stone, and Terry Winograd. Flowmenu: Combining command text and parameter entry. In *Proceedings of UIST*, pages 213–216, 2000.

[33] François Guimbretière, Maureen Stone, and Terry Winograd. Fluid interaction with high-resolution wall-size displays. In *Proceedings of UIST*, 2001.

[34] I. Haritaoglu, D. Harwood, and L.S. Davis. Ghost: a human body part labeling system using silhouettes. In *Fourteenth International Conference on Pattern Recognition, 16-20 Aug. 1998, Brisbane, Qld., Australia*, pages p.77–82 vol.1, 1998.

[35] C. Harris. Structure-from-motion under orthographic projection. In O. Faugeras, editor, *Computer Vision - ECCV 90. First European Conference on Computer Vision Proceedings, 23-27 April 1990, Antibes, France*, pages p.118–23, 1990.

[36] Janne Heikkilä and Olli Silvén. A four-step camera calibration procedure with implicit image correction. *Proceedings IEEE CVPR'97*, pages 1106–1112, 1997.

[37] Y. Hel-Or and M. Werman. Constraint fusion for recognition and localization of articulated objects. *International Journal of Computer Vision*, 19(1):5 – 28, July 1996.

[38] A. Hilton. Towards model-based capture of a persons shape appearance and motion. In *IEEE International Workshop on Modelling People, 20 Sept. 1999, Kerkyra, Greece*, pages p.37–44, 1999.

[39] D. C. Hogg. *Interpreting Images of a Known Moving Object*. PhD thesis, University of Sussex, UK, 1984.

[40] B.K.P. Horn. Closed-form solution of absolute orientation using unit quaternions. *Journal of the Optical Society of America A (Optics and Image Science)*, 4(4):629 – 42, April 1987.

[41] B.K.P. Horn. Relative orientation revisited. *Journal of the Optical Society of America A (Optics and Image Science)*, 8(10):1630 – 8, October 1991.

[42] M. Isard and A. Blake. Contour tracking by stochastic propagaion of conditional density. In *4th European Conference on Computer Vision*, volume 1, pages 343–56, 1996.

[43] Y. Iwai, K. Ogaki, and M. Yachida. Posture estimation using structure and motion models. In *Seventh IEEE International Conference on Computer Vision, 20-27 Sept. 1999, Kerkyra, Greece*, pages p.214–19 vol.1, 1999.

[44] S.X. Ju, M.J. Black, and Y. Yacoob. Cardboard people: a parameterized model of articulated image motion. In *Second International Conference on Automatic Face and Gesture Recognition, 14-16 Oct. 1996, Killington, VT, USA*, pages p.38–44, 1996.

[45] Soonki Jung and Kwangyun Wohn. Tracking and motion estimation of the articulated object: a hierarchical kalman filter approach. *Real-Time Imaging*, 3(6):415 – 32, December 1997.

[46] Thomas Kailath. *Linear Systems*. Prenticel-Hall, 1980.

[47] R. E. Kalman. A new approach to linear filtering and prediction problems. *Transaction of ASME–Journal of Basic Engineering*, pages 35–45, March 1960.

[48] Takeo Kanade, Peter Rander, and P.J. Narayanan. Virtualized reality: Constructing virtual worlds from real scenes. *IEEE Multimedia, Immersive Telepresence*, 4(1):34–47, January 1997.

[49] Takeo Kanade, Hideo Saito, and Sundar Vedula. The 3d room: Digitizing time-varying 3d events by synchronized multiple video streams. Technical Report CMU-RI-TR-98-34, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, December 1998.

[50] Jean-Claude Latombe. *Robot Motion Planning*. Kluwer Academic Publishers, 1991.

[51] J. Luck, D. Small, and C.Q. Little. Real-time tracking of articulated human models using a 3D shape-from-silhouette method. In *Robot Vision. International Workshop Rob Vis 2001, 16-18 Feb. 2001, Auckland, New Zealand*, pages p.19–26, 2001.

[52] Kenneth Meyer and Hugh L. Applewhite. A survey of position trackers. *Presence*, 1(2), Spring 1992.

[53] T.B. Moeslund and E. Granum. Multiple cues used in model-based human motion capture. In *Fourth International Conference on Automatic Face and Gesture Recognition, 28-30 March 2000, Grenoble, France*, pages p.362–7, 2000.

