Real-Time Implementation of Time-Varying Surface Prediction and Projection

by

Keegan Aaron Fernandes

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Applied Science
in
Electrical and Computer Engineering

Waterloo, Ontario, Canada, 2019

Abstract

Spatial augmented reality makes use of projectors to transform an object into a display surface. However, for time-varying, non-rigid surfaces this can prove to be difficult, and often leads to image distortion. In order to avoid this highly accurate measurements of the surface are required. Traditional methods of measuring surface deformations are inadequate due to noise as well as potential sources of time delay, such as projector lag. To get more accurate results, a mass spring model can be used to simulate the dynamics of the time-varying surface. This model can be put into a nonlinear state space form to get a first order differential equation. Numerical integration techniques can then be used to solve the differential equation presented.

In order to reduce uncertainty in the model generated a filtering algorithm can be used. Both, the extended Kalman filter (EKF) and the cubature Kalman filter (CKF) are evaluated as potential candidates. To be able to run these filters in real time a reduced order model is developed. This enables the use of fewer mass nodes in the model, allowing for faster compute times. Additionally, to reduce visual error, an optimal node placement algorithm is used. This ensures that the surface generated by the mass spring mesh closely matches the real, curved surface of the system, minimizing error. The EKF and CKF algorithms are implemented onto a hanging cloth system perturbed by an oscillating fan. A parameter identification technique is used to create a model that accurately represents this hanging cloth system. Additionally, noise parameters of the EKF and CKF are adjusted to compensate for modeling errors and sensor noise. Finally, The mean squared error of the EKF and CKF algorithms are compared to evaluate their effectiveness. Both algorithms provide satisfactory results for use in spatial augmented reality applications. However, in all cases tested the CKF is shown to have significantly lower error values.

Although the CKF algorithm is shown to be more accurate than its EKF counterpart, its computation time is much larger. However, the computation time required is still within the threshold of being able to perform real-time estimation at up to 100Hz. Furthermore, due to the nature of the construction of the CKF, it can be applied as a multi-threaded workload to significantly reduce computation time.

Therefore, the implementation of a CKF algorithm can be used to accurately estimate the positions of a measured surface for use in spatial augmented reality.

Acknowledgements

I would like to thank my supervisor, David Wang, for all of his guidance and feedback during my Master's experience, it is greatly appreciated. I would also like to thank Adam Gomes, who assisted me considerably by providing ideas and sharing his knowledge, throughout my degree. Finally, I would like to thank my friends and family for their undoubted support and company

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Spatial augmented reality (SAR) is the use of projection technology for the purpose of transforming any object into a display surface. This, currently, is most often used by the entertainment industry to project large, sometimes user interactive, scenes onto walls or other rigid surfaces such as tables and buildings. Projection mapping onto non-rigid surfaces could be very useful in the entertainment and fashion industries, and the field of training simulators. Current rigid mapping algorithms, however, would not be able to function if significant deformations were to alter the surface, such as those involved with textiles, leading to a lack of realism and immersion. Current methods to solve this problem[71, 74, 78] involve the real-time tracking of surface geometry and projecting a warped image onto the measured surface. However, for quickly changing surfaces, there is no mention of how well these techniques perform. If a surface being tracked is moving quickly the image processing and surface tracking time required may cause delays that lead to image distortions. Other solutions[67] that have been shown to work for high speed deformations rely on highly customised and expensive projector and tracking systems. Additional issues that arise due to the nature of this problem include inherent system time delays for real time purposes as well as occlusions.

Projectors are notorious for having slow response times, also called input lag, which along with algorithm processing times can be a large damper on any real time effects. Occlusions occur when an object being tracked is blocked from observation and so can no longer be measured. A common occurrence when tracking non-rigid surfaces is self-occlusion where by the object itself prevents it from being fully measured. A prediction scheme can be used to approximate the position of the surface at some future time, which can smooth the overall experience. Such a scheme is suggested by Gomes [41], where a deformable model used in conjunction with a Kalman filter based prediction scheme was shown to have accurate position approximations in simulation. To implement a prediction scheme a physically accurate deformable model is needed. Traditionally, the field of computer graphics has studied the use of deformable models for use in animation, computer generated imaging, fashion, and video games [23, 28, 90]. These techniques often have a trade-off between being aesthetically pleasing and quick to run or physically accurate and computationally costly.

An application that will benefit greatly from time varying surface prediction is within the entertainment industry, and its increased use of projection projection to convey information and effects. Figure 1.1 shows a performer dancing while interacting with a projection being displayed onto them. The projection is set up to project onto a predetermined surface and is synchronised to change purely using time alone. When there is any movement by the performer the projection doesn't adapt to the movement and instead maintains the static projection on the predetermined position. The projection also doesn't account for the deformations on the performers clothing, or any changes in the surface of their clothing. To allow for a more realistic, and more interactive experience, dynamic projection mapping and surface estimation can be used. This would allow the show to be more fluid by accounting for any geometric deformations and movements.

Additional applications of this prediction scheme include projections onto clothes for both fashion purposes and for retail purposes. Training simulators could also greatly benefit from this technology. Surgical simulators, looking to enhance realism, could implement this dynamic projection scheme to allow for realistic animations to be projected onto a life like surgical dummy, as in Figure 1.2.

The remainder of this thesis will explain the details of this algorithms developed and the details involved their implementation. Chapter 2 reviews relevant background concepts that are required for deformable object modelling and spatial augmented reality. Chapter 3 examines the mass spring model in detail, including the model formulation, model linearization, integration methods required to simulate the system, and methods to reduce the model to allow for easier computation while maintaining accuracy. Chapter 4 reviews common linear least squares filtering approaches and their usefulness in the application of nonlinear deformable model estimation. Chapter 5 discusses how the extended these filtering techniques, namely the extended Kalman filter and the cubature Kalman filter, can be applied to the model to create deformable surface prediction filters. The prediction filters are applied to to a hanging towel experimental scenario, and the results are discussed. Lastly, Chapter 6 summarizes the results of the experiments and presents topics for future research.

Figure 1.1: Live performance involving projection mapping [2]

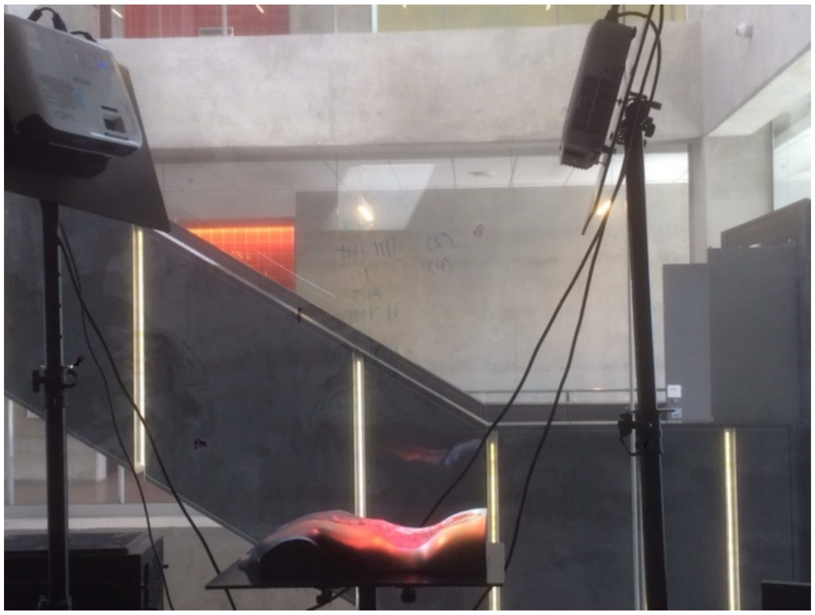


Figure 1.2: Surgical simulator projector setup

Chapter 2

Background

The use of non-rigid surfaces in spatial augmented reality, for the applications stated in this thesis, requires the development of a deformable model. In this chapter we review physically realistic modeling techniques commonly used in computer graphics to simulate deformable objects. Additionally, previous work on non-rigid object projection will be examined. Finally, projection mapping onto three-dimensional surfaces will be discussed.

2.1 Deformable Models

To properly estimate the surface of non-rigid objects, a physically accurate deformable model is required. Deformable models allow for the realistic simulation and analysis of objects in complex environments. The study of deformable model simulation in computer graphics has been an active research area since the 1980s. Two main categories of modeling techniques are non-physical and physically based methods [39]. Non-physical methods, such as free-form deformation, use purely geometric techniques to manipulate objects. These methods rely on the skill of the designer rather than the physical properties of the object being manipulated. This makes them difficult to use for complex systems . Alternatively, physical models allow the use of physical principles to compute a realistic simulation of complex processes [39]. Techniques such as mass spring models and finite element methods fall into this category. These approaches use numerical integration of accelerations and velocities to solve for the positions of of a system [29].

2.1.1 Non-Physical Model

One type of non-physical model is Free-form deformation (FFD). FFD is a general technique for changing the shape of objects. Object deformation is achieved by manipulating the space in which the object lies. Deformations are mapped using linear transformations (i.e. rotation and translation matrices) of the space, which in turn affect the objects which lie withing the space. FFDs, and non-physical models in general, are computationally efficient; however, they are not very intuitive to design. They require a high level of user experience and patience to be effectively utilized, since deformations need to be explicitly specified and the system is unaware of the physical nature of the object being manipulated. These difficulties make modelling complex structures nearly impossible using non-physical models [39].

2.1.2 Physical Models

In this section, two forms of physical models are analysed, mass spring models, and finite element models. The mass spring technique uses simple masses, springs, and dampers to model a system. Finite element models treat a surface like a continuous body and model the dynamics of a system using partial differential equations.

Mass Spring Models

Mass spring models are a technique that has been widely and effectively used for modelling deformable objects in the past [68]. In this method, an object is modelled using a collection of point masses interconnected via springs and dampers, forming the structure of the object. The dynamics of the system are represented by Newton's Second Law which governs the motion of each individual mass point in the model. Chapter 3 will discuss this method in more detail. Mass spring systems are simple physical models that are easy to construct and are computationally efficient, enabling their use in real-time applications. However, there are limitations. Mass spring models are developed by dividing a system into discrete mass points and are approximations of the true physics that occur. Parameter selection is another hindrance of this model. Determining optimal mass, spring, and damper parameters such that the simulation matches the real life system can be difficult. Recent studies have found close to optimal parameter values using learning algorithms or the physical properties of the real-world material. Teschner et al. [82] use generalized springs which preserve distances, areas and volumes. Bridson et al. [24] and Grinspun et al. [44] present a physically correct bending model by isolating the bending mode from all other modes of deformation for triangle meshes and simulating discrete shells respectively. Bhat et al. [16] use simulated annealing to estimate the spring constants in a cloth mesh. Eberhardt et al. [34] and Choi and Ko [31] improved the realism of the system by modeling nonlinear material properties.

Mass spring systems are prone to numerical instability when spring constants are chosen to be too large. For linear differential equations, it is known that numerically solutions are unstable if the time step (ΔT) chosen is greater that the natural period of the system T_0 [13]. The natural period of a linear system is given by

$$T_0 = 2\pi\sqrt{\frac{m}{k}} \tag{2.1}$$

where m is a given mass and k is the spring constant. The critical stiffness k_c can therefore be solved to be approximately,

$$k_c = m\frac{4\pi^2}{T_0^2} \tag{2.2}$$

Critical stiffness is the spring constant value at which the system becomes numerically unstable for a given time step $\Delta T = T_0$. Therefore, in order to increase the maximum value of the spring constant k, it is required to decrease the value of ΔT to maintain numerical stability. Several studies have tackled this issue. Provot [73] applied constraints to the movement of the mass nodes. This improves the numerical stability of the model, with the cost of physical accuracy. Baraff and Witkin [12] present a solution using alternate integration techniques, such as implicit integration, to improve model convergence. Others use both implicit integration to solve for the stiff portion of the models and explicit integration to solve for the non-stiff parts [11, 21, 45]

Finite Element Methods

An alternative physical model to mass spring systems are continuum models. These techniques treat objects like solid, continuous bodies with masses and energies distributed throughout the system. Continuum models consider the equilibrium of a general body acted on by external forces. The deformation is a function of these acting forces and the objects material properties. The partial differential equation (PDE) governing the dynamics of elastic materials is given by [68],

$$\rho\ddot{\mathbf{x}} = \nabla\cdot\sigma + \mathbf{f} \tag{2.3}$$

where ρ is the material density, $\mathbf{x}$ is the (x, y, z) position of the material, σ is the 3×3 stress tensor and $\mathbf{f}$ is a vector of externally applied forces. The portion $\nabla \cdot \sigma$ represents the internal forces in the deformed volume.

Since it is not always possible to find a closed-form solution to the PDE presented in Equation (2.3), a numerical approximation is desired. One of the most popular methods used is the finite element method (FEM) [68]. FEMs are used to turn a PDE into a set of algebraic equations which are then solved numerically. The method takes the continuous domain and discretize it into a finite number of elements. Therefore, instead of solving for the solution $\mathbf{x}(t)$ for the PDE for the entire body, it solves the PDE for each discrete element $\mathbf{x}_i(t)$. The solution $\mathbf{x}(t)$ can be then be approximated by linear combination of these discrete solutions, that is [68]

$$\tilde{\mathbf{x}}(t) = \sum_i \mathbf{b}_i \mathbf{x}_i(t) \tag{2.4}$$

where $\mathbf{b}_i$ are the nodal basis functions. The value of $\mathbf{b}_i$ is usually 1 at node i and 0 at all other nodes. Substituting $\tilde{\mathbf{x}}(t)$ into Equation (2.3) for $\mathbf{x}(t)$ results in a series of equations solving for each $\mathbf{x}_i(t)$, converting the PDE problem into a finite order ODE. Finding the solution is then viewed as an optimization problem minimizing the error between $\mathbf{x}(t)$ and $\tilde{\mathbf{x}}(t)$, as $\tilde{\mathbf{x}}(t)$ is not the exact solution to Equation (2.3).

To solve this problem in computer graphics, the explicit FEM method is used to solve for $\mathbf{x}_i(t)$. This is a simple form of the FEM that does not solve a system of equations for the positions $\mathbf{x}_i(t)$. The method treats nodes of the mesh as mass points, as in the mass spring model, and FEM elements as generalized springs connecting all adjacent mass points. The relationship between nodal forces and positions happen to be nonlinear, and when linearized, can be expressed as

$$\mathbf{f}_e = \mathbf{K}_e \mathbf{u}_e \tag{2.5}$$

where $\mathbf{f}_e$ are the nodal forces and $\mathbf{u}_e$ are the nodal displacements $(\mathbf{x} - \mathbf{x}_0)$ for all nodes connected to spring element e. The matrix $\mathbf{K}_e$ is the stiffness of the element e, and the stiffness for the entire mesh is given as the sum

$$\mathbf{K} = \sum_e \mathbf{K}_e \tag{2.6}$$

Using the linearized force-displacement relationship for each element, the equation of motion for the entire mesh becomes

$$\mathbf{M}\ddot{\mathbf{u}} + \mathbf{D}\dot{\mathbf{u}} + \mathbf{K}\mathbf{u} = \mathbf{F} \tag{2.7}$$

where **M** is the mass matrix, **D** is the damper matrix, **K** is the stiffness matrix, **F** is the applied force vector and **u** is node displacement vector. Often times, to save on computations, diagonal matrices are used for **M** and **D** in a method called mass-lumping [68]. In this method, **M** contains the mass of each node along its diagonal.

Finite element methods provide more physically realistic simulations, compared to mass spring systems, with fewer node points. However, due to the requirement of numerical integration for parameters of FEM systems, significant pre-processing time is required. Additionally, the linear elastic theory using FEM systems only assumes small deformations of the objects. If the position of a node moves greatly from its equilibrium point, the model is no longer physically accurate. Nonlinear FEM methods avoid this problem. However, they are computationally prohibitive [39].

2.2 Spatial Augmented Reality

Spatial augmented reality (SAR) is a technological variation of augmented reality. SAR merges the real world with a virtual world by superimposing computer-generated graphics onto real surfaces, usually with the use of projectors [20]. Since projectors traditionally project on flat rectangular surfaces, projection on non-planar surfaces require special rendering process. To accurately project images onto an object's surface, texture mapping needs to be performed on virtual models of the real life object. Texture mapping is the process by which an image is "painted" onto a virtual object by transforming the images coordinates to the coordinates of the object [77]. This texture mapped virtual object can then be projected onto the real world object to augment its appearance. For an accurate augmented mapping, a CAD model of the real world object is often used. This ensures that the virtual model being texture mapped matches the true object and so minimizes any projection disparities.

Projector based spatial displays have some negative qualities, namely the occurrence of shadows and occlusion, as well as the difficulties involved in calibration and maintaining alignment. Shadow and Occlusion correction methods have been studied by severally researchers. Sukthankar et al. [79] addresses the problem of undesired shadows by redundantly illuminating the display surface using multiple projectors in different locations. Punpongsanon et al. [74] use feature tracking techniques, where occluded features are ignored on each frame, to reduce projection error. Steimle et al. [78] track common elements that block projectors, such as hands and fingers, in order to to compensate for them while projecting.

2.2.1 Non-Rigid Projection Based SAR

Traditionally, projection based SAR has primarily focused on projection onto rigid objects, and very little work has been conducted on non-rigid objects. One of the first systems for projection on non-rigid surfaces was developed by Piper et al. [71]. In this system, users interacted with clay terrains while a projector superimposed depth related information onto the surface. Newer work, such as the work done by [54], project navigation information onto a patient's body, while the patient is undergoing surgery. To do this, the researchers estimate the bump deformation (deformation from pushing) using visual feature tracking. To capture the pose of a projection surface, a number of different strategies are used. These include marker and marker-less camera systems, texture feature selection [84], optical flow [46], and object contour analysis [30]. Recent work by Narita et al. [67] have used a special set of dot cluster markers arranged in a means to minimize marker swapping, which is the possibility of marker technologies to confuse one marker position for another in close proximity. High speed projection technology, called the DynaFlash, is also used by [67]. This allows projection of up to 1000 frames per second, albeit at the cost of expensive customised equipment. Work done by Fernandes et al. [37] and Gomes et al. [40] has shown that an estimation algorithm can be used on cost-effective, yet slower projectors, to accurately track moving surfaces.

2.2.2 Projection Mapping

Projection mapping is the general term used for converting irregular objects into a display surface. The rendering process requires a mathematical model for the projector, the shape of the display surface, and for cases where the user is not aligned with the display, the user's location. The display surface can be represented by a polygonal mesh using a piece-wise planar approximation. The mesh is defined by a set of vertices along with their associated normals to determine surface orientation. Common techniques for obtaining geometry is by using depth cameras or multiple camera set-ups such as in [67]. This allows the tracking of the nodes of the objects mesh.

A projector model is required to transform the real world (x, y, z) coordinates to its corresponding pixel position, also called UV coordinates (u, v). The projector model can be approximated by a pin-hole model, similar to a pin-hole camera model. As explained by Low [63], the pin hole camera model is defined by a 3×4 perspective projection matrix. To find the equivalent projection matrix for the projector model, let $m = [u, v]^T$ be the pixel location of the projector and $M = [X, Y, Z]^T$ be the associated point in three-dimensional space. These two points can be written as $\tilde{m} = [u, v, 1]^T$ and $\tilde{M} = [X, Y, Z, 1]^T$. The relationship between the pixel position m and 3D point M is found to be

$$w\tilde{m} = F \begin{bmatrix} R & t \end{bmatrix} \tilde{M} \tag{2.8}$$

where w is a scaling factor, R and t represent the rotation and translation required to convert the world coordinate system to the projector coordinate system, and F is the projector's intrinsic matrix defined as

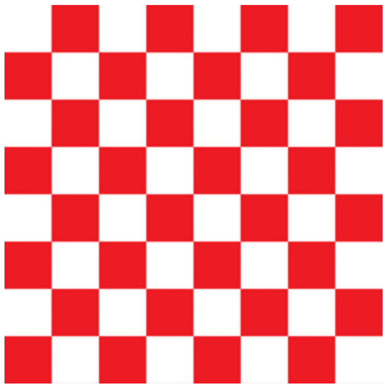
$$F = \begin{bmatrix} \frac{-2n}{r-l} & 0 & \frac{r+l}{r-l} \\ 0 & \frac{-2n}{t-b} & \frac{t+b}{t-b} \\ 0 & 0 & -1 \end{bmatrix} \tag{2.9}$$

where n is the distance from the projector to the plane of the surface, l, r, t, and b are the left, right, top and bottom extents of the display surface, relative to the centre of the projector. The equivalent projection matrix can be extracted form this formulation to be

$$P = F \begin{bmatrix} R & t \end{bmatrix} \tag{2.10}$$

which is a 3×4 matrix that completely specifies the idealized pin-hole projection for a projector. The projection matrix P can easily be found by mapping each pixel position and each surface point of already computed display surface.

To render an image onto a surface a texture map is created and projected onto the the polygonal model of the display surface. UV mapping is the process by which a two-dimensional texture is projected onto a three-dimensional model. UV mapping defines a coordinate system (u, v) for the texture and a coordinate system (x, y, z) for the 3D model. The mapping creates polygons in the (u, v) texture, and paints this portion of the image onto the associated polygon in (x, y, z) coordinates. This is done by assigning each vertex i in the 3D model with the associated values (u_i, v_i), and allowing the graphics engine, usually OpenGL, to decide how to render each polygon. After the UV mapping is complete, the textured model is then rendered from the projector's viewpoint using Equation (2.8), and displayed onto the object of interest. Figure 2.1b shows a 3D printed replica torso projected with a red and white checkerboard pattern rendered upon it using projection based AR. The checkerboard pattern itself is shown in Figure 2.1a.


(a) Checkerboard pattern

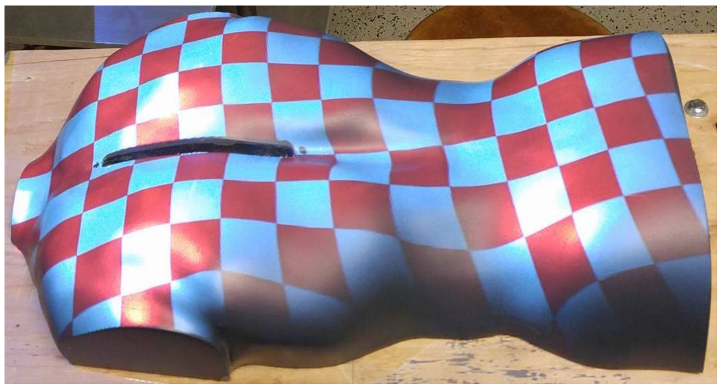

(b) SAR example of checkerboard rendered onto torso

Figure 2.1: Comparison of original checkerboard texture map to rendered projection onto a torso

Chapter 3

Derivation of Mass-Spring-Damper Model

In this chapter, the mass spring model will first be developed in state space form. The nonlinear state space form will be then be linearized. Methods for numerical integration and simplification of the model will be discussed for implementation into an estimation filter.

3.1 State Space Formulation

A well defined mass spring model for applications in cloth simulation is described by Provot in [73]. In this section, a realistic model of cloth is created by interconnecting a set of point masses (nodes) with spring and damper elements. Point masses are connected to each other via a combination of structural, shear, and diagonal elements. Structural elements connect point masses to their direct neighbors above, below, to the right, and to the left. They help constrain the mesh during compression or stretching. Additionally, shear elements connect masses which are diagonally adjacent to one another and help constrain the mesh when shear stresses are applied. Finally, flexion elements connect nodes that are two elements away and help constrain the mesh under bending. Figure 3.1 shows a 3×3 node cloth system with all elemental components connecting each node easily visible. The mass spring model for a cloth can be put into a state space representation by describing it using a first order differential equation of the form,

$$\dot{x} = f(x, t) \tag{3.1}$$

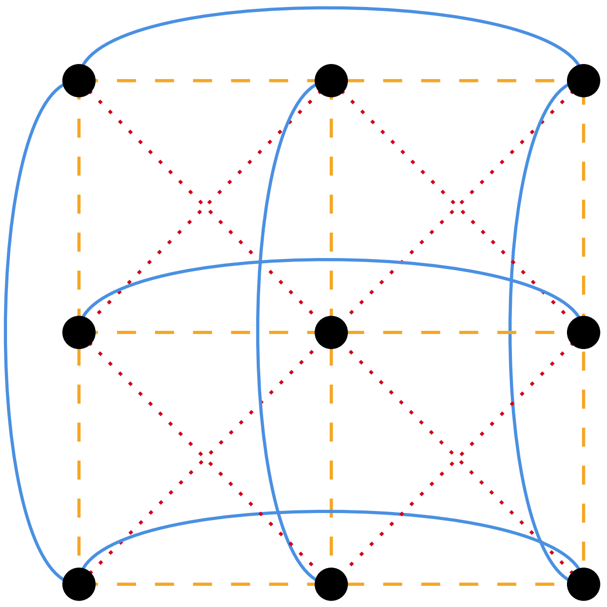


Figure 3.1: Connection of mass nodes with structural springs (orange dashed), shear springs (red dotted), and flexion springs (blue solid)

where, for n given nodes, $x \in \mathbb{R}^{6n}$ is the state vector containing the position $p_i \in \mathbb{R}^3$ and velocity $\dot{p_i} =: v_i \in \mathbb{R}^3$ for each node $i = 1, 2, \ldots, n$ in the space, i.e.,

$$x = [p_1^T p_2^T \ldots p_n^T v_1^T v_2^T \ldots v_n^T]^T \tag{3.2}$$

Therefore, the state vector can be rewritten as,

$$x = \begin{bmatrix} \mathbf{p} \\ \dot{\mathbf{p}} \end{bmatrix} \tag{3.3}$$

where $\mathbf{p} = [p_1^T \ldots p_n^T]^T$ and $\dot{\mathbf{p}} = [v_1^T \ldots v_n^T]^T$. $f(\cdot) : \mathbb{R}^{6n} \to \mathbb{R}^{6n}$ is a sufficiently smooth nonlinear state transition function, i.e., $f(\cdot) \in C^1$. The function $f(\cdot)$ is required to be differentiable since the Jacobian of the system is required for both numerical integration and the filter formulation. As the model is a time invariant system, the state transition function can be simplified to $f(x) \equiv f(x, t)$. Therefore, the state derivative vector can be written as

$$\dot{x} = f(x) = \begin{bmatrix} v_1 \\ \vdots \\ v_n \\ a_1(x) \\ \vdots \\ a_n(x) \end{bmatrix} \tag{3.4}$$

where $\dot{v_i} =: a_i(\cdot) : \mathbb{R}^{6n} \to \mathbb{R}^3$ represents the acceleration of each node. The resulting state derivative vector is composed of the point velocities concatenated with the point accelerations. These point accelerations can be derived using the dynamics of the system and are found to be

$$a_i(x) = \frac{1}{m_i}(m_i\mathbf{g} + \sum_{j\in\mathcal{A}_i} F_{ji}), i = 1\dots n \tag{3.5}$$

Here F_{ji} is the force applied onto point mass m_i by point mass m_j, and $\mathbf{g}$ is the acceleration due to gravity (i.e. $\mathbf{g} = \begin{bmatrix}0 & -9.81 & 0\end{bmatrix}^T$m/s). The set $\mathcal{A}_i$ is the set of all nodes which are connected to the node i by a spring and damper. Since each node is connected to another in the set $\mathcal{A}_i$, the internal force on a node i can be written in the form of a second order differential equation:

$$m_i\ddot{p}_i = -k_{s_i}p_i - k_{d_i}\dot{p}_i. \tag{3.6}$$

The force F_{ji}, caused by each node j on node i, can be found to be [68]:

$$F_{ji} = -k_{s_{ij}}(\|p_i - p_j\|_2 - r_{ij})\frac{p_i - p_j}{\|p_i - p_j\|_2} - k_{d_{ij}}(v_i - v_j) \tag{3.7}$$

the constants $k_{s_{ij}}$ and $k_{d_{ij}}$ are the spring and damper coefficients for a connection between nodes i and j, respectively. The spring component of the force is derived using Hooke's law, where the resting length of the spring between nodes i and j, $r_{ij} \in \mathbb{R}$, is used to counteract the effect of the spring and stop it from collapsing. A visual representation of Hooke's law can be seen in Figure 3.2. When the spring is stretched beyond its rest length it applies a force in the opposite direction, causing node i to move towards node j. Similarly, when the spring is compressed below its rest length it pushes node i away from node j. The damping force is always applied in the opposite direction of the velocity vector created between points i and j. Spring and damper coefficients are assumed to be linear constants for this model. More realistic models can be created with nonlinear spring and damper coefficients[31]; however, their characteristics are difficult to determine and could lead to significant computational delays. The set $\mathcal{A}_i$ can range in size from node to node. A node on the corner of a mesh has fewer connections, compared to a node on the edge of a mesh, compared to a node in the centre. A corner node, for example, will always have two fewer structural springs, three fewer shear springs and two fewer flexion springs than a center node. Thus, the size of the connected points set, $\mathcal{A}_i$, can range from 3 to 12 elements depending on the position of the node and the order the system.

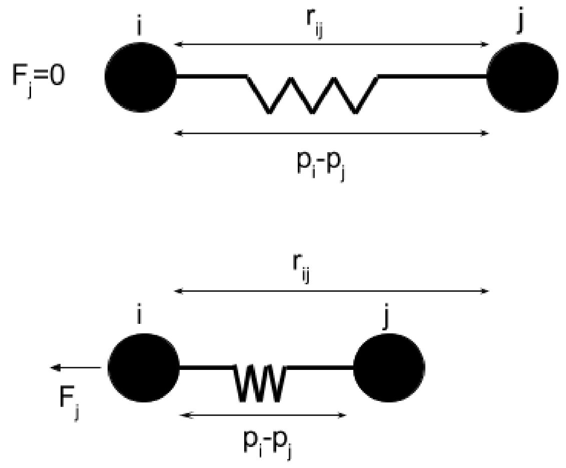


Figure 3.2: Illustration of spring force between two nodes i and j

3.2 Linearization

To be able to apply the system from Equation (3.1) into the extended Kalman filter based prediction algorithm it will first have to be linearized. The Jacobian of a vector function $\mathbf{f}(x_1, \dots, x_n)$ is

$$J = \frac{\partial f_i}{\partial x_j} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \cdots & \frac{\partial f_2}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial x_1} & \frac{\partial f_n}{\partial x_2} & \cdots & \frac{\partial f_n}{\partial x_n} \end{bmatrix} \tag{3.8}$$

For the model derived in Section 3.1, the Jacobian can be divided into four sub-matrices, i.e.,

$$J(x) = \begin{bmatrix} \frac{\partial \mathbf{v}}{\partial \mathbf{p}} & \frac{\partial \mathbf{v}}{\partial \mathbf{v}} \\ \frac{\partial \mathbf{a}}{\partial \mathbf{p}} & \frac{\partial \mathbf{a}}{\partial \mathbf{v}} \end{bmatrix} \tag{3.9}$$

where $\mathbf{p}$, $\mathbf{v}$ and $\mathbf{a}$ are $\mathbb{R}^{3n}$ vectors representing the positions, velocities and accelerations of all nodes respectively.

The upper left quadrant of $J(x)$ contains the derivatives of the velocity states v_i with respect to the position states p_j. These derivatives are 0, therefore, the upper left sub-matrix is a $3n \times 3n$ zeros matrix $\mathbf{0}_{3n\times 3n}$, i.e.,

$$\frac{dv_i}{dp_j} = \mathbf{0}_{3\times 3} \tag{3.10}$$

The upper right quadrant of $J(x)$ is made up of the derivatives of states v_i with respect to states v_j and is again trivially found. The result is a $3n \times 3n$ identity matrix, $\mathbf{I}_{3n\times 3n}$. In other words,

$$\frac{dv_i}{dv_j} = \begin{cases} \mathbf{I}_{3\times 3}, & i = j \\ \mathbf{0}_{3\times 3}, & i \neq j \end{cases} \tag{3.11}$$

The lower two sub-matrices contain the derivatives of the acceleration with respect to the state variables $\mathbf{p}$ and $\mathbf{v}$, and can be found using Equation (3.5). The lower right quadrant contains the derivatives of the accelerations a_i with respect to the velocity states v_k:

$$\begin{aligned} \frac{\partial a_i}{\partial v_k} &= -\frac{1}{m_i} \sum_{j\in A} \frac{\partial}{\partial v_k} k_d (v_i - v_j) \\ \frac{\partial a_i}{\partial v_k} &= -\frac{1}{m_i} \sum_{j\in A} \frac{\partial}{\partial v_k} k_d v_i - \frac{\partial}{\partial v_k} k_d v_j \end{aligned} \tag{3.12}$$

This derivative is not zero only for values of $k = j$, if $j \in \mathcal{A}$, and for $k = i$. Thus, the result is as follows,

$$\begin{aligned} \frac{\partial a_i}{\partial v_j} &= \frac{\partial}{\partial v_j} \frac{k_d}{m_i} v_j = \frac{k_d}{m_i} \mathbf{I}_{3\times 3} \\ \frac{\partial a_i}{\partial v_i} &= -\frac{1}{m_i} \sum_{j\in A} \frac{\partial}{\partial v_i} k_d v_i = -\frac{1}{m_i} \sum_{j\in A} k_d \mathbf{I}_{3\times 3} \end{aligned} \tag{3.13}$$

The lower right quadrant of the Jacobian contains the values $-\sum_{j\in A} \frac{k_d}{m_i}$ along the diagonal, and the sub-matrices $\frac{k_d}{m_i}\mathbf{I}_{3\times 3}$ sparsely placed based on spring connections.

Finally, the lower left quadrant of the Jacobian is derived from taking the derivatives of accelerations, a_i, with respect to the positions, p_k and results in the following:

$$\frac{\partial a_i}{\partial p_k} = -\frac{1}{m_i}\sum_{j\in A}\frac{\partial}{\partial p_k}[k_s(\|p_i - p_j\|_2 - r)\frac{p_i - p_j}{\|p_i - p_j\|_2}]. \tag{3.14}$$

Equation (3.14) can again be broken into two non-zero components; one when $k = j$, if $j \in \mathcal{A}$, and one when $k = i$. As seen in Gomes [41], this can be simplified using the product rule as follows, for $k = i$,

$$\frac{\partial a_i}{\partial p_i} = -\frac{k_s}{m_i}\sum_{j\in A}[(1 - \frac{r}{\|p_i - p_j\|})[\mathbf{I}_{3\times 3} - \frac{(p_i - p_j)(p_i - p_j)^T}{(p_i - p_j)^T(p_i - p_j)}] + \frac{(p_i - p_j)(p_i - p_j)^T}{(p_i - p_j)^T(p_i - p_j)}] \tag{3.15}$$

and similarly when $k = j$,

$$\frac{\partial a_i}{\partial p_j} = \frac{k_s}{m_i}[(1 - \frac{r}{\|p_i - p_j\|})[\mathbf{I}_{3\times 3} - \frac{(p_i - p_j)(p_i - p_j)^T}{(p_i - p_j)^T(p_i - p_j)}] + \frac{(p_i - p_j)(p_i - p_j)^T}{(p_i - p_j)^T(p_i - p_j)}]. \tag{3.16}$$

Altogether, the Jacobian can be written in the block matrix form of,

$$J(x) = \begin{bmatrix} \mathbf{0}_{3n\times 3n} & \mathbf{I}_{3n\times 3n} \\ \mathbf{J}_{ap} & \mathbf{J}_{av} \end{bmatrix} \tag{3.17}$$

where $\mathbf{J}_{ap} = \frac{\partial \mathbf{a}}{\partial \mathbf{p}}$ is composed of the matrices from Equations (3.15) and (3.16), $\mathbf{J}_{av} = \frac{\partial \mathbf{a}}{\partial \mathbf{v}}$ is composed of the matrices from Equation (3.13). By looking at Equation (3.17) it is important to note that 3 of the 4 components of the Jacobian $J(x)$ (i.e. $\mathbf{0}_{3n\times 3n}$, $\mathbf{I}_{3n\times 3n}$, and $\mathbf{J}_{av}$) are independent of the values of the state vector $\mathbf{x}$ and thus remain constant. Therefore, the only component that needs to be calculated is the $3n \times 3n$ sub-matrix $\mathbf{J}_{ap}$, significantly reducing the computation time required.

3.3 Integration Methods

Equation (3.4) provides an ordinary differential equation which needs to be solved in order to obtain the state vector $x(t)$. An approximate solution can be achieved by using a numerical integration algorithm. Several studies have investigated the convergence of numerical integration algorithms on mass spring cloth models [31]. Two broad categories for this type of integration are explicit and implicit methods. Explicit methods calculate the next state of the system using the current state while implicit methods solve a set of coupled equations involving both the current and future states of the system [68]. Explicit integration algorithms, such as the Euler forward method and Runge-Kutta method, are simple to implement and quick to run. However, they can be numerically unstable if the integration step time is chosen to be too large. Implicit integration algorithms, on the other hand, are often inherently numerically stable. This is because they solve a system of equations consisting on the current and next state of the system. The main drawback of this type of method is the increased computational complexity. However, because of the stability of the method, larger time steps can be used without the concern of unstable solutions. Additionally, implicit integration schemes allow for more robust parameter choices for deformable models. From [73], larger spring constants require smaller explicit integration step times for numerically stable solutions; implicit schemes however, tend to settle when using larger step sizes, making them suitable to more rigid mass spring models.

In this section the explicit fourth order Runge-Kutta method and the implicit Euler backwards method are compared to demonstrate the trade-offs between explicit and implicit integration methods.

Runge-Kutta Method

Given the system,

$$\dot{x} = f(t, x), \quad x(t_0) = x_0$$

Numerical integration methods use a given time step, ΔT,to discretely solve each integration step, $x[k]$, defined as follows,

$$x[k] = x(t_0 + k\Delta T) \qquad k = 0, 1, \dots \tag{3.18}$$

The fourth order Runge-Kutta solves the next integration step $x[k+1]$, given ΔT and the current solution $x[k]$, using the following steps [80]:

$$x[k+1] = x[k] + \frac{\Delta T}{6}(k_1 + 2k_2 + 2k_3 + k_4) \tag{3.19}$$

where,

$$\begin{aligned}
k_1 &= f(t_k, x[k]) \\
k_2 &= f(t_k + \frac{\Delta T}{2}, x[k] + \frac{\Delta T k_1}{2}) \\
k_3 &= f(t_k + \frac{\Delta T}{2}, x[k] + \frac{\Delta T k_2}{2}) \\
k_4 &= f(t_k + \Delta T, x[k] + \Delta T k_3)
\end{aligned}$$

This process is repeated as necessary to solve for x at any time step. The fourth order Runge-Kutta method is, as the name suggests, a fourth-order method. This means that the global truncation error, i.e. the total accumulated error over time, is on the order of $O(\Delta T^4)$. The small truncation error of this methods allows for high levels of accuracy, especially when using the small time steps required to maintain numerical stability.

Backward Euler Method

The backward Euler method is one of the most simple implicit numerical integration methods. Its unconditional stability allows for the use of larger time steps, which, in spite of its computational difficulties, often leads to faster run times compared to explicit methods [31]. The backward Euler method solves the following problem,

$$x[k+1] = x[k] + \Delta T f(x[k+1]) \tag{3.20}$$

For the system presented in Sections 3.1 and 3.2 this solution is as follows,

$$\Delta x = \begin{bmatrix} \Delta p \\ \Delta v \end{bmatrix} = \begin{bmatrix} p(t_k + \Delta T) - p(t_k) \\ v(t_k + \Delta T) - v(t_k) \end{bmatrix} \tag{3.21}$$

taking the first order approximation we get,

$$\begin{bmatrix} \Delta p \\ \Delta v \end{bmatrix} = \Delta T \begin{bmatrix} \Delta v + v_k \\ \Delta a + a_k \end{bmatrix} \tag{3.22}$$

Δa is the change in acceleration and is given by Equation (3.5). To aid in the solution of Δv a first order Taylor approximation is used on the nonlinear $a(x)$,

$$\Delta v \approx \Delta T(a(p_k, v_k) + \frac{\partial a}{\partial p}\Delta p + \frac{\partial a}{\partial v}\Delta v) \tag{3.23}$$

Substituting Δp from Equation (3.22) into Equation (3.23) and rearranging for Δv provides the following linear system,

$$(\mathbf{I} - \Delta T^2 \frac{\partial a}{\partial p} - \Delta T \frac{\partial a}{\partial v})\Delta v = \Delta T a(p_k, v_k) + \Delta T^2 \frac{\partial a}{\partial p} v_k \tag{3.24}$$

The partial derivatives $\frac{\partial a}{\partial p}$ and $\frac{\partial a}{\partial v}$ are found from the Jacobian derived in Section 3.2,

$$J(x) = \begin{bmatrix} \frac{\partial \mathbf{v}}{\partial \mathbf{p}} & \frac{\partial \mathbf{v}}{\partial \mathbf{v}} \\ \frac{\partial \mathbf{a}}{\partial \mathbf{p}} & \frac{\partial \mathbf{a}}{\partial \mathbf{v}} \end{bmatrix}.$$

Solving Equation (3.24) for Δv, and substituting into Equations (3.21) and (3.22), $v(t_k + \Delta T)$ and $p(t_k + \Delta T)$ can be found,

$$\begin{aligned} v(t_k + \Delta T) &= \Delta v + v_k \\ p(t_k + \Delta T) &= \Delta p + p_k = \Delta T(\Delta v + v_k) + p_k \end{aligned} \tag{3.25}$$

The backwards Euler method is an order one integration method. Therefore the global truncation error for this scheme is on the order of $O(\Delta T)$, which is lower than the fourth order Runge-Kutta, at $O(\Delta T^4)$. However, as the main desire of the integration scheme is overall computation time, and since the model used is already an approximation of the true system, the magnitude in error is outweighed by the benefit of the stability of the backward Euler method with larger step size. Additionally, the error value is further diminished by the use of a filtering algorithm, discussed in Chapter 4, which corrects for any accumulated error in the model with the use of measurements.