[54] D.D. Morris and J.M. Rehg. Singularity analysis for articulated object tracking. In *1998 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 23-25 June 1998, Santa Barbara, CA, USA*, pages p.289–96, 1998.

[55] Richard Murray, Zexiang Li, and S. Shankar Sastry. *A Mathematical Introduction to Robotic Manipulation*. CRC Press, 1994.

[56] P.J. Narayanan, P.W. Rander, and T. Kanade. Constructing virtual worlds using dense stereo. In *IEEE 6th International Conference on Computer Vision, 4-7 Jan. 1998, Bombay, India*, pages p.3–10, 1998.

[57] John Neter, William Wasserman, and Michael H. Kutner. *Applied linear statistical models*. IRWIN, Homewood, IL 60430; Boston, MA 02116, fourth edition, 1996.

[58] K. Nickels and S. Hutchinson. Model-based tracking of complex articulated objects. *IEEE Transactions on Robotics and Automation*, 17(1):28 – 36, February 2001.

[59] N. Ohta. Structure from motion with confidence measure and its application for moving object detection. *Transactions of the Institute of Electronics, Information and Communication Engineers D-II*, J76D-II(8):1562 – 71, August 1993.

[60] M. Okutomi and T. Kanade. A multiple-baseline stereo. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15(4):353 – 63, April 1993.

[61] G. Olague and R. Mohr. Optimal camera placement to obtain accurate 3d point positions. In *Fourteenth International Conference on Pattern Recognition, 16-20 Aug. 1998, Brisbane, Qld., Australia*, pages p.8–10 vol.1, 1998.

[62] J. Oliensis. Multiframe structure from motion in perspective. In *IEEE Workshop on Representation of Visual Scenes (In Conjunction with ICCV'95), 24 June 1995, Cambridge, MA, USA*, pages p.77–84, 1995.

[63] J. Oliensis and Y. Genc. New algorithms for two-frame structure from motion. In *Seventh IEEE International Conference on Computer Vision, 20-27 Sept. 1999, Kerkyra, Greece*, pages p.737–44 vol.2, 1999.

[64] J. Oliensis and M. Werman. Structure from motion using points lines and intensities. In *IEEE Conference on Computer Vision and Pattern Recognition. CVPR 2000, 13-15 June 2000, Hilton Head Island, SC, USA*, pages p.599–606 vol.2, 2000.

[65] Joseph O'rourke. *Art gallery theorems and algorithms*. Oxford University Press, 1987.

[66] P. Peixoto, J. Batista, and H. Araujo. A surveillance system combining peripheral and foveated motion tracking. In *Fourteenth International Conference on Pattern Recognition, 16-20 Aug. 1998, Brisbane, Qld., Australia*, pages p.574–7 vol.1. Los Alamitos, CA, USA : IEEE Comput. Soc, 1998.

[67] Network Time Protocol. url = http://www.ntp.org/.

[68] Peter Rander, P.J. Narayanan, and Takeo Kanade. Virtualized reality: Constructing time-varying virtual worlds from real events. In *Proceedings of IEEE Visualization '97*, pages 277–283, Phoenix, Arizona, October 1997.

[69] J.M. Rehg and T. Kanade. Model-based tracking of self-occluding articulated objects. In *IEEE International Conference on Computer Vision, 20-23 June 1995, Cambridge, MA, USA*, pages p.612–17, 1995.

[70] J. Rekimoto. A multiple device approach for supporting whiteboard-based interactions. In *Proceedings of CHI*, pages 344–351, 1998.

[71] G. Rigoll, S. Eickeler, and S. Muller. Person tracking in real-world scenarios using statistical methods. In *Fourth International Conference on Automatic Face and Gesture Recognition, 28-30 March 2000, Grenoble, France*, pages p.342–7, 2000.

[72] M. Ringer and J. Lasenby. Modelling and tracking articulated motion from multiple camera views. In B. Mirmehdi, M.; Thomas, editor, *BMVC2000. Proceedings of the 11th British Machine Vision Conference, 11-14 Sept. 2000, Bristol, UK*, pages p.172–81 vol.1, 2000.

[73] Yong Rui, Liwei He, Anoop Gupta, and Qiong Liu. Building an intelligent camera management system. In *ACM Multimedia*, 2001.

[74] H. Saito and T. Kanade. Shape reconstruction in projective grid space from large number of images. In *1999 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 23-25 June 1999, Fort Collins, CO, USA*, pages p.49–54 Vol. 2, 1999.