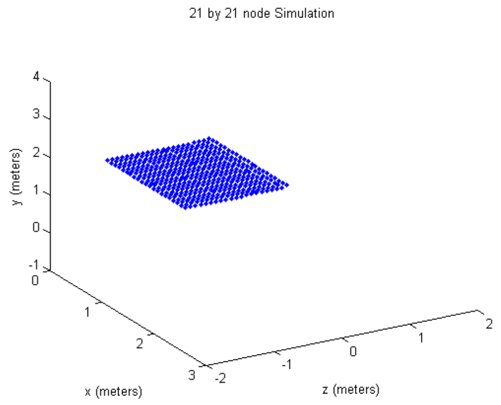
For a falling 21×21 node cloth model, like that shown in Figure 3.3, the Runge-Kutta method was required to run at a time step of $\Delta T = 0.001$ to maintain stability. The backward Euler method, on the other hand, was able to provide stability with a lower value of $\Delta T = 0.01$. The average relative difference between the two schemes is defined by the average distance between the nodes of each method,

$$e[k] = \frac{1}{n} \sum_{i=1}^{n} \|p_i^{rk}[k] - p_i^{eb}[k]\|_2, \quad k = 0, 1, \ldots \tag{3.26}$$

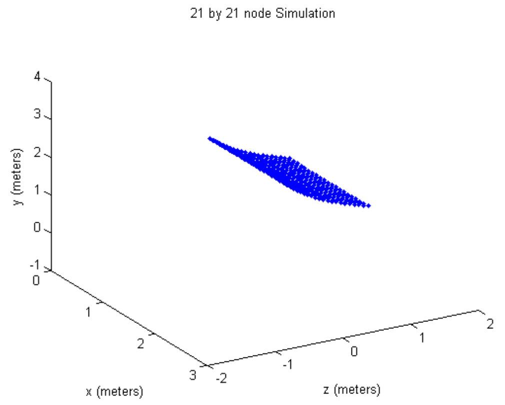
Here $p_i^{rk}[k]$ is the position of node i at time step k produced by the Runge-Kutta method, and $p_i^{eb}[k]$ is the position of the associated node produced by the Backwards Euler method. A comparison of these methods with Runge-Kutta using $\Delta T = 0.001$ and backward Euler using $\Delta T = 0.01$ is shown in Figure 3.4. This clearly shows that the model differences are relatively small, and that any large discrepancies will be able to be corrected using a filter that is run with an appropriately short step size.

3.4 Node Placement

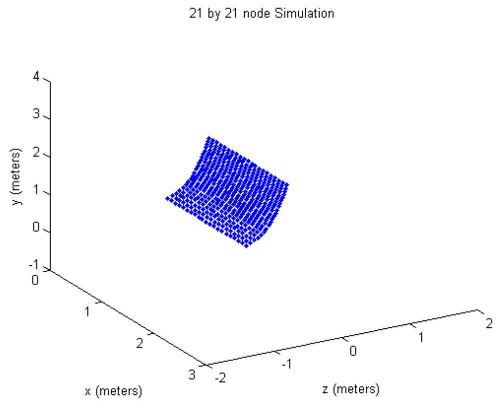
Since the mass spring model for a cloth, described in Section 3.1, is composed of discrete nodes connected by linear elements, it would be desirable to find node locations that will minimize the error between the surface created by the mesh and the true surface of the cloth. This can be achieved by first determining the dynamics of a cloth. These dynamics



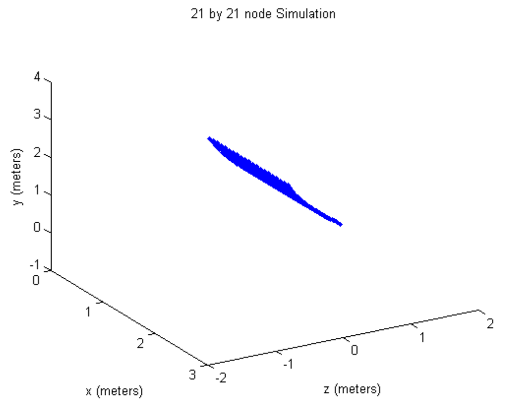
(a) Initial cloth position, $t = 0$s



(b) Intermediate cloth position, $t = 3.3$s



(c) Intermediate cloth position, $t = 6.6$s



(d) Intermediate cloth position, $t = 9.9$s

Figure 3.3: Simulation of a falling 21×21 node cloth mesh anchored at its top

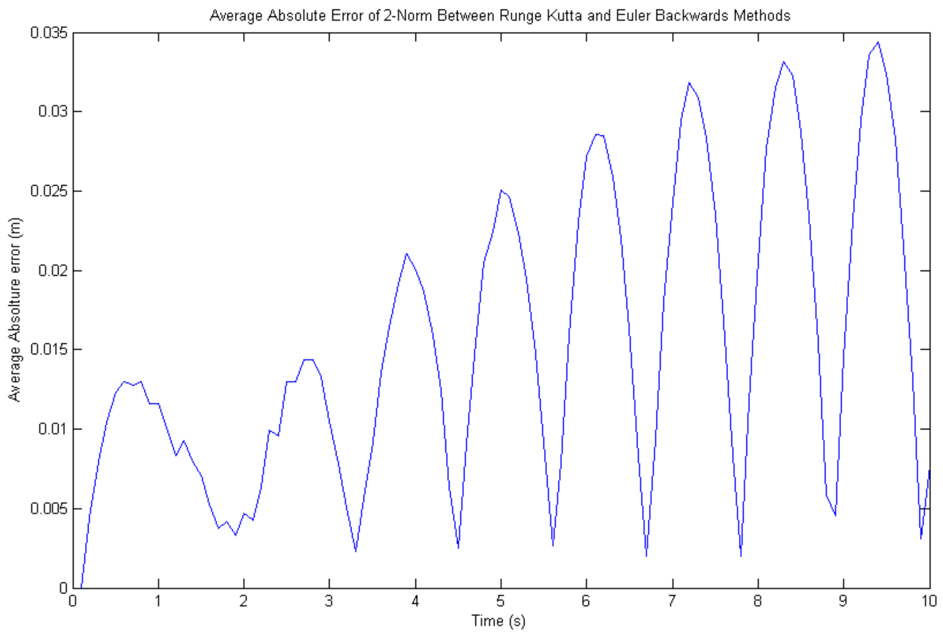


Figure 3.4: Comparison of Runge-Kutta at $\Delta T = 0.001$s and Euler Backwards Methods at $\Delta T = 0.01$s

can then be used to model its displacement modes. Once these modes are determined algorithm can be applied to minimize surface error due to node location.

A simple tool to analyse the deformations in thin plates is Kirchhoff-Love plate theory [62], which in itself is an extension of Euler-Bernoulli beam theory. For the purposes of simply evaluating the dynamics of a hanging cloth, Euler-Bernoulli beam theory is used. The theory is independently applied to the two different axis of the cloths planar surface in order to ease the analysis. The longitudinal (y) axis of the cloth is described by a cantilever (fixed-free) beam model; where as, the lateral (x) axis is described using a free-free beam.

In Euler-Bernoulli beam theory, the beam equation for free vibrations is defined as [26, 89],

$$EI\frac{d^4z}{dx^4} + \mu\frac{d^2z}{dt^2} = 0 \tag{3.27}$$

where E is elastic modulus, I is the second moment of area of the cross section, and μ is the mass per unit length. The general solution of Equation (3.27) for the displacements is,

$$Z_n(x) = A_1\cosh(k_nx) + A_2\sinh(k_nx) + A_3\cos(k_nx) + A_4\sin(k_nx) \tag{3.28}$$

Here Z_n is the displacement and k_n is a frequency constant, both dependant on the boundary conditions of the system and its nth natural frequency ω_n. Analysis of the natural modes of the beam analogue provides the fundamental components of its displacement shape. The most general motion of a system is a superposition of its modes [69]. For a cantilevered system with length L the boundary conditions are as follows,

$$Z_n(0) = 0 \ , \quad \frac{dZ_n(0)}{dx} = 0 \ , \quad \frac{d^2 Z_n(L)}{dx^2} = 0 \ , \quad \frac{d^3 Z_n(L)}{dx^3} = 0 \tag{3.29}$$

Applying this to Equation (3.28), and solving for nontrivial solutions, yields,

$$Z_n(x) = (\cosh(k_n x) - \cos(k_n x)) + \frac{\cos(k_n L) + \cosh(k_n L)}{\sin(k_n L) + \sinh(k_n L)} (\sin(k_n x) - \sinh(k_n x)) \tag{3.30}$$

with the following frequency values found in Table 3.1. Similarly, for a free-free system the initial conditions are,

$$\frac{d^2 Z_n(0)}{dx^2} = 0 \ , \quad \frac{d^3 Z_n(0)}{dx^3} = 0 \ , \quad \frac{d^2 Z_n(L)}{dx^2} = 0 \ , \quad \frac{d^3 Z_n(L)}{dx^3} = 0 \tag{3.31}$$

with solution,

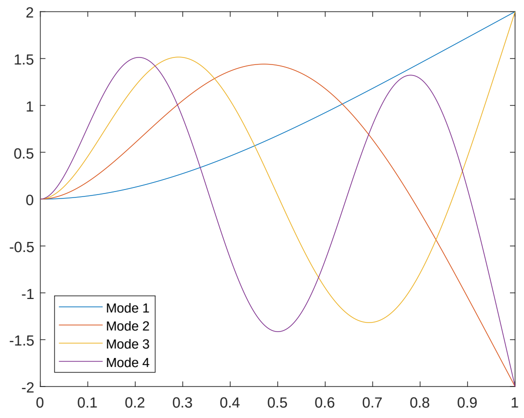
$$Z_n(x) = (\sin(k_n x) + \sinh(k_n x)) + \frac{\sin(k_n L) - \sinh(k_n L)}{\cosh(k_n L) - \cos(k_n L)} (\cos(k_n x) + \cosh(k_n x)) \tag{3.32}$$

again with frequencies found in Table 3.1. Figure 3.5 shows the first four modes of both a fixed-free and free-free beam with length $L = 1$. These mode shape functions can now be

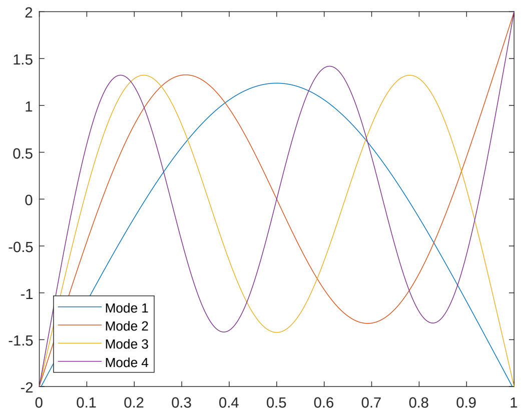
Table 3.1: Free vibration frequency values

	Fixed-Free	Free-Free
Mode (n)	$k_n L$	$k_n L$
1	1.8751	4.7300
2	4.6941	7.8532
3	7.8548	10.9956
4	10.9955	14.1372

used in conjunction with an optimization algorithm to find node locations that minimizes the distance between the flat surfaces, created by the triangle mesh, and the true surface.



(a) Fixed-free mode shapes



(b) Free-free mode shapes

Figure 3.5: First four mode shapes of a fixed-free and a free-free beam

A least-squares solver is used to solve problem in Equation (3.33), where F is a nonlinear function dependant on coefficients $\mathbf{p}$.

$$\min_{\mathbf{p}} \|F(\mathbf{p}, x) - Z_n(x)\|_2^2 \tag{3.33}$$

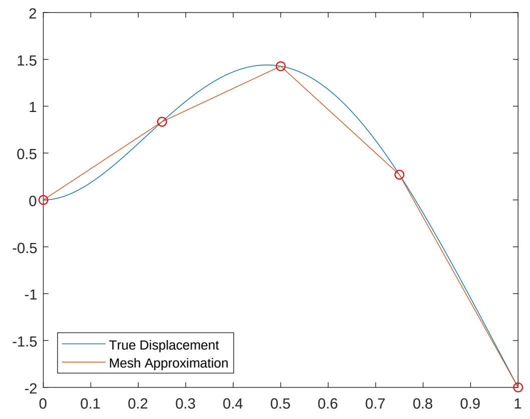
In the case of the model defined in Section 3.1, this function is defined by a set of nodes that lie on the surface of the cloth and are connected by straight edges. The function takes the form,

$$F = \begin{cases} f_1 = Z_n(p_0) + \frac{Z_n(p_1)-Z_n(p_0)}{p_1-p_0}(x - p_0), & p_0 <= x <= p_1 \\ f_2 = f_1(p_1) + \frac{Z_n(p_2)-Z_n(p_1)}{p_2-p_1}(x - p_1), & p_1 <= x <= p_2 \\ \vdots & \\ f_l = f_{r-1}(p_{r-1}) + \frac{Z_n(p_r)-Z_n(p_{r-1})}{p_r-p_{r-1}}(x - p_{r-1}), & p_{r-1} <= x <= p_r \end{cases} \tag{3.34}$$

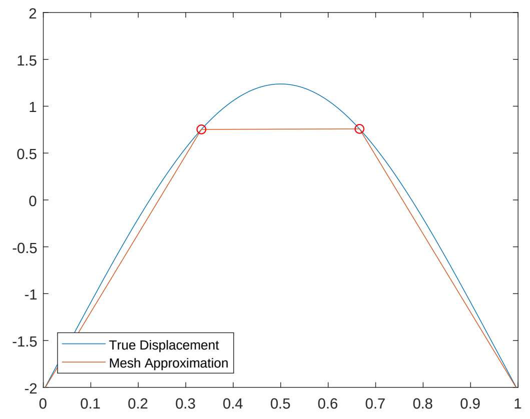
where, given r nodes, p_i is the location of the ith node, for $i = 0, 1, \dots r$.

By observation of a falling cloth, Figure 3.3, and the beam modes, Figure 3.5, it is clear that the dominant behaviour in the longitudinal direction is best described by the second mode of a fixed-free beam, while the behaviour in the lateral direction can be described by the first mode of a free-free beam. By using Equations (3.30, 3.32, 3.34) to solve for Equation (3.33) a set of optimal node points can be generated. An example of this optimization can be seen in Figure 3.6, with $L = 1$, $p_0 = 0$, $p_r = L$, and $r = 5$ and $r = 4$

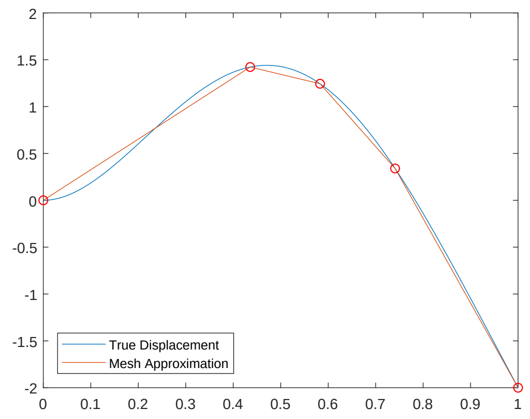
for the longitudinal and lateral directions respectively. Here, it is clear that the optimal node locations, produced by the algorithm, more closely match the object surface than an evenly distributed set of nodes.



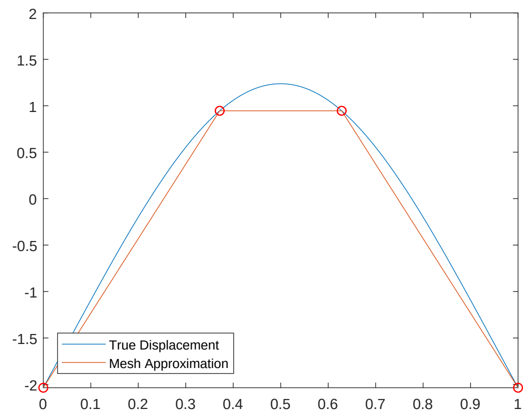
(a) Evenly distributed node locations of a fixed-free beam in its second mode



(b) Evenly distributed node locations of a free-free beam in its first mode



(c) Optimal node locations of a fixed-free beam in its second mode



(d) Optimal node locations of a free-free beam in its first mode

Figure 3.6: Evenly distributed and optimal node locations of a fixed-free and a free-free beam, with 5 and 4 nodes respectively

3.5 Model Compression

In order for the model, derived in Section 3.1, to be incorporated into a real-time filtering algorithm, it will require a computationally quick solution. To achieve this goal, the number of nodes used in the system will have to be reduced. Although modelling the system with a large number of nodes provides greater accuracy, the need for timely computation is essential for real-time applications. To reduce the number of nodes in the model, optimal spring, damper and mass parameters can be found to closely match the behaviour of a sparse desired model to a dense actual model.

For a mass spring cloth model Gomes [41] suggests an optimization algorithm that finds a parameter vector θ which minimizes the following cost function,

$$\begin{aligned} C(\theta) = \quad & \sum_{i=0}^{N} e^{\alpha i \Delta T} \| Hz[i] - \mathbf{A} Hx[i] \|_2^2 \\ \text{s.t.} \quad & z[i] = f(z[i-1], \theta) \end{aligned} \tag{3.35}$$

Here $x \in \mathbb{R}^m$ is the state vector of the higher order system, $z \in \mathbb{R}^n$, is the state vector of the lower order system, with $m > n$. Both $x[i]$ and $z[i]$ are solved by numerically integrating the state space formulation from Equation (3.1). H is the observation model, $\mathbf{A}$ is a $\mathbb{R}^{n \times m}$ matrix that selects the states of the higher order system to compare to the lower order system. $e^{\alpha i \Delta T}$ is used to ensure that the error in the later part of the simulation is at a higher cost, therefore causing the sparser model to match the higher density one for a longer time. Here α is a scaling factor for the future time weight, and ΔT is the integration time used to solve for both $x[i]$ and $z[i]$. The parameter vector θ is a $\mathbb{R}^{2s+n}$ vector, where s is the number of spring connections in the reduced model. It is made up of s spring constants k_{s_i}, s damper constants k_{d_i}, and n masses m_i. It is written as follows,

$$\theta = \begin{bmatrix} k_{s_1} \\ \vdots \\ k_{s_s} \\ k_{d_1} \\ \vdots \\ k_{d_s} \\ m_1 \\ \vdots \\ m_n \end{bmatrix} \tag{3.36}$$

The cost function in Equation (3.35) is minimized using a heuristic algorithm called simulated annealing. Simulated annealing is chosen over other heuristic methods, such as genetic and evolutionary algorithms [17, 61], due to its relatively low computation time. Other algorithms require large sets of solutions to be pre-processed, and so have a significant computational load. Simulated annealing on the other hand only finds one, usually more optimal, solution per iteration [33].

The simulated annealing algorithm first takes an initial guess of the parameter vector and considers it the current optimal solution,

$$\theta_{op} = \theta_0 \tag{3.37}$$

Next, a zero-mean Gaussian noise, W, is added to the previous cost to create a new candidate solution,

$$\theta_c = \theta_{op} + W \tag{3.38}$$

The cost function $C(\theta)$, in this case given by Equation (3.35), is then used to evaluate the validity of the new parameter. If the cost of the candidate parameter is lower than that of the old parameter

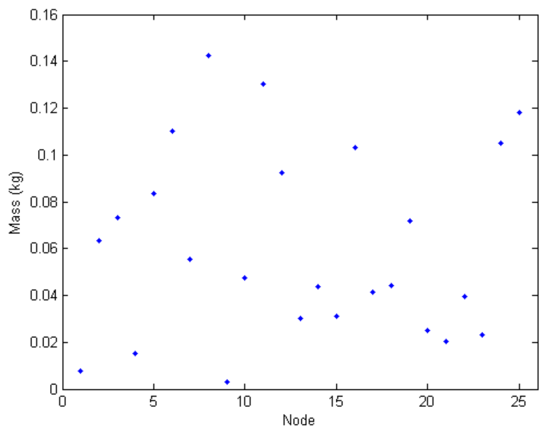
$$C(\theta_c) \leq C(\theta_{op}) \tag{3.39}$$

it may be assigned as the new optimal parameter with probability P,

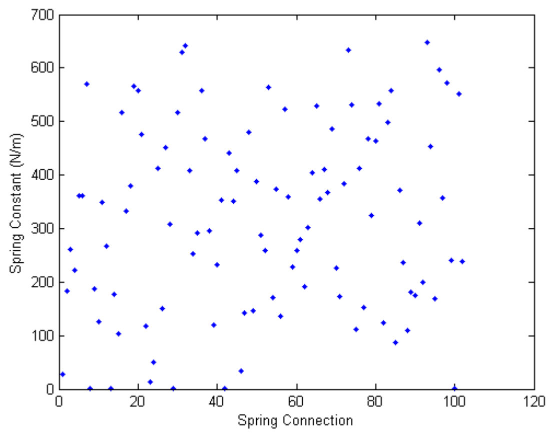
$$P = e^{\frac{-\Delta C}{T}}$$

where ΔC is the difference in cost between the new solution and the old solution, i.e. $\Delta C = C(\theta_1) - C(\theta_0)$, and T is a temperature term. T is initially set based on how many iterations of the algorithm the programmer requires. After a certain number of iterations, the temperature term T decreases in value, which causes the probability of choosing a bad solution to decrease. This process is repeated until a termination condition, such as a minimum value for T, is reached. The final solution, θ_N, at the point of termination should produce a result that is close to the global minimum, however there is no guarantee. The longer the algorithm is run for, the more likely the solution θ_N will converge to the global minimum. The choice of initial temperature and rate of cooling have some effect on the final solution, as they directly contribute to the number of iterations and the probability of choosing a suboptimal solution.

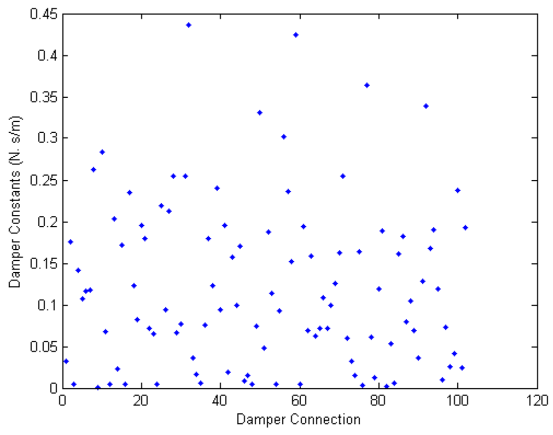
This solution can now be evaluated on a higher order system to verify that a reduced order model would still provide satisfactory results. To test this a 21×21 high order model, representing a $1.5m \times 1.5m$ cloth, is reduced down to a 5×5 sparse model. The initial parameter vector θ_0 is set such that $k_{s_i} = 300$, $k_{d_i} = 0.08$, and $m_1 = 0.025$ for all i., and $\alpha = 0.001$ Figure 3.7 shows the resulting parameter values of the compressed system and the reduced cost of the optimal parameter set over the original values. It is clear that the overall cost of the final chosen parameter vector, θ_N is much smaller than that of the original parameters set. Finally, Figure 3.8 shows that the mean squared error of the reduced order system is significantly low enough that it can be used as a valid alternative to the full system. Additionally, this method can be used to find the parameters of a real life system being estimated by the prediction filters discussed in Chapter 4. The measured values of the real life system, can be used to train the parameters of a model, and will be used in Chapter 5



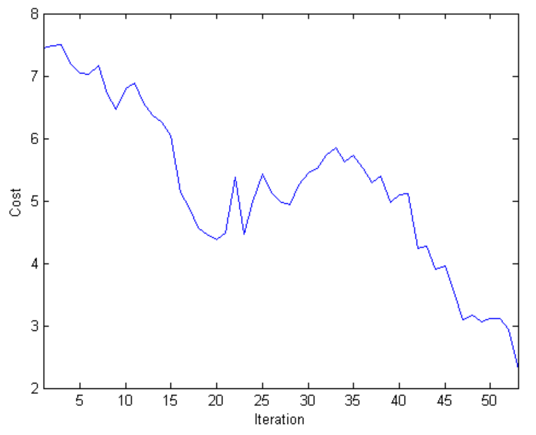
(a) Mass parameters



(b) Spring parameters



(c) Damper parameters



(d) Costs for compression

Figure 3.7: Final parameter vector θ_N values for 5×5 compression

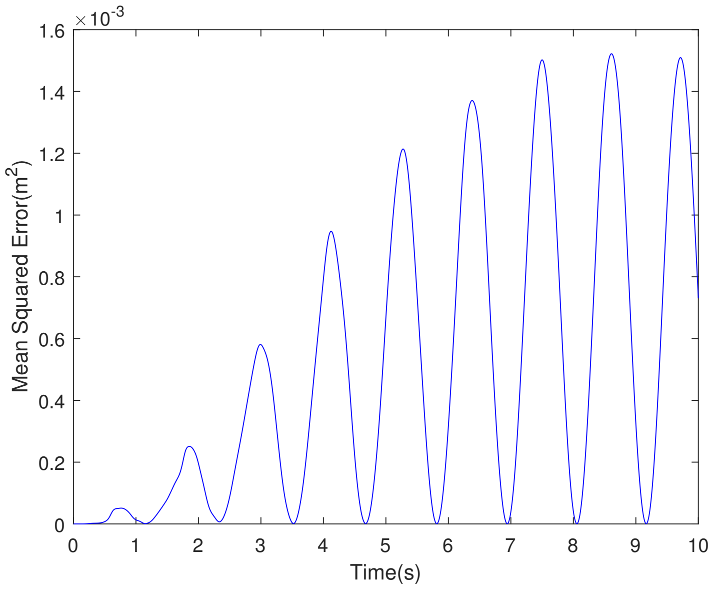


Figure 3.8: Mean squared error between 5x5 compressed model and the original model

Chapter 4

Review of Filtering Approaches

Due to the approximations made by representing a real life system with a mass spring model, along with the inherent sensor noise of measured positions on a time-varying system, a filtering technique is required to obtain an accurate prediction of the system's states. Additionally, as the only states that are measurable in the dynamic model, presented in Section 3.1, are the positions, the model velocities need to be estimated. A filtering algorithm allows for this estimation to occur with minimal error. This is achieved by minimizing the error between the actual state of the system and the estimated state. This chapter discusses several filtering techniques, mainly revolved around the least squares estimate, such as the Kalman filter, the extended Kalman filter, and the cubature Kalman filter.

4.1 Least Squares Estimation

A common form of this type of filtering is the linear least squares method. This method minimizes squared error of the estimate,

$$\arg\min_{\hat{x}} \|x - \hat{x}\|^2 \tag{4.1}$$

where $\hat{x}$ is the state estimate and x is the actual state value. The least squares method has been applied to numerous application areas for both state estimation and parameter identification. The problem is defined as follows [58], given a sequence of n noisy measurements $\{y_i\}_{i=1}^{n}$ and random variable Z, the true state value, the least squares estimate $\hat{Z}$ is the solution to the minimization problem

$$\hat{Z} = f(y_1, y_2, \ldots, y_n) = \arg\min_{g} \{\mathbb{E}[(Z - g(y_1, y_2, \ldots, y_n))^2] |\ g : \mathbb{R}^n \to \mathbb{R}\} \tag{4.2}$$

where $\mathbb{E}[\cdot]$ is the expected value function, and $g(\cdot)$ is a function applied to the set of noisy measurement data. When Z and $\{y_i\}_{i=1}^n$ are all normally distributed the solution to this problem becomes,

$$\hat{Z} = \mathbb{E}[Z|y_1, y_2, \ldots, y_n] = \sum_{i=1}^{n} \alpha_i y_i + \beta_i, \quad \alpha_i, \beta_i \in \mathbb{R} \tag{4.3}$$

This chapter will review the Kalman Filter, the extended Kalman filter, the unscented Kalman filter, and the cubature Kalman filter. All of these techniques use the linear least squares estimate to find the states of a dynamic system of the form

$$\begin{aligned} \dot{x} &= f(x) \\ y &= h(x) \end{aligned} \tag{4.4}$$

where x is the state vector, $f(\cdot)$ is the state transition function, y is the output vector, and $h(\cdot)$ is the observation model.

4.2 Kalman Filter

The Kalman filter is a recursive filter that estimates the internal state of a linear dynamic system from a series of noisy measurements. It was developed by Kalman [51], and is able to produce an optimal state estimate given a linear system with only independant Gaussian additive noise. Though optimal results are only guaranteed for Gaussian noise, it is not a requirement for the Kalman filter to be able to produce convergent results. Thus, it is used in a wide variety of applications, including GPS tracking, signal processing, and motion planning [38, 43].

To apply the Kalman filter a linear dynamic system must first be defined,

$$\begin{aligned} x[k+1] &= F_k x[k] + w_k \\ y[k] &= H_k x[k] + v_k \end{aligned} \tag{4.5}$$

where $F_k \in \mathbb{R}^{n \times n}$ is the state transition matrix, $H_k \in \mathbb{R}^{n \times n}$ is the observation matrix, $x[k] \in \mathbb{R}^n$ is the state vector, $y[k] \in \mathbb{R}^m$ is the output vector, $w_k \in \mathbb{R}^n$ is the process noise vector, and $v_k \in \mathbb{R}^m$ is the measurement noise vector, all at time step k. The noise vectors w_k and v_k are assumed to be Gaussian, independent random variables with covariances of Q_k and R_k respectively, that is:

$$\begin{aligned} \mathbb{E}[w_k w_k^T] &= Q_k \\ \mathbb{E}[v_k v_k^T] &= R_k \\ \mathbb{E}[w_k v_k^T] &= 0. \end{aligned} \tag{4.6}$$

By applying the least-squares estimator the state estimate vector $\hat{x}_{k|k}$ at time step k can be found, as described in Equation (4.3). The Kalman filter algorithm can be broken down into two steps, a prediction step and an update step. The prediction step proceeds as follows,

$$\begin{aligned} \hat{x}_{k|k-1} &= F_k \hat{x}_{k-1|k-1} \\ P_{k|k-1} &= F_k P_{k-1|k-1} F_k^T + Q_k \end{aligned} \tag{4.7}$$

Here, $\hat{x}_{k|k-1}$ is the state prediction given the previous $k-1$ measurements and $P_{k|k-1}$ is the state covariance matrix given the previous $k-1$ measurements. In this step the previous state estimate is used to produce an estimate of the state at the current time step. This predicted state is known as the *a priori* state estimate as it does not contain any information about the observations at the current time step. Following the predict step, the Kalman update step, shown in Equation (4.8), can be carried out.

$$\begin{aligned} \tilde{y}_k &= y[k] - H_k \hat{x}_{k|k-1} \\ S_k &= H_k P_{k|k-1} H_k^T + R_k \\ K_k &= P_{k|k-1} H_k^T S_k^{-1} \\ \hat{x}_{k|k} &= \hat{x}_{k|k-1} + K_k \tilde{y}_k \\ \Sigma_{k|k} &= (I - K_k H_k) P_{k|k-1} \end{aligned} \tag{4.8}$$

Here $\tilde{y}_k$ is the error between the measured output and the predicted output, also called the innovation residual, S_k is the innovation covariance, K_k is the Kalman gain, $\hat{x}_{k|k}$ is the updated state estimate and $P_{k|k}$ is the updated estimate covariance, all at time step k. The update step improves the *a priori* state estimate by combining it with current observation information and refining the estimate. The new estimate is called the *a posteriori* state estimate.

Typical behaviour for the Kalman filter is to execute prediction and update steps alternatively. However, this doesn't always have to be the case. In the situation whereby there is a delay in measurements, the predict step may be run multiple times until a measurement is received [52]. An important aspect of the Kalman filter is the choice of initial conditions, $\hat{x}_{0|0}$ and $P_{0|0}$, as well as noise covariance matrices Q_k and R_k. The selection of these values can are often difficult to find and can greatly affect the performance of the Kalman filter. These parameters can be estimated using a priori statistical knowledge about the system [7, 76]; however, often times, they are simply tuned until the best results are observed. A severe drawback of the Kalman filter is its limitation of only being applicable to linear systems. Therefore, the algorithm will need to be adapted to be applied to nonlinear systems.

4.3 Extended Kalman Filter

The extended Kalman filter (EKF) is an implementation of the Kalman filter, designed for use on non-linear systems. Though it is no longer optimal, it is very useful as it enables the estimation of state variables through linear approximation. The EKF is very similar to the regular Kalman filter, with the caveat of linearizing the nonlinear system dynamics at every time step [9].

For applications of the EKF a nonlinear dynamic system of the following form is given,

$$\begin{aligned} x[k+1] &= f(x[k]) + w_k \\ y[k] &= h(x[k]) + v_k \end{aligned} \tag{4.9}$$

where $f : \mathbb{R}^n \to \mathbb{R}^n$ and $h : \mathbb{R}^n \to \mathbb{R}^m$ are sufficiently smooth functions, $x[k]$ is the state vector, w_k is the process noise, $y[k]$ is the output, and v_k is the measurement noise, all at time step k. To be able to adapt this nonlinear to system to use with the standard Kalman filter, the functions f and h need to be linearized. The Jacobian of f and h are evaluated at every time step to arrive at the linearized system, assumed to be smooth, and as a result, differentiable, the Jacobian of f and h can be evaluated at each time step k. Equation (4.10) lists the Jacobians of f and h, which are used in place of the state transition matrix F_k and observation matrix H_k in the standard Kalman filter, respectively.

$$\begin{aligned} F_{k-1} &= \frac{\partial f}{\partial x}\Big|_{\hat{x}_{k-1|k-1}} \\ H_k &= \frac{\partial h}{\partial x}\Big|_{\hat{x}_{k|k-1}}. \end{aligned} \tag{4.10}$$

The EKF now recursively estimates the states of the system in a manner very similar to the Kalman filter. The algorithm can again be broken down into two steps the predict step and the update step. The EKF prediction step is as follows,

$$\begin{aligned} \hat{x}_{k|k-1} &= f(\hat{x}_{k-1|k-1}) \\ P_{k|k-1} &= F_{k-1} P_{k-1|k-1} F_{k-1}^T + Q_{k-1} \end{aligned} \tag{4.11}$$

Next the update step of the state prediction is computed,

$$\begin{aligned} \tilde{y}_k &= y_k - h(\hat{x}_{k|k-1}) \\ S_k &= H_k P_{k|k-1} H_k^T + R_k \\ K_k &= P_{k|k-1} H_k^T S_k^{-1} \\ \hat{x}_{k|k} &= \hat{x}_{k|k-1} + K_k \tilde{y}_k \\ P_{k|k} &= (\mathbf{I} - K_k H_k) P_{k|k-1} \end{aligned} \tag{4.12}$$

An important piece of information required for the EKF are the initial state estimate $\hat{x}_{0|0}$ and the initial covariance matrix $P_{0|0}$. As the system involved in the EKF is a first order approximation of the nonlinear system, an incorrect initial value could lead to divergence; therefore, it is important to choose these values carefully.

4.4 Unscented Kalman Filter

The unscented Kalman filter (UKF)[86] addresses the main issues of the EKF, such as approximating the dynamics of a given system using a first order approximation, as well as using only the mean value of the prior estimate, propagated through a nonlinear function, to calculate the posterior mean and covariance. In contrast, the UKF provides a third order approximation, when the noise variables are Gaussian distributed, and a second order approximation, if not, of the estimate. It does this by utilizing several carefully chosen sample points, called sigma points, to help capture the true mean and covariance of the transformed random variables.

The UKF algorithm begins by choosing a set of $2n + 1$ sigma points, where n is the number of states, based on the state estimate, $\hat{x}_{k-1|k-1}$, and the state covariance matrix, $\mathbf{P}_{k-1|k-1}$ [81],

$$\begin{aligned} \chi^0_{k-1|k-1} &= \hat{x}_{k-1|k-1} \\ \chi^i_{k-1|k-1} &= \hat{x}_{k-1|k-1} + (\sqrt{(n+\lambda)\mathbf{P}_{k-1|k-1}})_i, \quad i = 1, \ldots, n \\ \chi^i_{k-1|k-1} &= \hat{x}_{k-1|k-1} - (\sqrt{(n+\lambda)\mathbf{P}_{k-1|k-1}})_{i-n}, \quad i = n+1, \ldots, 2n \end{aligned} \tag{4.13}$$

where $(\sqrt{(n+\lambda)\mathbf{P}_{k-1|k-1}})_i$ is the ith column of the matrix $\sqrt{(n+\lambda)\mathbf{P}_{k-1|k-1}}$. A matrix $\mathbf{B}$ is the square root of a matrix $\mathbf{A}$ if $\mathbf{B}\mathbf{B}^T = \mathbf{A}$. Each χ^i has a corresponding weight used to calculate the state and covariance values; they are defined as

$$\begin{aligned} W_s^0 &= \frac{\lambda}{n+\lambda} \\ W_c^0 &= \frac{\lambda}{n+\lambda} + 1 - \alpha^2 + \beta \\ W_s^i = W_c^i &= \frac{1}{2(n+\lambda)} \\ \lambda &= \alpha^2(n+\kappa) - n \end{aligned} \tag{4.14}$$

where λ is a scaling parameter. α determines the spread of the sigma points around $\hat{x}_{k-1|k-1}$, κ is a secondary scaling parameter and β is used to incorporate prior knowledge of the distribution of $\hat{x}_{k-1|k-1}$.

For the prediction step of the UKF algorithm, the sigma points are first passed through the nonlinear state transition function $f(\cdot)$ from Equation (4.9), such that

$$\chi^i_{k|k-1} = f(\chi^i_{k-1|k-1}) \tag{4.15}$$

These transformed sigma points are recombined to produce the predicted state and covariance matrix,

$$\begin{aligned} \hat{x}_{k|k-1} &= \sum_{i=0}^{2n} W_s^i \chi^i_{k|k-1} \\ P_{k|k-1} &= \sum_{i=0}^{2n} W_c^i (\chi^i_{k|k-1} - \hat{x}_{k|k-1})(\chi^i_{k|k-1} - \hat{x}_{k|k-1})^T + Q_k \end{aligned} \tag{4.16}$$

After this, new sigma points $\chi^i_{k|k-1}$ are found by repeating Equation (4.13) with $\hat{x}_{k-1|k-1}$ and $\mathbf{P}_{k-1|k-1}$ replaced $\hat{x}_{k|k-1}$ and $\mathbf{P}_{k|k-1}$ respectively. These sigma points are then passed through the observation model $h(\cdot)$,

$$\gamma^i_k = h(\chi^i_{k|k-1}) \tag{4.17}$$

The update step is then executed,

$$\begin{aligned}
\hat{y} &= \sum_{i=0}^{2n} W_s^i \gamma_k^i \\
P_{y_k y_k} &= \sum_{i=0}^{2n} W_c^i (\gamma_k^i - \hat{y}_k)(\gamma_k^i - \hat{y}_k)^T + R_k \\
P_{x_k y_k} &= \sum_{i=0}^{2n} W_c^i (\gamma_k^i - \hat{x}_{k|k-1})(\gamma_k^i - \hat{y}_k)^T \\
K_k &= P_{x_k y_k} P_{y_k y_k}^{-1} \\
\hat{x}_{k|k} &= \hat{x}_{k|k-1} + K_k (y_k - \hat{y}_k) \\
P_{k|k} &= P_{k|k-1} - K_k P_{y_k y_k} K_k^T
\end{aligned} \tag{4.18}$$

As can be seen, the UKF does not require the use of the Jacobian of the system, and so can be used in systems that are non-differentiable or difficult to differentiate. The effectiveness of the UKF can be significantly affected by the choice of α, κ, and β, which should be carefully chosen in order to get good results. Figure 4.1 provides a simple illustration of how the sigma points of the UKF allows it to be a more accurate prediction algorithm than the EKF. Using multiple, strategically chosen, sample points more accurately capture the mean and covariance of a non-linearly transformed system, short of using a true random sampling technique with a significantly higher computation time.

4.5 Cubature Kalman Filter

The main drawback of the UKF is choosing the values of α, κ, and β, in Equation (4.14), such that the filter is accurate. One way of obtaining these values is by using a set of rules, called cubature rules, as is done in the Cubature Kalman filter (CKF). The CKF was developed as a more accurate way of predicting the states of a nonlinear system by take advantage of the "nice" properties of Gaussian distributions. If it is assumed that the nonlinear system described in Equation (4.9) is modelled with error that is normally distributed, these properties can be exploited. It is well known that the best state prediction, $\hat{x}_{k|k-1}$, of a dynamic system is given by a conditional expectation [10]. Furthermore, when the source of noise in the model is normally distributed, the state prediction can be written as an integral of the nonlinear model $f(x)$ and of the Gaussian density function.

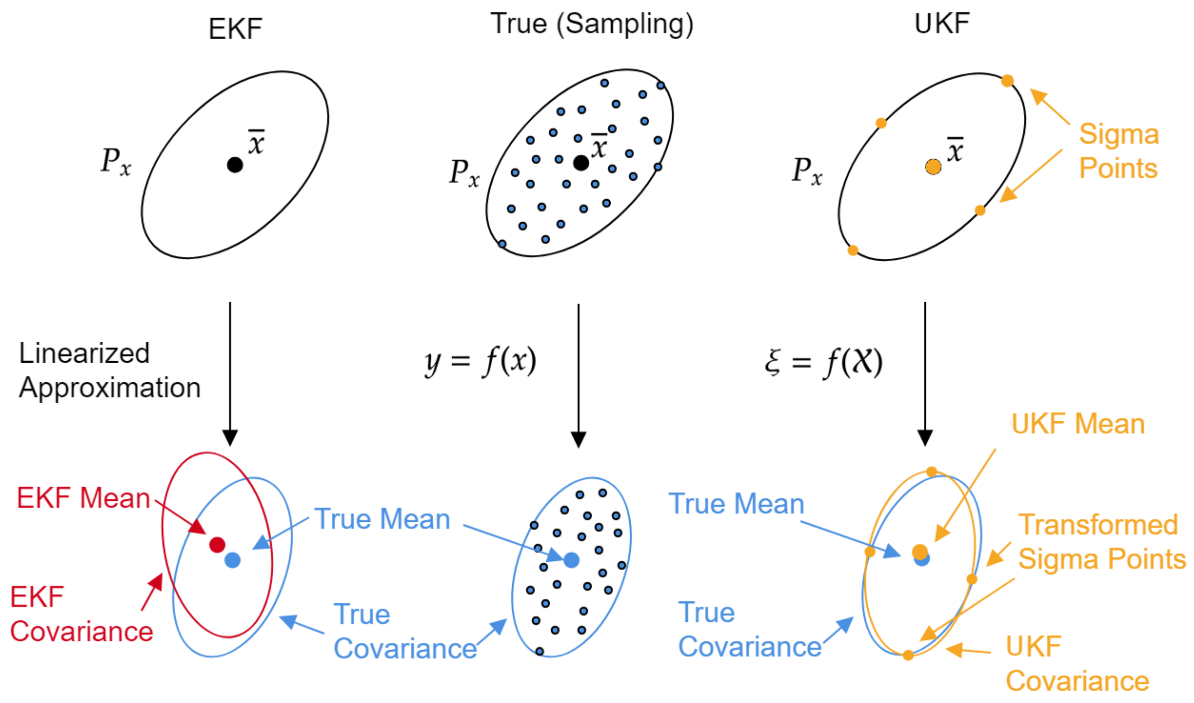


Figure 4.1: Visualisation of the EKF and UKF algorithms propagation accuracy when compared to a true sampling technique

Since this integral may be difficult to solve, it can be approximated as

$$\hat{x}_{k|k-1} = \int f(x)\mathcal{N}(x_{k-1}; \hat{x}_{k-1|k-1}, \mathbf{P}_{k-1|k-1})dx_{k-1} \approx \sum_{i=1}^{m} \omega_i f(\zeta_i) \tag{4.19}$$

where ω_i are weights and ζ_i are sample points which are chosen in a specific manner. This method of approximation is called a Gaussian quadrature rule for solving the conditional expectation of a normally distributed random variable. A Gaussian quadrature rule allows for a computationally less intensive approximation of the conditional expectation. The difficult part of finding an accurate approximation, as given in Equation (4.19), is solving for the weights ω_i and sample points ζ_i. There may be many combinations weights and sample points that give a an accurate approximation for the state prediction, such as the UKF. The CKF provides a set of weights and sample points that provide the closest know approximate based on a normally distributed noise [10]. In fact, the CKF provides a solution for the state prediction that is more accurate (third order approximation of the true solution) than the solution given by the EKF (first order approximation). The CKF uses what are called cubature rules to solve for the weights and sample points. The weights and sample points given by the cubature rules are found to be the same as the UKF, with $\alpha = 1$, $\kappa = 0$, and $\beta = 0$, therefore:

$$\begin{aligned}
&\chi^0_{k-1|k-1} = \hat{x}_{k-1|k-1} \\
&\chi^i_{k-1|k-1} = \hat{x}_{k-1|k-1} + (\sqrt{n\mathbf{P}_{k-1|k-1}})_i, \quad i = 1, \ldots, n \\
&\chi^i_{k-1|k-1} = \hat{x}_{k-1|k-1} - (\sqrt{n\mathbf{P}_{k-1|k-1}})_{i-n}, \quad i = n+1, \ldots, 2n \\
&W^0_s = W^0_c = 0 \\
&W^i_s = W^i_c = \frac{1}{2n}
\end{aligned} \tag{4.20}$$

For the mass spring model derived in Chapter 3, the filtering techniques that can be applied will be the EKF and the CKF, due to the model's nonlinearity. Both of these techniques will be implemented onto a real-time system and compared in Chapter 5

Chapter 5

Implementation of Estimation Filtering Algorithm

In this chapter, two different filtering algorithms, the EKF and CKF discussed in Sections 4.3 and 4.5, will be implemented on a real, hanging cloth system, to create a prediction filter for deformable surface motion. These algorithms will incorporate the dynamic system, and its linearization, described in Sections 3.1 and 3.2, as well as the backward Euler integration technique discussed in Section 3.3. Next, the filtering algorithms are used on a hanging cloth system, where position measurements are provided by an infrared tracking system, to both estimate true positions and velocities, as well as to predict future positions to compensate for system delays. The model parameters for the real system are identified using techniques specified in Section 3.5 and the noise parameters of the EKF and CKF are tuned to minimize predicted and measured outputs. The results of the EKF and CKF algorithms are compared and discussed.