[75] Heung-Yeung Shum, Qifa Ke, and Zhengyou Zhang. Efficient bundle adjustment with virtual key frames: a hierarchical approach to multi-frame structure from motion. In *1999 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 23-25 June 1999, Fort Collins, CO, USA*, pages p.538–43 Vol. 2. Los Alamitos, CA, USA : IEEE Comput. Soc, 1999, 1999.

[76] G. P. Stein. Tracking from multiple view points: Self-calibration of space and time. In *IEEE Computer Vision and Pattern Recognition*, volume I, pages 521–527, 1999.

[77] Vicon Motion Systems. http://www.vicon.com/.

[78] K.A. Tarabanis, P.K. Allen, and R.Y. Tsai. A survey of sensor planning in computer vision. *IEEE Transactions on Robotics and Automation*, 11(1):86 – 104, February 1995.

[79] K.A. Tarabanis, R.Y. Tsai, and P.K. Allen. The MVP sensor planning system for robotic vision tasks. *IEEE Transactions on Robotics and Automation*, 11(1):72 – 85, February 1995.

[80] Sundar Vedula Tekeo Kanade, Hideo Saito. The 3D room: Digitizing time-varying 3d events by synchronized multiple video streams. Technical Report CMU-RI-TR-98-34, Robotics Institute, Carnegie-Mellon University, 1998.

[81] New York Times. Turning the super bowl into a game of pixels, January 25 2001.

[82] G. Toscani and O.D. Faugeras. Structure from motion using discrete feature points. In *Onzieme Colloque sur le Traitement du Signal et des Images (Eleventh Symposium on Signal and Image Processing), 1-5 June 1987, Nice, France*, pages p.535–8, 1987.

[83] B. Triggs and C. Laugier. Automatic camera placement for robot vision tasks. In *Proceedings of 1995 IEEE International Conference on Robotics and Automation (Cat. No.95CH3461-1)*, pages 1732–7. Ieee, New York, NY, USA, May 1995.

[84] Roger Y. Tsai. A versatile camera calibration technique for high-accuracy 3d machine vision metrology using off-the-shelf tv cameras and lenses. *IEEE Journal of Robotics and Automation*, RA-3(4):323–344, August 1987.

[85] J. Wakunda and A. Zell. Eva: a tool for optimization with evolutionary algorithms. In *EUROMICRO 97. Proceedings of the 23rd EUROMICRO Conference: New Frontiers of Information Technology, 1-4 Sept. 1997, Budapest, Hungary*, pages p.644–51, 1997.

[86] G. Welch, G. Bishop, L. Vicci, S. Brumback, K. Keller, and D. Colucci. The hi-ball tracker: high-performance wide-area tracking for virtual and augmented environments. In M. Slater, editor, *VRST'99. Proceedings of the ACM Symposium on Virtual Reality Software and Technology, 20-22 Dec. 1999, London, UK*, pages p.1–188, 1999.

[87] G. Welch, G. Bishop, L. Vicci, S. Brumback, K. Keller, and D. Colucci. High-performance wide-area optical tracking. the hiball tracking system. *Presence*, 10(1):1 – 21, February 2001.

[88] Gregory F. Welch. *SCAAT: Incremental Tracking with Incomplete Information*. Ph.D. thesis, TR96-051, University of North Carolina – Chapel Hill, October 1996.

[89] C.R. Wren, A. Azarbayejani, T. Darrell, and A.P. Pentland. Pfinder: real-time tracking of the human body. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(7):780 – 5, July 1997.

[90] C.R. Wren, B.P. Clarkson, and A.P. Pentland. Understanding purposeful human motion. In *Fourth International Conference on Automatic Face and Gesture Recognition, 28-30 March 2000, Grenoble, France*, pages p.378–83, 2000.

[91] J.J. Wu, R. Sharma, and T.S. Huang. Analysis of uncertainty bounds due to quantization for three-dimensional position estimation using multiple cameras. *Optical Engineering*, 37(1):280 – 92, January 1998.

[92] S. Yi, R.M. Haralick, and L.G. Shapiro. Optimal sensor and light source positioning for machine vision. *Computer Vision and Image Understanding*, 61(1):122 – 37, January 1995.

[93] Zhengyou Zhang. An automatic and robust algorithm for determining motion and structure from two perspetive images. In *6th International Conference on Computer Analysis of Images and Patterns*, pages 174–81, 1995.