5.1 Filtering Algorithm

The filtering algorithms presented in Chapter 4 are discrete time filters, and so require a discrete time system model. Therefore, the model equation presented in Section 3 will need to be discretized using the backwards Euler technique specified in Section 3.3. In other words, the state transition function

$$\dot{x} = f(x)$$

will be evaluated using the backward Euler method to solve for $x[k] := x(t + k\Delta T)$, where ΔT is some sampling time. This gives

$$\begin{aligned}
x_{k+1} &= \begin{bmatrix} p_{k+1} \\ v_{k+1} \end{bmatrix} = x_k + \Delta T f(x_{k+1}) \\
v_{k+1} &= \Delta v + v_k \\
p_{k+1} &= \Delta T v_k + 1 + p_k \\
\text{s.t. } &(\mathbf{I} - \Delta T^2 \frac{\partial a}{\partial p} - \Delta T \frac{\partial a}{\partial v})\Delta v = \Delta T a(p_k, v_k) + \Delta T^2 \frac{\partial a}{\partial p} v_k.
\end{aligned} \tag{5.1}$$

This discretized system can now be used on both the discrete time EKF and CKF algorithms, as the solution to the state prediction step, i.e.

$$\hat{x}_{k|k-1} = \hat{x}_{k-1|k-1} + \Delta T f(\hat{x}_{k|k-1}) \tag{5.2}$$

Extended Kalman Filter Implementation

To combine the EKF and the deformable model, it needs to be linearized using a first order approximation. This implies that the Jacobian of the new discrete time system, Equation (5.2), must be used. The new Jacobian, $F(\hat{x}_{k|k})$ now becomes

$$F(\hat{x}_{k-1|k-1}) = \frac{\partial \hat{x}_{k|k-1}}{\partial x}|_{\hat{x}_{k-1|k-1}} = \mathbf{I} + \Delta T \cdot J(\hat{x}_{k|k-1}) \tag{5.3}$$

where $J(\hat{x}_{k|k-1})$ is the Jacobian derived in Section 3.2. The prediction step for the EKF is now,

$$\begin{aligned}
\hat{x}_{k|k-1} &= \hat{x}_{k-1|k-1} + \Delta T f(\hat{x}_{k|k-1}) \\
P_{k|k-1} &= F(\hat{x}_{k-1|k-1}) P_{k-1|k-1} F(\hat{x}_{k-1|k-1})^T + Q_{k-1}
\end{aligned} \tag{5.4}$$

with the initial conditions $\hat{x}_{0|0} = x[0]$ and $P_{0|0} = \mathbf{0}$, and $Q_k = \alpha \mathbf{I}$ $\forall k \in \mathbb{Z}_+$, where $\alpha \in \mathbb{R}_+$. A diagonal matrix is chosen for Q_k as the process noise vectors are assumed to be independent random vectors.

After the prediction of the next state $\hat{x}_{k|k-1}$ and covariance matrix $P_{k|k-1}$ are calculated, and new measurement data is available, the update step of the EKF is applied to obtain the current state estimate and covariance matrix

$$\begin{aligned}
\tilde{y}_k &= y[k] - H_k \hat{x}_{k|k-1} \\
S_k &= H_k P_{k|k-1} H_k^T + R_k \\
K_k &= P_{k|k-1} H_k^T S_k^{-1} \\
\hat{x}_{k|k} &= \hat{x}_{k|k-1} + K_k \tilde{y}_k \\
P_{k|k} &= (\mathbf{I} - K_k H_k) P_{k|k-1}
\end{aligned} \tag{5.5}$$

The output function, $h_k(x)$, in Equation (4.12) is converted to a linear operation, $H_k x$, in Equation (5.5). This linear operation selects only the position states of x at time step k. Therefore,

$$H_k = H = \begin{bmatrix} \mathbf{I}_{3n\times 3n} & \mathbf{0}_{3n\times 3n} \end{bmatrix} \qquad \forall k = 0, 1, \ldots \tag{5.6}$$

The measurement noise covariance matrix, R_k, is chosen to be a diagonal matrix, similar to Q_k. However, it must be positive definite. Therefore, $R_k = \beta \mathbf{I} \;\; \forall k = 0, 1, \ldots,\ \beta \in \mathbb{R}_{++}$ (strictly positive numbers). A diagonal matrix is chosen for R_k as the measurement noise vectors are assumed to be independent random vectors.

Cubature Kalman Filter Implementation

Development of the cubature Kalman filter was done in conjunction with Cong Yue [37]. Implementation of the CKF is very similar to the extended Kalman filter, with the addition of a few steps in the creation of sample points. The prediction step of the CKF has two portions, one to generate a set of sample point, and one to generate the predicted state and covariance matrix. The weight generation step is the same as in Section 4.5, i.e.

$$\begin{aligned} \chi^i_{k-1|k-1} &= \hat{x}_{k-1|k-1} + (\sqrt{n\mathbf{P}_{k-1|k-1}})_i, \quad i = 1, \ldots, n \\ \chi^i_{k-1|k-1} &= \hat{x}_{k-1|k-1} - (\sqrt{n\mathbf{P}_{k-1|k-1}})_{i-n}, \quad i = n+1, \ldots, 2n \\ W^i_s &= W^i_c = \frac{1}{2n} \end{aligned} \tag{5.7}$$

with n being the number of mass nodes used in the model presented in Section 3.1. The predicted state and covariance matrices can then be calculated using the discretized state transition function shown in Equation (5.2), i.e.

$$\begin{aligned} \chi^i_{k|k-1} &= \chi_{k-1|k-1} + \Delta T f(\chi_{k|k-1}) \\ \hat{x}_{k|k-1} &= \sum_{i=1}^{2n} W^i_s \chi^i_{k|k-1} \\ P_{k|k-1} &= \sum_{i=1}^{2n} W^i_c (\chi^i_{k|k-1} - \hat{x}_{k|k-1})(\chi^i_{k|k-1} - \hat{x}_{k|k-1})^T + Q_k \end{aligned} \tag{5.8}$$

with the initial conditions and process noise covariance matrix being set to the same values as the EKF, that is, $\hat{x}_{0|0} = x[0]$, $P_{0|0} = \mathbf{0}$, and $Q_k = \alpha \mathbf{I}\ \forall k \in \mathbb{Z}_+,\ \alpha \in \mathbb{R}_+$.

After this, the update step is performed. This step can also be split up into a sample point generation phase and a separate update phase. The new sample points are generated using the same base functions as before. However, the state and variance, $\hat{x}_{k-1|k-1}$ and $\mathbf{P}_{k-1|k-1}$, are replaced by the predicted state and variance, $\hat{x}_{k|k-1}$ and $\mathbf{P}_{k|k-1}$, respectively. In other words,

$$\begin{aligned} \chi^i_{k-1|k-1} &= \hat{x}_{k-1|k-1} + (\sqrt{n\mathbf{P}_{k-1|k-1}})_i, \quad i = 1, \dots, n \\ \chi^i_{k-1|k-1} &= \hat{x}_{k-1|k-1} - (\sqrt{n\mathbf{P}_{k-1|k-1}})_{i-n}, \quad i = n+1, \dots, 2n. \end{aligned} \tag{5.9}$$

Theses samples points are then passed through the remainder of the update phase

$$\begin{aligned} \gamma^i_k &= H_k \chi^i_{k|k-1} \\ \hat{y} &= \sum_{i=0}^{2n} W^i_s \gamma^i_k \\ P_{y_k y_k} &= \sum_{i=1}^{2n} W^i_c (\gamma^i_k - \hat{y}_k)(\gamma^i_k - \hat{y}_k)^T + R_k \\ P_{x_k y_k} &= \sum_{i=1}^{2n} W^i_c (\gamma^i_k - \hat{x}_{k|k-1})(\gamma^i_k - \hat{y}_k)^T \\ K_k &= P_{x_k y_k} P^{-1}_{y_k y_k} \\ \hat{x}_{k|k} &= \hat{x}_{k|k-1} + K_k(y_k - \hat{y}_k) \\ P_{k|k} &= P_{k|k-1} - K_k P_{y_k y_k} K^T_k \end{aligned} \tag{5.10}$$

where again the output function, $h_k(x)$, in Equation (4.17) is converted to a linear operation, $H_k x$, shown in Equation (5.6). The measurement noise covariance matrix R_k is set to a diagonal matrix as in the EKF, $R_k = \beta\mathbf{I} \;\; \forall k = 0, 1, \dots,\ \beta \in \mathbb{R}_{++}$.

Often times, when applying the EKF or CKF to a system, the sample rate of the measurements being provided is far higher than the sample time required by an integration method to provide an accurate solution to its dynamical model. Therefore, it is required that the integration solver, i.e. the prediction step, be run at a rate higher than when a measurement can be provided. This is completely possible by running the prediction step multiple times, until the next available measurement is made [10, 52]. This allows for a smoother estimate, with less error. Additionally, this can be used to improve computation time, since the prediction step can be run while the system waits for a measurement, and then updated when the measurement comes in, versus executing both steps at the same time.

When using a projector for spatial augmented reality applications, there are inherent limitations that need to be addressed. One such flaw is the intrinsic delay involved in processing the image to be projected, as well as the time required for the projector to display the image. It is well known that projectors suffer from delays when processing images and these delays usually range from 20ms to 100ms depending on the type of projector [59]. This delay, T_d, is troublesome when using a prediction filter for surface prediction in real-time. Since an image needs to be sent to the projector T_d seconds in advance, to be projected at the correct time, the filter needs to predict the geometry of the surface T_d seconds in the future at each predict step. Now, since measurements are received every T_m seconds, the prediction filter can only update the state estimate every T_m seconds. An issue arises when the delay time T_d and measurement time T_m do not match (i.e. are vastly different). The time of the current state prediction and the time at which the measurement is made will never be the same. This means the traditional EKF/CKF algorithm will not work, as the prediction and measurement times need to line up. To fix this issue, a further prediction, using numerical integration, is made to align the time of the current state prediction with the current measurement. At this stage, a new estimate can be made using the regular estimation algorithm. Figure 5.1 shows how the timing of the algorithm functions. Every T_m seconds a measurement is received and an update is made. This update is then used to create a state prediction T_d seconds in the future. The predicted state is then advanced every ΔT seconds to ensure that a prediction is available when needed. When a new measurement comes in the stored predicted state at the time of the measurement is used, and the algorithm repeats. A simplistic flow chart displaying the steps involved in the EKF and CKF implementation can be seen in Figure 5.2.

5.2 Experimental Setup

In order to validate the EKF and CKF algorithms it must be implemented on a real world system. To mimic a fairly flexible and dynamic system, a cloth-like material is desired. This would test the effectiveness of the algorithm, in estimating positions of a very unpredictable surface. As a result, a towel is used for experimental data collection. The prediction filters being implemented require the positions of points along the object surface to be measured and used as a variable during their update stage. Therefore, a sensor system that can capture positional data is required for data collection. A number of different systems exist for capturing positional data, such as image processing techniques or time-of-flight sensors. However, for greater data accuracy, an infrared motion capture system is used in this experiment. The NaturalPoint OptiTrack system [4] is an infrared camera based motion capture system. It incorporates a multi camera setup (Figure 5.3a) to provide positional data within millimetre precision. The OptiTrack system measures the position

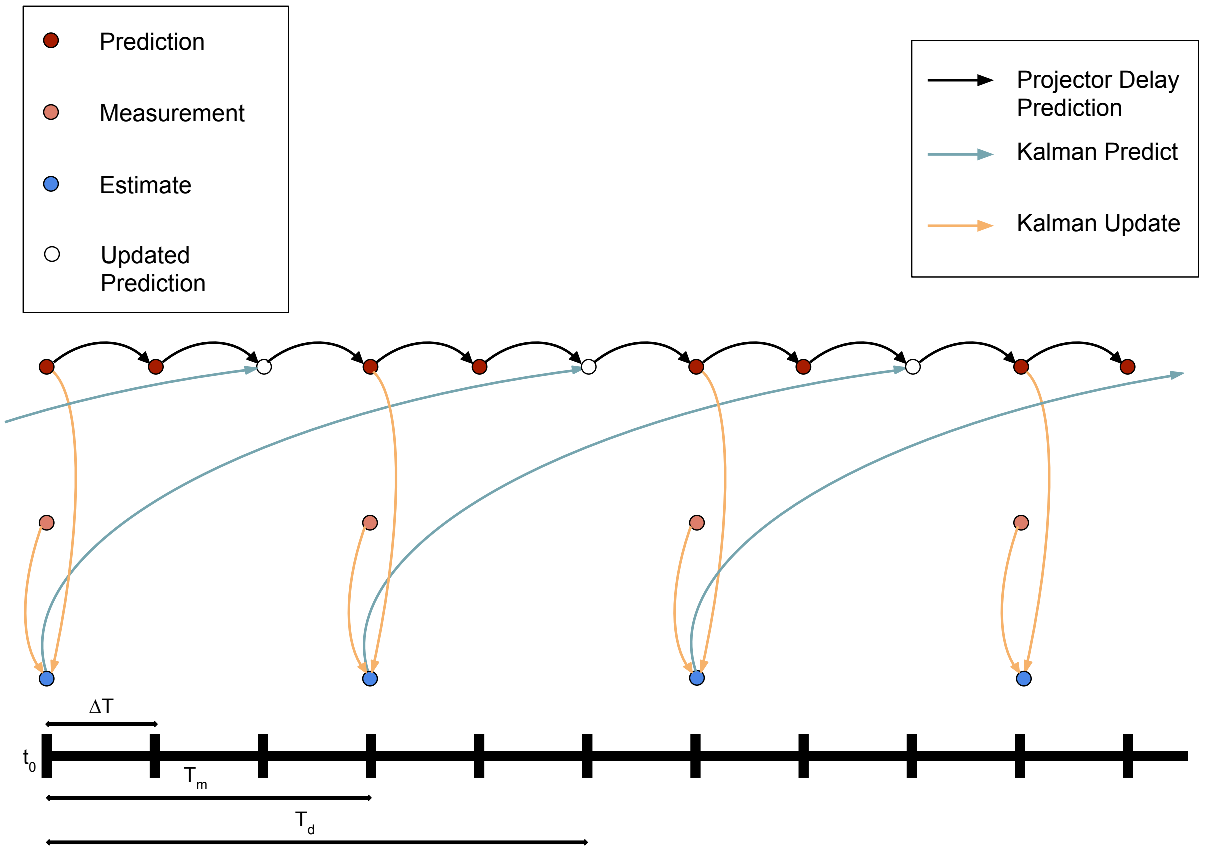


Figure 5.1: Timing diagram of a projector compensated filtering algorithm. ΔT is the integration time-step, T_m is the measurement time, and T_d is the delay time.

of retro-reflective infrared markers by triangulating each marker with multiple cameras, calibrated to have a known position. For this experiment, a four camera configuration, made up of the Flex:V100, is used to measure the position of 12.7mm diameter infrared markers (Figure 5.3c). The markers are placed on the surface of the towel and are matched to the initial positions of the desired nodes in the model. The four OptiTrack cameras are placed in a way that minimizes occlusion by making sure that at least two cameras are in sight of all markers at all times, virtually shown in Figure 5.4. The towel used is 0.57m wide by 0.81m long and is fitted with 20 infrared markers. These markers are placed in a 5×4 grid with locations determined using the nodal placement algorithm in Section 3.4, as shown in Figure 5.5.

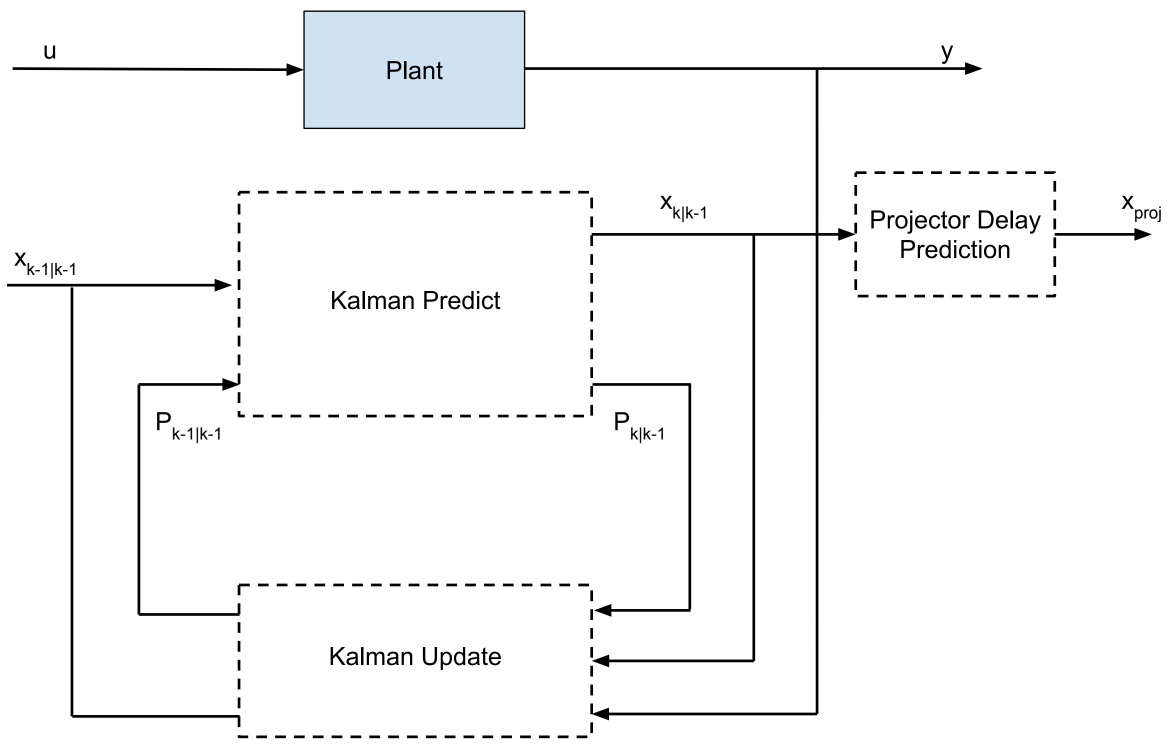
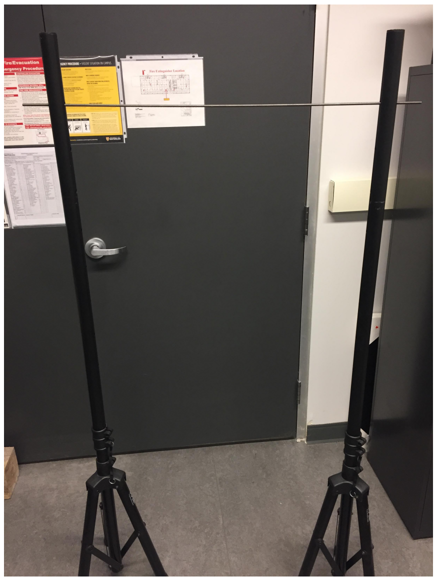


Figure 5.2: Flow chart of the EKF/CKF algorithm with the mass-spring model

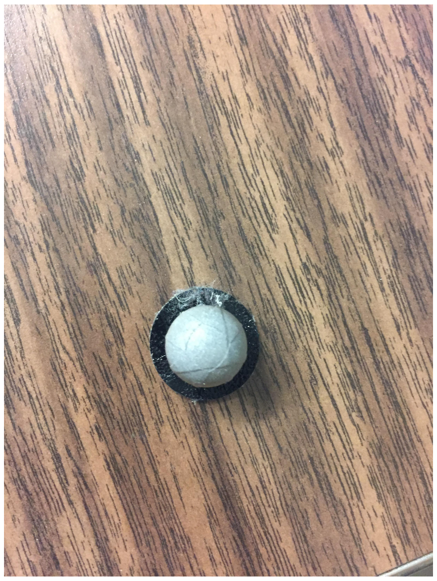
The cloth is anchored along the top row of markers by an improvised fixture shown in Figure 5.3b. As a source of disturbance, a small fan is placed behind the cloth to provide a randomly acting external force to the cloth. The fan rotates sideways, producing oscillatory forces on the cloth.

Before any data collection or disturbances are added to the cloth, its rest position is recorded to calculate the rest lengths of the springs in the model. Additionally, this rest position is used to initialize the filtering algorithms by setting $\hat{x}_{0|0}$ to this measured value, allowing us to set $P_{0|0}$ is set to the zero matrix, $\mathbf{0}_{6n\times 6n}$, since the cloth model begins at the same position as the actual towel. Once this is complete, data from the OptiTrack cameras can be broadcast, using the NatNet SDK provided by NaturalPoint. The marker positions are measured 100 times per second, in other words, a measurement is made every 0.01 seconds. If the measurement is needed to be reduced, for robustness testing, the unnecessary data points recorded are ignored by the algorithm. Data for the markers is

(a) OptiTrack Flex:V100 camera



(b) Cloth hanging fixture



(c) 12.7mm infra-red marker

Figure 5.3: Experimental setup equipment

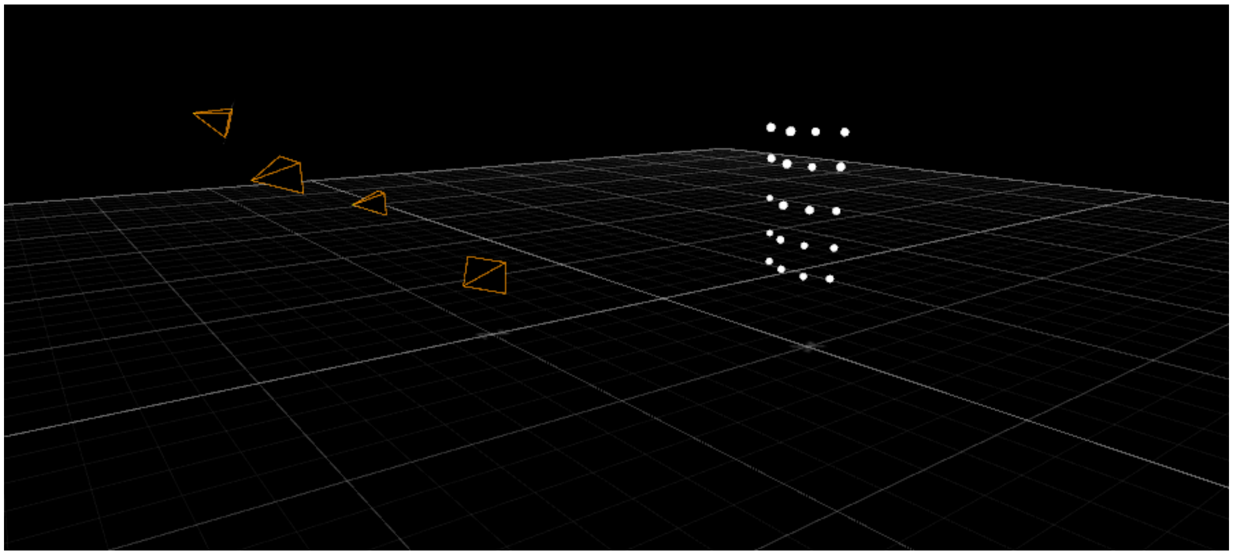
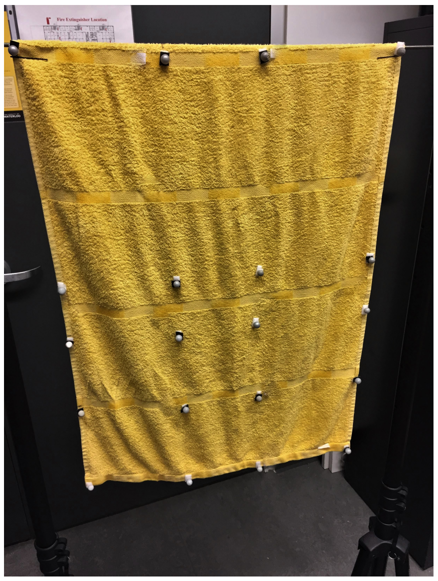


Figure 5.4: Camera location relative to towel


(a) Towel with 20 IR markers

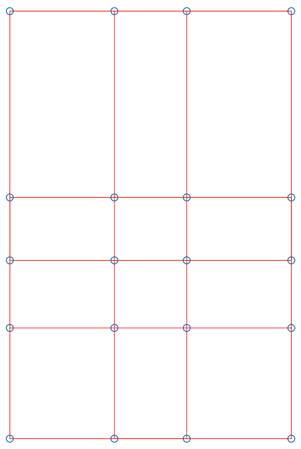

(b) Optimal node location

Figure 5.5: Optimal nodal locations vs. placed node locations

recorded as a set of (x, y, z) coordinates. However, the organization of the marker data can be random, leading to what is known as marker swapping [72]. Marker swapping is the phenomena in marker tracking where a markers label gets swapped with the label of a nearby marker and so swaps the identification of each marker is lost. In order to combat this a least squares approach is used. The set of marker associations that minimize the sum of the squared errors between the estimated output $\hat{y}$ and measured position $\tilde{y}$ is chosen as the true measurement y, i.e.

$$y = \arg\min_{\tilde{y}} \sum_{i=1}^{\frac{n}{2}} \|\hat{y}_i - \tilde{y}_i\|_2 \tag{5.11}$$

Another important facet of the experimental setup is the projector. The projector is used to display the rendered and mapped image, produced by the estimation algorithm, on the surface of the measured object. The projector used for the experiment is the Epson VS240. It is necessary to identify some projector parameters that will be used to calculate the projection matrix, discussed in Section 2.2.2, as well as the projector delay time, described in Section 5.1. The projector matrix is required to ensure that the image rendered by the system matches what is displayed. This is especially problematic with the projector used, as the VS40 is a short throw projector. What this means is that the lensing characteristics of the projector cause the top of the projection to undergo significantly more stretching compared to the bottom. From Section 2.2.2, we know that the intrinsic projection matrix F is dependant on the distance n from the projector to the display surface, and the distance from the centre of the projector to the left, right, top, and bottom edges of the display surface, $l, r, t, \text{and} b$. These parameters can be visualised in Figure 5.6, and are provided by [5]. Therefore, with $n = 4$, $l = -1.365$, $r = 1.365$, $t = 1.624$, and $b = -0.0862$ the projection matrix becomes,

$$F = \begin{bmatrix} \frac{-2n}{r-l} & 0 & \frac{r+l}{r-l} \\ 0 & \frac{-2n}{t-b} & \frac{t+b}{t-b} \\ 0 & 0 & -1 \end{bmatrix} = \begin{bmatrix} \frac{-8}{2.73} & 0 & 0 \\ 0 & \frac{-8}{1.71} & \frac{1.538}{1.71} \\ 0 & 0 & -1 \end{bmatrix} \tag{5.12}$$

In addition to the projection matrix of the projector, the projectors delay time T_d can be measured for incorporation with the filtering algorithms of Section 5.1. To measure this delay a rudimentary system is built using an Arduino micro controller, an LED, and a camera recording at 240 frames per second. The Arduino controller is used to enable an LED by actuating an on board push button (Figure 5.7), simultaneously a command is sent to cause the projector display to flash on. This sequence of events is recorded using

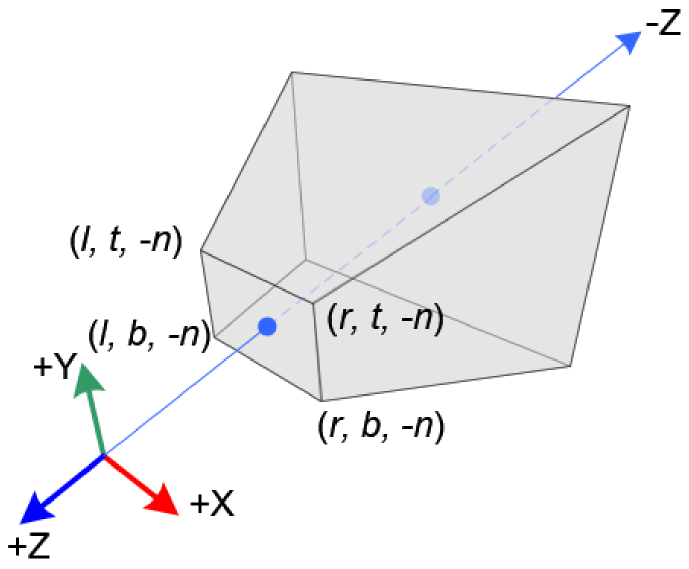


Figure 5.6: Perspective parameters in relation to a projector at the origin

a slow motion camera, and the time between the LED enabling and the projector flashing on is measured as the delay time T_d. This is repeated multiple times and the average value of 0.075 seconds is used.

The overall experimental layout can be seen in Figure 5.8. All experimental data and algorithms were collected and executed on a single computer. The specifications of the computer include a 3.8GHz i5-7600K CPU with 4 threads, a GTX 1080 Ti graphics card, and 16GB of RAM.

5.3 Evaluation and Tuning

The effectiveness of the prediction algorithms presented in Section 5.1 are evaluated on the experimental setup, described in Section 5.2, using both qualitative and quantitative methods. A qualitative comparison is used to ensure that the results of the experiment are visually pleasing, and not jarring to experience, where as a quantitative measure is used to ensure that the prediction algorithm is truly behaving as expected, and closely

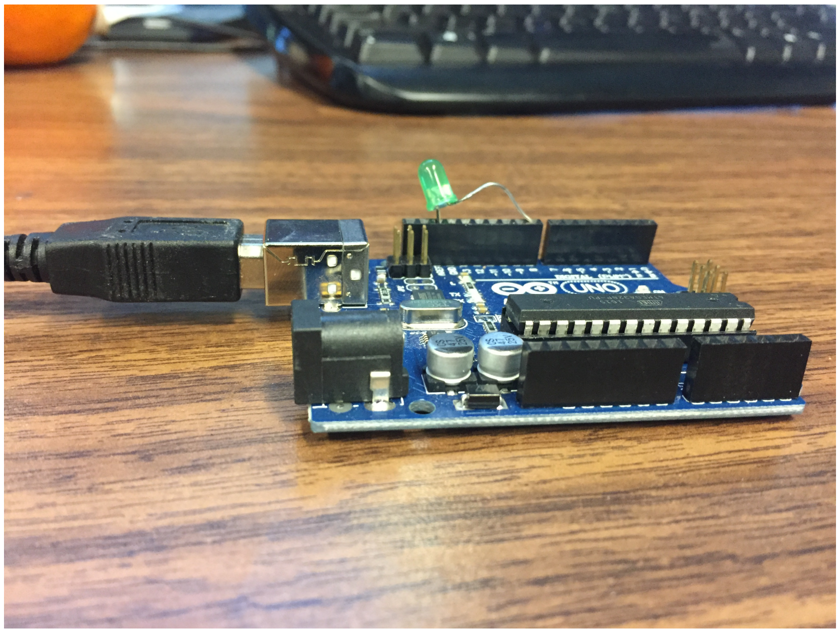


Figure 5.7: Arduino setup with LED used for delay time identification

matching the true position of the surface. Qualitatively, the results of the both the EKF and CKF prediction algorithms are visually compared to each other. When the image is projected onto a flat, static surface (the towel at rest), both projection methods produce the exact same results. However, once the towel is disturbed by the fan, there is a noticed difference. Both algorithms perform better than no algorithm running. However, the CKF method produces slightly more true-to-life results when compared to the EKF algorithm. When projecting onto a surface that has been deformed to a static state, both the CKF and EKF algorithms perform identically and far outperform the standard, sans algorithm, projection method, as shown in Figure 5.9. In the three orientations shown in Figure 5.9 the CKF and EKF algorithms both compensate identically since the towel being stationary means their solutions converge. However, the uncompensated projection produces clipped and undesirable results. Specifically, the uncompensated projection method displays parts of the image past the towel, onto the wall, while the prediction algorithms "paints" the image on the towel.

Quantitatively, the success of the prediction algorithms are evaluated using their mean squared error (MSE) between the measured position of the markers and the predicted position of the mass nodes. At every measurement time-step, the difference between measured position of a node and its predicted position are squared and then averaged. The mean

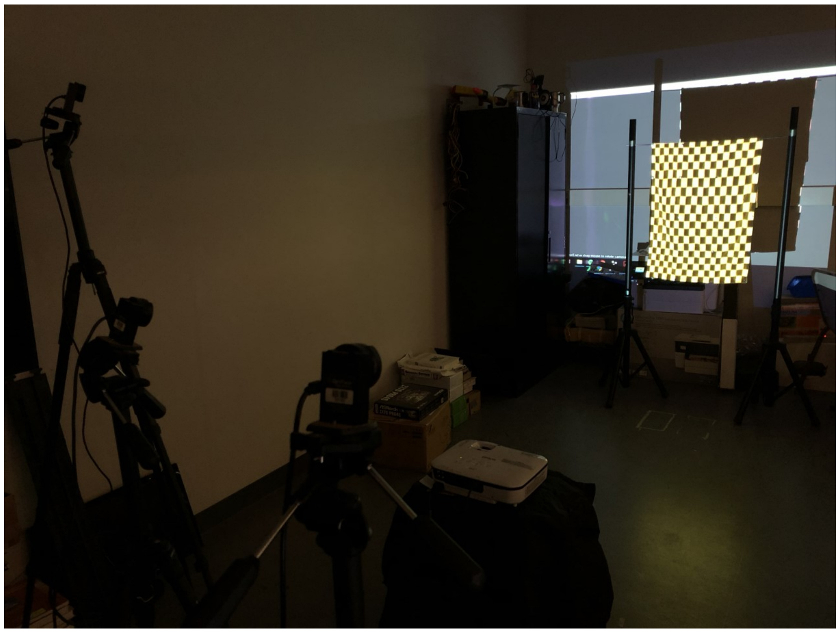


Figure 5.8: Photo of experimental setup with motion capture cameras, a projector and a towel being projected onto.

squared error is defined as

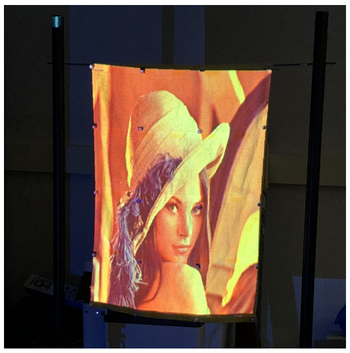
$$\mathbf{e}[k] = \frac{1}{N}\sum_{k=1}^{N} \|y[k] - H\hat{x}_{k|k}\|_2 \tag{5.13}$$

where N is the total number of nodes (20 in this case), $y[k]$ is the measured position, and $x_{k|k}$ is the state estimated vector. This is used as an overall view of the total error in the system. Additionally, the maximum Euclidean error (MEE), between a predicted and measured node, at each time step is used to identify the, worst case scenario, error of the system. This is given by,

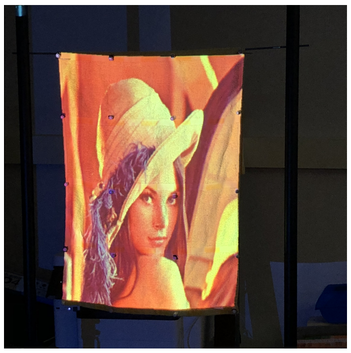
$$e[k] = \max_i \|p_i[k] - \hat{p}_i[k]\|_2 \tag{5.14}$$

where p_i is the position of the ith node, and $\hat{p}_i$ is its predicted value. The MEE provides the error of the node, at time k, that was least effectively estimated, therefore giving the largest nodal error at each time step.

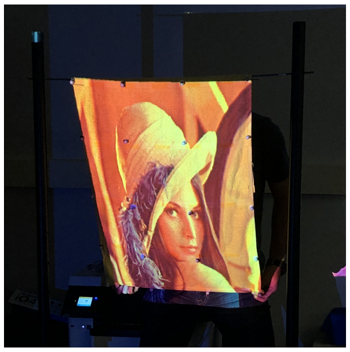
The first test scenario evaluates the effectiveness of model parameter identification. The test is done on a 5×4, evenly distributed node configuration (i.e. without using the optimal node locations), with $\hat{x}_{0|0}$ set to the initial measured value, allowing us to set $P_{0|0} = \mathbf{0}_{6n\times6n}$.



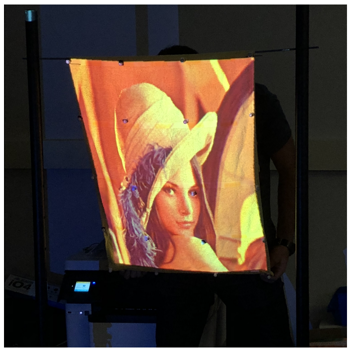
(a) No prediction: at rest



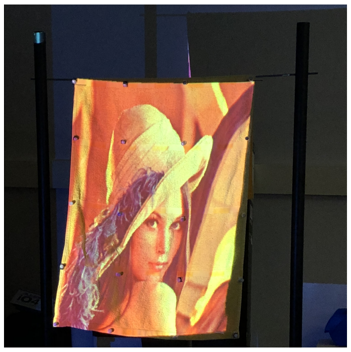
(b) CKF prediction: at rest



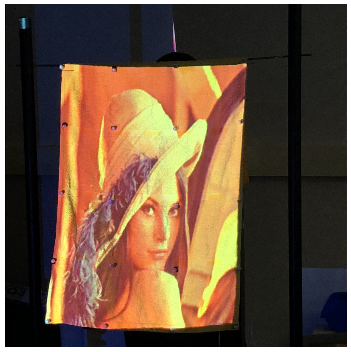
(c) No prediction: pulled backwards



(d) CKF prediction: pulled backwards



(e) No prediction: pushed forward



(f) CKF prediction: pushed forward

Figure 5.9: Visual comparison of standard projection and prediction algorithms on static deformations using the CKF algorithm without optimal node locations

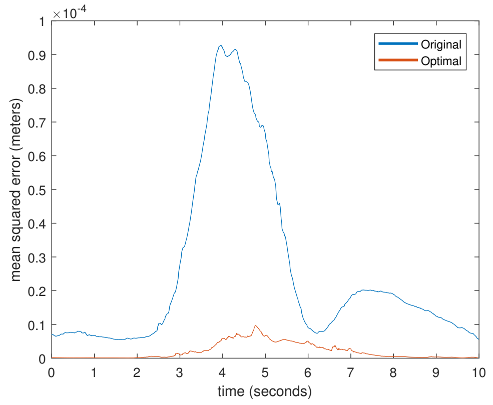
Additionally, R_k and Q_k are set to $\mathbf{I}$ and $10000\mathbf{I}$ respectively. Measurement data is provided to the filtering algorithm at a rate of 100 samples a second, i.e. 1 sample every 0.01 seconds, and an integration time of 0.005 seconds is used. Although the EKF algorithm is capable at integrating with a much finer time compared to the CKF, it is kept at 0.005 seconds in order to maintain consistency between the two algorithms. Additionally, the CKF algorithm could always be made to execute faster with the use of higher end hardware.

In order for the dynamic model, described in Section 5.1, to be able to calculate the predicted states,it needs to be an accurate representation of the real life system. Therefore, the parameters of the mass spring model need to be identified for it to closely match the towel's behaviour. To accomplish this the real life hanging towel is dropped from a predictable initial condition and its dynamics are measured. This set of measurements can then be passed into the model compression algorithm devised in Section 3.5. The simulated annealing parameter identification technique is used to find a model that best matches the real world measurements, i.e. the following minimization problem is solved

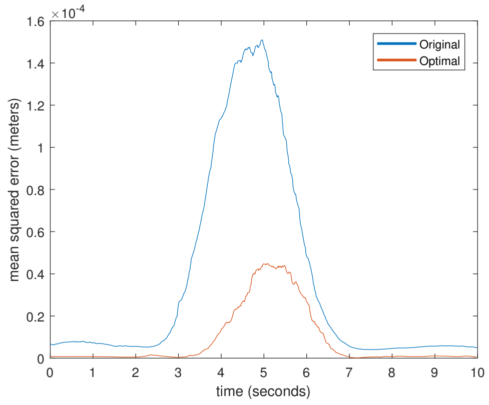
$$\begin{aligned} C(\theta) = & \sum_{i=0}^{N} e^{\alpha i \Delta T} \| Hz[i] - Hx[i] \|_2^2 \\ \text{s.t.} \qquad & z[i] = f(z[i-1], \theta) \end{aligned} \tag{5.15}$$

where x is the measured state, z is the simulated state, H is the observation model, θ is the parameter vector and $\alpha = 0.001$. The final parameter vector is applied to the filter model. The results are implemented on the experimental setup specified in Section 5.2, with an oscillatory fan used as random input to the system. The effect of the optimized values can be seen in Figures 5.10 and 5.11, where it is compared to an model with arbitrary parameters, set as follows: $k_{s_i} = 420$, $k_{d_i} = 0.05$, and $m_i = 0.13$ for all parameters. It is clear that the MSE of the optimal system has far lower errors than the arbitrary system, for both the CKF and EKF algorithms. The mean squared error is more than halved for the EKF and is nearly a tenth of the original error in the CKF system. The MEE graph in Figure 5.11 also presents promising results, with the maximum nodal error dropping from 12cm to a more useful 4-7cm on both algorithms showing that the parameter identification algorithm is effective. In both, the mean squared error and the maximum Euclidean error graphs, there is a spike in error at the 5 second mark. This corresponds to the oscillatory motion of the disturbance fan. As the fan turns towards the towel, it exerts a larger amount of force directly onto it. As neither the EKF or the CKF algorithms are aware of the external input, the error is larger when the force is being applied, however, these algorithms still succeed in keeping the error to a minimum.

Next, to further increase the realism of the system, the optimal nodal placement algorithm from Section 3.4 is implemented. A standard 5×4 arrangement of evenly spaced



(a) CKF MSE

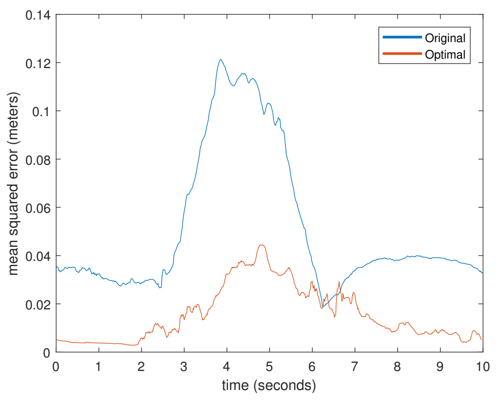


(b) EKF MSE

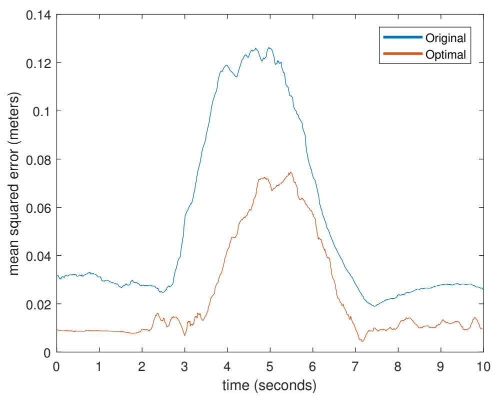
Figure 5.10: CKF and EKF MSE comparison between original and optimal parameter

markers may seem like a good arrangement to capture the dynamics of the hanging cloth system, however, as explained in Section 3.4, there is a possible distribution, Figure 5.5b, that could lead to a better visual experience. Nevertheless, this system also needs to provide adequate dynamic modeling, in order to keep the prediction error to a minimum. The errors of the standard even arrangement is compared to that of the optimal location model in Figures 5.12 and 5.13. These error graphs show that the optimized distribution has a very similar, but still lower error when compared to its evenly distributed counterpart. The slight improvement can be explained by the system being able to better capture the points of maximum curvature in the towel, therefore leading to less cumulative error in the system. The use of the optimal node distribution allows the projection to be both accurate and visually pleasing.

Both the EKF and CKF require the a priori knowledge of the process and measurement noise statistics [10, 76] to produce accurate results. However, this statistic is often times difficult to obtain and must be tuned to improve the performance of the estimation filter. To achieve this a trial and error approach is taken. The noise affecting the system is assumed to be independent, zero-mean and normally distributed with equal variance. Therefore, the noise covariances are made up of scaled diagonal matrices, i.e. $R_k = \beta\mathbf{I}$, $Q_k = \alpha\mathbf{I}$. The measurement noise parameter is known to be small, because of the sensitivity of the sensors used, and is calculated based on the specification of the OptiTrack camera system, to be $R_k = 0.001\mathbf{I}$. Therefore, the value of Q_k can be determined by varying the value of α until a satisfactory result is achieved. Figures 5.14 and 5.15 show the effect different values of α have on the MSE of the system. Figure 5.14 shows that for values of Q_k that are too



(a) CKF MEE



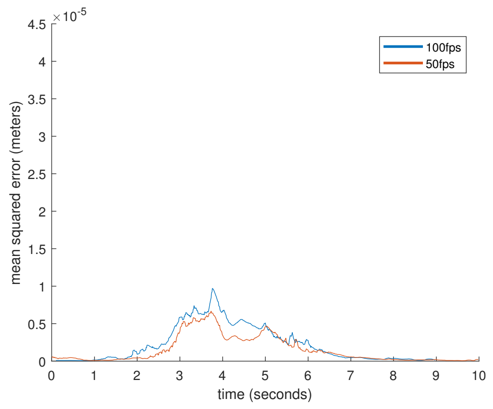
(b) EKF MEE

Figure 5.11: CKF and EKF MEE comparison between original and optimal parameter

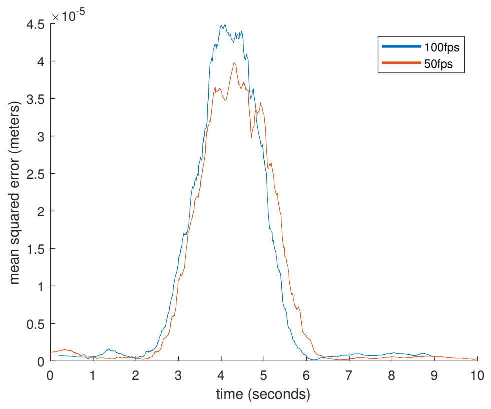
low, the CKF algorithm has significantly greater errors, However as its value increases, the error in the system drops appreciably. At a value of 10$\mathbf{I}$ Q_k is shown to have markedly low MSE while increasing it any higher having negligible benefit. Though, it is important to note that the large error of the CKF at low Q_k values is still relatively small, at 0.0002m^2. Therefore, for the CKF algorithm $Q_k = 10\mathbf{I}$. The EKF algorithm however, is much more robust and is able to handle lower, and wider, range of Q_k values without significant error. Therefore, for consistency, its Q_k value is set to 10$\mathbf{I}$ to match the CKF algorithm.

A visual comparison of the CKF prediction algorithm, using all of the optimised parameters, and no prediction algorithm can be seen in Figure 5.16. This shows that the prediction algorithm significantly compensates for the deviations caused by the moving fan behind the towel. Most notably, the image projected onto the bottom corners of the towel can be seen to more accurately match the surface of the towel when the algorithm is used. In contrast, a lack of prediction causes the image to bleed outside of the towel boundaries.

Lastly, the overall effectiveness of both the EKF and CKF algorithms is tested by combining all of the optimization and tuning algorithms previously discussed in this section. The optimized model parameters are combined with both the optimal mass node locations, and the tuned noise parameters to enable the filtering algorithms to perform as best as they can. To further push the limits of the EKF and CKF algorithms, they are applied to a system with a measurement sample rate of 50 samples a second, half that of what has been used so far. This is done to ensure that the system is effective in scenarios where high speed equipment is not available and as a robustness test for the algorithms being used. Figure 5.17 shows the results of the CKF and EKF algorithms being run at a 50 sample



(a) CKF MSE

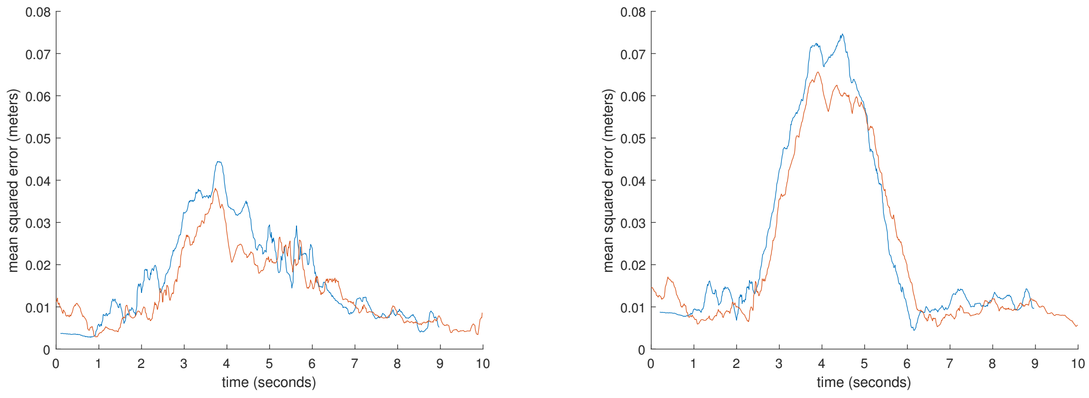


(b) EKF MSE

Figure 5.12: CKF and EKF MSE comparison between even and optimal nodal locations

per second measurement rate, compared to a 100 sample per second measurement. The results show that, although the algorithms running at a rate of 100 samples per second are more accurate, they are still very accurate when run at a slower rate. The overall error of the slower system is very close to the faster one. However, the slower system has an artifact, in that there is a lot of jitter, or noise in the estimated output. This, although small, can have the effect of making the projected image look jarring and unrealistic. The CKF algorithm can be seen to have a much smaller amount jitter in this case, and so is perceived to be more realistic. The CKF algorithm as a whole, performs significantly better than the EKF. In all the test cases presented in this section, the CKF algorithm has had errors at least half that of the EKF algorithm. This implies that the multi sampling method of the CKF algorithm is far more effective than the EKF, especially in systems with low sample rates.

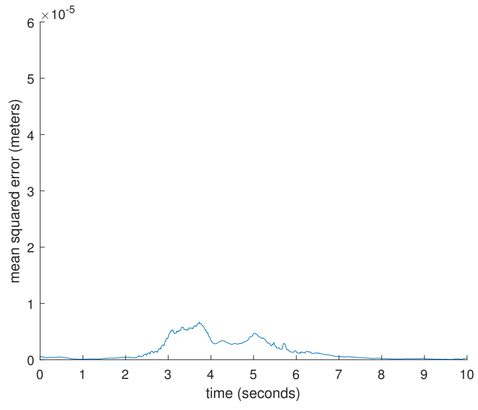
The computation costs of running a CKF in real time are slightly higher than that of an EKF, though, its efficiency can be improved by implementing parallel computations. The largest use of time in the algorithm is the evaluation of the state prediction for each of the $2n$ sample points. However, due to the nature of the algorithm, each of these points can be predicted independently, and so can be easily parallelized. Each sample can be assigned to a separate computer thread and computed individually, thus greatly improving the execution time. The average run time per iteration for a 5×4 hanging cloth system, with $\Delta T = 0.005$ seconds, of the CKF without parallelization is 0.035 seconds. However, after parallelization, the run time drops significantly to 0.011 seconds. Although the EKF algorithm runs significantly faster at 0.002 seconds, the CKF is still usable for real time



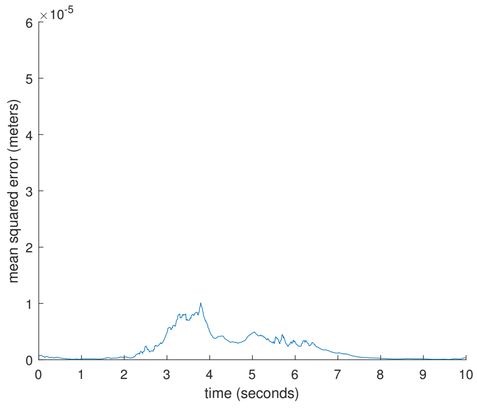
(a) CKF MEE (b) EKF MEE

Figure 5.13: CKF and EKF MEE comparison between even and optimal nodal locations

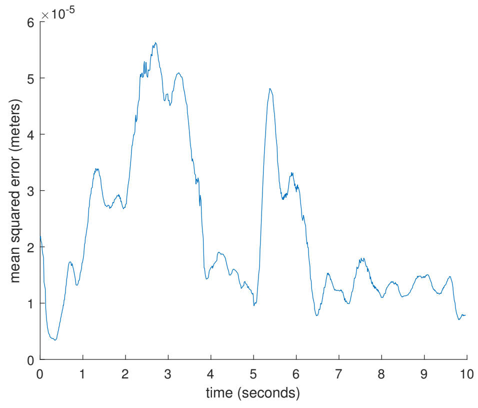
usage all the way up to 100fps. Further still, this time can be significantly improved by using CPUs with higher thread counts, as this problem is very scalable. Therefore, it is highly advisable that the CKF be utilised on systems of this nature over the EKF.



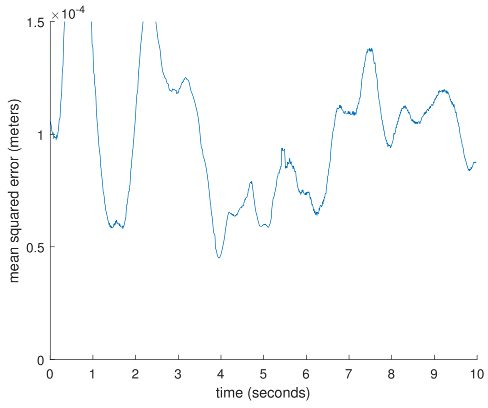
(a) CKF with $Q_k = 10\mathbf{I}$ and $R_k = 0.001\mathbf{I}$



(b) CKF with $Q_k = 1\mathbf{I}$ and $R_k = 0.001\mathbf{I}$

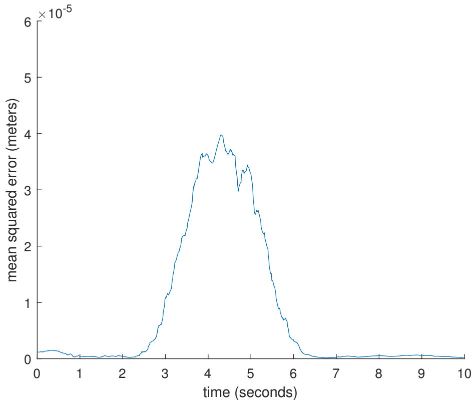


(c) CKF with $Q_k = 0.1\mathbf{I}$ and $R_k = 0.001\mathbf{I}$

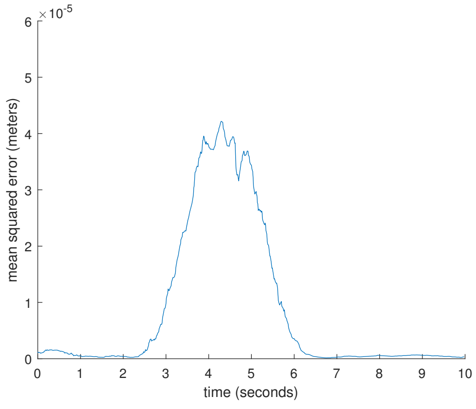


(d) CKF with $Q_k = 0.01\mathbf{I}$ and $R_k = 0.001\mathbf{I}$

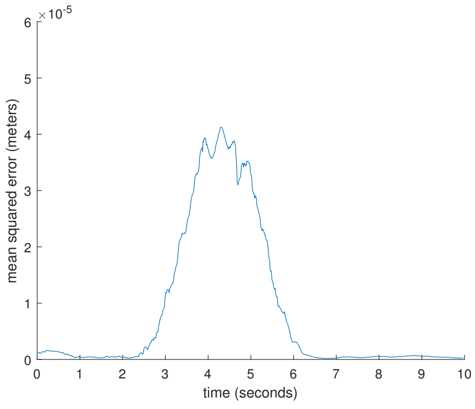
Figure 5.14: CKF MSE for varying noise parameter Q_k



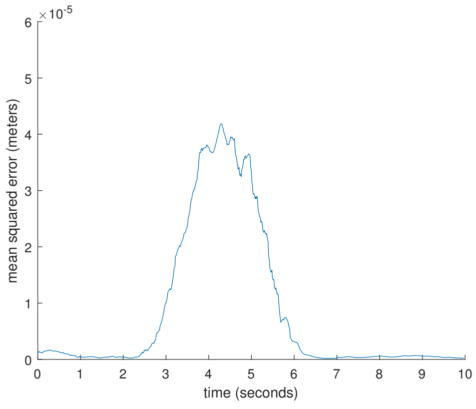
(a) EKF with $Q_k = 10\mathbf{I}$ and $R_k = 0.001\mathbf{I}$



(b) EKF with $Q_k = 1\mathbf{I}$ and $R_k = 0.001\mathbf{I}$

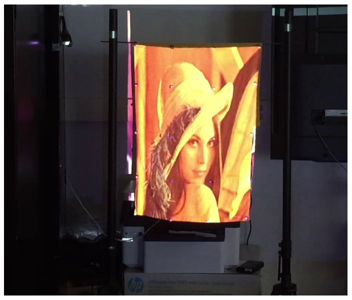


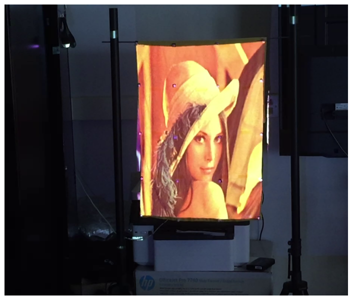
(c) EKF with $Q_k = 0.1\mathbf{I}$ and $R_k = 0.001\mathbf{I}$

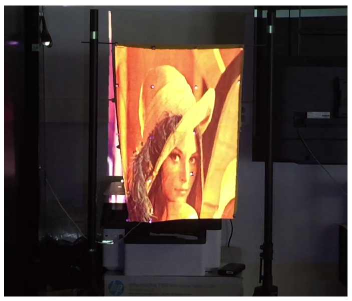


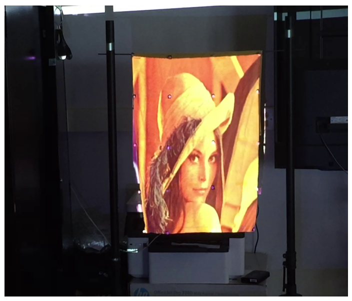
(d) EKF with $Q_k = 0.01\mathbf{I}$ and $R_k = 0.001\mathbf{I}$

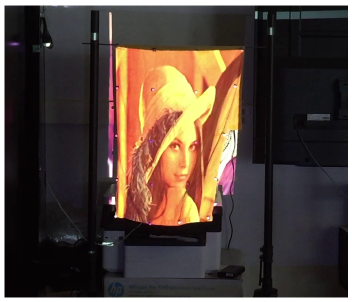
Figure 5.15: EKF MSE for varying noise parameter Q_k


(a) No prediction: t = 0s


(b) CKF prediction: t = 0s


(c) No prediction: t = 4s


(d) CKF prediction: t = 4s


(e) No prediction: t = 6s

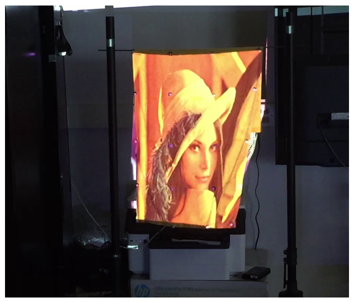
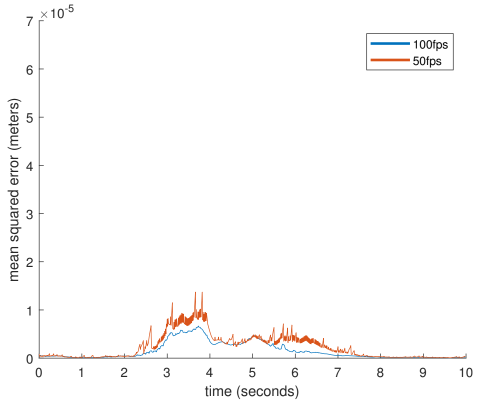
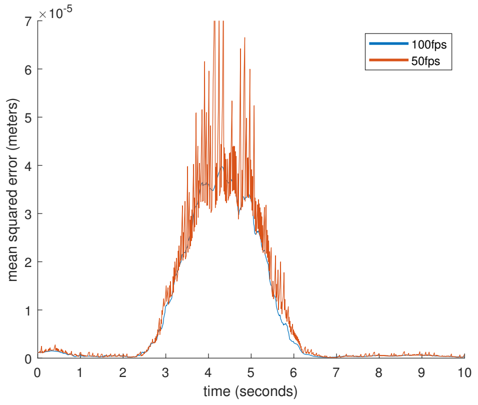

(f) CKF prediction: t = 6s

Figure 5.16: Visual comparison of standard projection and CKF algorithms with oscillating fan disturbance



(a) CKF MSE



(b) EKF MSE

Figure 5.17: CKF and EKF MSE comparison between 100fps and 50 fps measurement rate

Chapter 6

Conclusion and Future Work

The purpose of this thesis was to implement a filtering algorithm to be used for prediction of positions on time varying surfaces, more specifically, to be used along with a projection system to allow for immersive augmented reality experiences. Two prediction schemes are proposed, an extended Kalman filter approach, and a cubature Kalman filter approach. Both algorithms made use of a mass spring model, to simulate the dynamics of a real system, and a series of optimization techniques to improve their accuracy.

In Section 2.1, a number of deformable models were explored. Two main types of models were identified, non-physical and physical. Non-physical models use purely geometric techniques to manipulate an object. They rely on designer skill, rather than accounting for material properties, and so are unsuitable for application on complex systems. Physical methods use physics to compute realistic simulations of the dynamics of a system. Two main physical methods are explored, mass spring models and finite element models. Mass spring models were found to provide a simple representation of a system by using a set of discrete mass nodes interconnected by springs and dampers. They are easy to construct and computationally efficient. However, they are only approximations of the real system, due to their discrete nature. Finite element models, on the other hand, are a continuum model and treat objects as a continuous objects, making them more accurate. This accuracy, however, comes at the cost of computational expense. Therefore, the mass spring model is the model of choice, as computational efficiency is a requirement for real time applications.

Section 3.1 goes into details of the mass spring model as well as its derivation. The dynamics of this model can be represented using a nonlinear state space model. The internal dynamics of the system are created by forces from springs and dampers and are additionally affected by external forces. Linearization of the model is derived in Section 3.2 for the purposes of implementation on the EKF and CKF algorithms. Section 3.3 discusses two numerical integration schemes, explicit and implicit, used to solve the dynamic equation provided by the model. The Runge-Kutta method is discussed as a viable explicit integration scheme, while the backward Euler method is used for the second implicit integration scheme. The Runge-Kutta method relies on solving a set of equation that use only previous data points to calculate the next output. As a result, it can become unstable when dealing with numerically stiff systems. Thus, small integration step times are required to obtain stable results. The backwards Euler method, on the other hand, makes use of a system of equations made up of both the previous and the next data point to compute the next output, making it inherently stable. To solve this system of equations for real time use, a Taylor series approximation is made. Although the computation time required to use this method is larger than for the Runge-Kutta method, its inherent stability allows for larger integration step times. The use of larger step times allows the backward Euler method to have comparable, in execution time, to the Runge-Kutta method. Additionally, the added benefit in stability allows the system to be more robust.

Nodal placement and model compression algorithms can be used to enhance the efficiency of the EKF and CKF filters. Section 3.4 discuses the use of optimal nodal placement to more closely capture the dynamics of a hanging cloth system. It is observed that a cloth, that is fixed on one end and allowed to fall, has dynamics comparable to a cantilever beam in free vibration. The longitudinal axis of the cloth behaves like the second mode of vibration in a fixed-free beam, while the lateral axis of the cloth is similar to the first mode of a free-free beam. Using this information, along with displacement equations for Euler-Bernoulli beams, an optimal nodal placement algorithm is derived that closely estimates the continuous surface of the cloth with discrete linearly connected points. Additionally, a model compression algorithm is discussed in Section 3.5. Model compression is used to verify that a continuous surface, best described by a large number of mass nodes, can also be emulated by a sparser mass spring system, with optimized parameters. Smaller node numbers allow for faster computation time, which is highly desirable for real-time applications. To achieve this goal, a simulated annealing minimization algorithm is used to find a set of parameters that minimize the error between a higher order system and an equivalent low order one. This algorithm was used to compress a 21×21 model to a 5×5 model. The results showed that mean squared error of the reduced order system, compared to the original system, is low, and so can be used by the EKF and CKF algorithms to model the real life system. Additionally, this simulated annealing algorithm can be used to identify the parameters of a real system. This is done by simply minimizing the error between the compressed model and the real life measured data, instead of a high order model.

In Chapter 4, the EKF and CKF are formulated. The EKF algorithm uses a linearized version of the mass spring damper model, as a state transition matrix, to predict the value of its state. This predicted value is then updated to a more accurate estimate once a measurement is received. The CKF performs similar set of operations, however, it uses the original, nonlinear dynamic equation of the mass spring damper system to predict the next value of the system state. Additionally, to better capture the variance of the state, multiple sample points are used and propagated through the system. Again, once a measurement is received, this predicted state can be updated to produce a more accurate estimate. The EKF and CKF algorithms are both evaluated on a hanging cloth in Chapter 5. Both algorithms were tuned using the simulated annealing parameter identification and nodal placement algorithms. Additionally, the noise covariance matrices, R_k and Q_k for each algorithm, were determined by using the known error of the OptiTrack camera system and by varying the Q_k parameter such that the mean squared error of the estimate was minimized. Lastly, the EKF and CKF algorithms were both evaluated on a limited system where measurements were made 50 times a second, versus the usual 100. The optimizing and tuning algorithms were all shown to be useful in reducing the error of the system, and the EKF and CKF were both able to estimate the position of the hanging cloth accurately in even the low measurement rate scenario. Through all of the cases tested the CKF estimate was shown to perform significantly better than its EKF counterpart, with position errors close to half that of the EKF.

In Chapter 5, a marker based tracking system is used for measuring highly accurate position values. However, they are prone to marker swap and require the placement of physical markers on the target surface. To remedy marker swap a unique placement system, like the dot cluster marker in [67] can be used. The dot cluster algorithm uses a uniquely identifiable set of node markers that can be easily identified without worry of marker swap. In place of marker tracking, a computer vision based solution, such as feature selection can be used to eliminate the use of physical markers.

Improvements to the algorithm could be made by further investigating the model parameter identification procedure. The simulated annealing based algorithm, specified in Section 3.5, requires a large amount of time to generate parameters, and so the parameter identification needs to be pre-computed. Alternatively, adaptive control techniques could be used to identify the unknown parameters of a system in real time. This would be a much more useful appliction. The node placement algorithm in Section 3.4 could also be improved. An algorithm that takes into account the true dynamics of a cloth, and its curvature, could provide better node locations than an approximation made by analyzing beam deformations. Furthermore, the filtering algorithm itself could be improved. The current findings show that the multi sample approach taken by the CKF is effective at producing excellent results. This could further be improved by using a Monte-Carlo sampling approach, such as a particle filter [32]. The main issue with the particle filter is the added computation time, however, as is with the case of the CKF, modern CPUs with high thread counts can easily solve this by parallelizing the workload.

Future work on this project would be to take advantage of being able to physically change the materials of the surface being used. Certain materials, could be more aptly defined by the mass spring model used, thus leading to less modelling errors. To extend this further, a synthetic surface could be created by physically using spheres of known mass as nodes; and connecting these masses by springs to create a real life mass spring damper mesh. This system would be perfectly captured by a mass spring model and result in low errors.

References

[1] Derivative touchdesigner. http://www.madmapper.com/. Accessed: 2016-08-05.

[2] Guitar gently weeps love. https://ultimateclassicrock.com/beatles-while-my-guitar-gently-weeps-video/. Accessed: 2019-03-16.

[3] Madmapper. http://www.madmapper.com/. Accessed: 2016-08-05.

[4] Optitrack motion capture systems. http://www.http://optitrack.com/. Accessed: 2016-08-05.

[5] Throw distance calculator. https://files.support.epson.com/pdc/eai/flash/Index.html. Accessed: 2019-03-22.

[6] Tutorial 5 : A textured cube. http://www.opengl-tutorial.org/beginners-tutorials/tutorial-5-a-textured-cube/. Accessed: 2016-08-05.

[7] Bernt M Åkesson, John Bagterp Jørgensen, Niels Kjølstad Poulsen, and Sten Bay Jørgensen. A generalized autocovariance least-squares method for kalman filter tuning. *Journal of Process control*, 18(7-8):769–779, 2008.

[8] A Alipour and F Zareian. Study rayleigh damping in structures; unceratinties and treatments. In *Proceedings of 14th World Conference on Earthquake Engineering, Beijing, China*, 2008.

[9] Brian DO Anderson and John B Moore. *Optimal filtering*. Courier Corporation, 2012.

[10] Ienkaran Arasaratnam. *Cubature Kalman filtering theory & applications*. PhD thesis, McMaster University, 2009.

[11] Uri M Ascher, Steven J Ruuth, and Brian Wetton. *Implicit-explicit methods for time-dependent PDE's*. University of British Columbia, Department of Computer Science, 1993.

[12] David Baraff and Andrew Witkin. Large steps in cloth simulation. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pages 43–54. ACM, 1998.

[13] KJ Bathe and H Saunders. Finite element procedures in engineering analysis, 1984.

[14] Ted Belytschko, Yury Krongauz, Daniel Organ, Mark Fleming, and Petr Krysl. Meshless methods: an overview and recent developments. *Computer methods in applied mechanics and engineering*, 139(1):3–47, 1996.

[15] Jan Bender, Matthias Müller, Miguel A Otaduy, and Matthias Teschner. Position-based methods for the simulation of solid objects in computer graphics. Eurographics, 2013.

[16] Kiran S Bhat, Christopher D Twigg, Jessica K Hodgins, Pradeep K Khosla, Zoran Popović, and Steven M Seitz. Estimating cloth simulation parameters from video. In *Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 37–51. Eurographics Association, 2003.

[17] Gérald Bianchi, Barbara Solenthaler, Gábor Székely, and Matthias Harders. Simultaneous topology and stiffness identification for mass-spring models based on fem reference deformations. In *International Conference on Medical Image Computing and Computer-Assisted Intervention*, pages 293–301. Springer, 2004.

[18] Oliver Bimber and Bemd Frohlich. Occlusion shadows: using projected light to generate realistic occlusion effects for view-dependent optical see-through displays. In *Mixed and Augmented Reality, 2002. ISMAR 2002. Proceedings. International Symposium on*, pages 186–319. IEEE, 2002.

[19] Oliver Bimber and Ramesh Raskar. *Spatial augmented reality: merging real and virtual worlds.* CRC press, 2005.

[20] Oliver Bimber and Ramesh Raskar. Modern approaches to augmented reality. In *ACM SIGGRAPH 2006 Courses*, page 1. ACM, 2006.

[21] Eddy Boxerman and Uri Ascher. Decomposing cloth. In *Proceedings of the 2004 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 153–161. Eurographics Association, 2004.

[22] Stephen Boyd and Lieven Vandenberghe. *Convex optimization.* Cambridge university press, 2004.

[23] Derek Bradley, Tiberiu Popa, Alla Sheffer, Wolfgang Heidrich, and Tamy Boubekeur. Markerless garment capture. In *ACM Transactions on Graphics (TOG)*, volume 27, page 99. ACM, 2008.

[24] Robert Bridson, Sebastian Marino, and Ronald Fedkiw. Simulation of clothing with folds and wrinkles. In *Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 28–36. Eurographics Association, 2003.

[25] Morten Bro-Nielsen. Finite element modeling in surgery simulation. *Proceedings of the IEEE*, 86(3):490–503, 1998.

[26] Mauro Caresta. Vibrations of a free-free beam, 2014.

[27] Mark Carlson, Peter J Mucha, and Greg Turk. Rigid fluid: animating the interplay between rigid bodies and fluid. *ACM Transactions on Graphics (TOG)*, 23(3):377–384, 2004.

[28] Mark Carlson, Peter J Mucha, R Brooks Van Horn III, and Greg Turk. Melting and flowing. In *Proceedings of the 2002 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 167–174. ACM, 2002.

[29] Paul Michael Chapman and Derek PM Wills. An overview of physically-based modelling techniques for virtual environments. *Virtual Reality*, 5(3):117–131, 2000.

[30] Yunqiang Chen, Yong Rui, Thomas S Huang, et al. Multicue hmm-ukf for real-time contour tracking. *IEEE transactions on pattern analysis and machine intelligence*, 28(9):1525, 2006.

[31] Kwang-Jin Choi and Hyeong-Seok Ko. Stable but responsive cloth. In *ACM SIGGRAPH 2005 Courses*, page 1. ACM, 2005.

[32] Fred Daum. Nonlinear filters: beyond the kalman filter. *IEEE Aerospace and Electronic Systems Magazine*, 20(8):57–69, 2005.

[33] Oliver Deussen, Leif Kobbelt, and Peter Tücke. Using simulated annealing to obtain good nodal approximations of deformable bodies. In *Computer Animation and Simulation95*, pages 30–43. Springer, 1995.

[34] Bernhard Eberhardt, Andreas Weber, and Wolfgang Strasser. A fast, flexible, particle-system model for cloth draping. *IEEE Computer Graphics and Applications*, 16(5):52–59, 1996.

[35] Naser El-Sheimy, Eun-Hwan Shin, and Xiaoji Niu. Kalman filter face-off: Extended vs. unscented kalman filters for integrated gps and mems inertial. *Inside GNSS*, 1(2):48–54, 2006.

[36] T Ertl. Computer graphicsprinciples and practice. In *Data Acquisition and Analysis for Multimedia GIS*, pages 411–421. Springer, 1996.

[37] Keegan Fernandes, Adam Gomes, Cong Yue, Yousef Sawires, and David Wang. Surface prediction for spatial augmented reality. In *International Conference on Virtual, Augmented and Mixed Reality.* Springer, 2019.

[38] David Gaylor and E Glenn Lightsey. Gps/ins kalman filter design for spacecraft operating in the proximity of international space station. In *AIAA Guidance, Navigation, and Control Conference and Exhibit*, page 5445, 2003.

[39] Sarah FF Gibson and Brian Mirtich. A survey of deformable modeling in computer graphics. Technical report, Citeseer, 1997.

[40] Adam Gomes, Keegan Fernandes, and David Wang. Surface prediction for spatial augmented reality. In *International Conference on Virtual, Augmented and Mixed Reality*, pages 43–55. Springer, 2018.

[41] Adam Daniel Gomes. Prediction for projection on time-varying surfaces. Master's thesis, University of Waterloo, 2016.

[42] Vincent Granville, Mirko Křivánek, and J-P Rasson. Simulated annealing: A proof of convergence. *IEEE transactions on pattern analysis and machine intelligence*, 16(6):652–656, 1994.

[43] Mohinder S Grewal and Angus P Andrews. Applications of kalman filtering in aerospace 1960 to the present [historical perspectives]. *IEEE Control Systems Magazine*, 30(3):69–78, 2010.

[44] Eitan Grinspun, Anil N Hirani, Mathieu Desbrun, and Peter Schröder. Discrete shells. In *Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 62–67. Eurographics Association, 2003.

[45] Michael Hauth, Olaf Etzmuß, and Wolfgang Straßer. Analysis of numerical methods for the simulation of deformable models. *The Visual Computer*, 19(7-8):581–600, 2003.

[46] Anna Hilsmann, David C Schneider, and Peter Eisert. Realistic cloth augmentation in single view video under occlusions. *Computers & Graphics*, 34(5):567–574, 2010.

[47] Masaru Hisada, Keiko Yamamoto, Ichiroh Kanaya, and Kosuke Sato. Free-form shape design system using stereoscopic projector-hyperreal 2.0. In *2006 SICE-ICASE International Joint Conference*, pages 4832–4835. IEEE, 2006.

[48] Daisuke Iwai and Kosuke Sato. Document search support by making physical documents transparent in projection-based mixed reality. *Virtual Reality*, 15(2-3):147–160, 2011.

[49] Thomas Jakobsen. Advanced character physics. In *Game Developers Conference*, volume 3, 2001.

[50] Rolf Johansson. *System modeling and identification*. Prentice-hall, 1993.

[51] Rudolph Emil Kalman. A new approach to linear filtering and prediction problems. *Journal of basic Engineering*, 82(1):35–45, 1960.

[52] Alonzo Kelly. A 3d state space formulation of a navigation kalman filter for autonomous vehicles. Technical report, CARNEGIE-MELLON UNIV PITTSBURGH PA ROBOTICS INST, 1994.

[53] Marc D Killpack. Automated tracking and estimation for control of non-rigid cloth. *arXiv preprint arXiv:1403.1653*, 2014.

[54] Bojan Kocev, Felix Ritter, and Lars Linsen. Projector-based surgeon–computer interaction on deformable surfaces. *International journal of computer assisted radiology and surgery*, 9(2):301–312, 2014.

[55] Robert Kooima. Generalized perspective projection. *J. Sch. Electron. Eng. Comput. Sci*, 2009.

[56] TJ Lahey and GR Heppler. Mechanical modeling of fabrics in bending. *Journal of applied mechanics*, 71(1):32–40, 2004.

[57] John Denholm Lambert. *Numerical methods for ordinary differential systems: the initial value problem*. John Wiley & Sons, Inc., 1991.

[58] D Lee, Martin Morf, and Benjamin Friedlander. Recursive least squares ladder estimation algorithms. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 29(3):627–641, 1981.

[59] Bill Livolsi. What it is and why you should care, May 2015.

[60] Bryn Lloyd, Gábor Székely, and Matthias Harders. Identification of spring parameters for deformable object simulation. *IEEE Transactions on Visualization and Computer Graphics*, 13(5):1081–1094, 2007.

[61] Jean Louchet, Xavier Provot, and David Crochemore. Evolutionary identification of cloth animation models. In *Computer Animation and Simulation95*, pages 44–54. Springer, 1995.

[62] Augustus Edward Hough Love. Xvi. the small free vibrations and deformation of a thin elastic shell. *Philosophical Transactions of the Royal Society of London.(A.)*, (179):491–546, 1888.

[63] Kok-Lim Low and Adrian Ilie. Computing a view frustum to maximize an object's image area. *Journal of Graphics Tools*, 8(1):3–15, 2003.

[64] Kok-Lim Low, Greg Welch, Anselmo Lastra, and Henry Fuchs. Life-sized projector-based dioramas. In *Proceedings of the ACM symposium on Virtual reality software and technology*, pages 93–101. ACM, 2001.

[65] Nicholas Metropolis, Arianna W Rosenbluth, Marshall N Rosenbluth, Augusta H Teller, and Edward Teller. Equation of state calculations by fast computing machines. *The journal of chemical physics*, 21(6):1087–1092, 1953.

[66] Maryam Moafimadani, Adam Gomes, Karl Zabjek, Reinhard Zeller, and David Wang. *Haptic Training Simulator for Pedicle Screw Insertion in Scoliosis Surgery*, pages 301–311. Springer International Publishing, Cham, 2016.

[67] Gaku Narita, Yoshihiro Watanabe, and Masatoshi Ishikawa. Dynamic projection mapping onto deforming non-rigid surface using deformable dot cluster marker. *IEEE transactions on visualization and computer graphics*, 23(3):1235–1248, 2017.

[68] Andrew Nealen, Matthias Müller, Richard Keiser, Eddy Boxerman, and Mark Carlson. Physically based deformable models in computer graphics. In *Computer graphics forum*, volume 25, pages 809–836. Wiley Online Library, 2006.

[69] Robert E Nickell. Nonlinear dynamics by mode superposition. *Computer Methods in Applied Mechanics and Engineering*, 7(1):107–129, 1976.

[70] Brian J Odelson, Murali R Rajamani, and James B Rawlings. A new autocovariance least-squares method for estimating noise covariances. *Automatica*, 42(2):303–308, 2006.

[71] Ben Piper, Carlo Ratti, and Hiroshi Ishii. Illuminating clay: a 3-d tangible interface for landscape analysis. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 355–362. ACM, 2002.

[72] Lorenzo Pollini, Mario Innocenti, and Roberto Mati. Vision algorithms for formation flight and aerial refueling with optimal marker labeling. In *AIAA Modeling and Simulation Technologies Conference and Exhibit*, page 6010, 2005.

[73] Xavier Provot. Deformation constraints in a mass-spring model to describe rigid cloth behaviour. In *Graphics interface*, pages 147–147. Canadian Information Processing Society, 1995.

[74] Parinya Punpongsanon, Daisuke Iwai, and Kosuke Sato. Projection-based visualization of tangential deformation of nonrigid surface by deformation estimation using infrared texture. *Virtual Reality*, 19(1):45–56, 2015.

[75] Witawat Rungjiratananon, Yoshihiro Kanamori, and Tomoyuki Nishita. Chain shape matching for simulating complex hairstyles. In *Computer graphics forum*, volume 29, pages 2438–2446. Wiley Online Library, 2010.

[76] Manika Saha, Bhaswati Goswami, and Ratna Ghosh. Two novel costs for determining the tuning parameters of the kalman filter. *arXiv preprint arXiv:1110.3895*, 2011.

[77] Mark Segal, Carl Korobkin, Rolf Van Widenfelt, Jim Foran, and Paul Haeberli. Fast shadows and lighting effects using texture mapping. In *ACM Siggraph Computer Graphics*, volume 26, pages 249–252. ACM, 1992.

[78] Jürgen Steimle, Andreas Jordt, and Pattie Maes. Flexpad: highly flexible bending interactions for projected handheld displays. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 237–246. ACM, 2013.

[79] Rahul Sukthankar, Tat-Jen Cham, and Gita Sukthankar. Dynamic shadow elimination for multi-projector displays. In *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, volume 2, pages II–151. IEEE, 2001.

[80] Endre Süli and David F Mayers. *An introduction to numerical analysis*. Cambridge university press, 2003.

[81] Gabriel A Terejanu. Unscented kalman filter tutorial. *University at Buffalo, Buffalo*, 2011.

[82] Matthias Teschner, Bruno Heidelberger, Matthias Muller, and Markus Gross. A versatile and robust model for geometrically complex deformable solids. In *Computer Graphics International, 2004. Proceedings*, pages 312–319. IEEE, 2004.

[83] Sebastian Thrun. Particle filters in robotics. In *Proceedings of the Eighteenth conference on Uncertainty in artificial intelligence*, pages 511–518. Morgan Kaufmann Publishers Inc., 2002.

[84] Aydin Varol, Mathieu Salzmann, Engin Tola, and Pascal Fua. Template-free monocular reconstruction of deformable surfaces. In *2009 IEEE 12th International Conference on Computer Vision*, pages 1811–1818. IEEE, 2009.

[85] Julien Villard and Houman Borouchaki. Adaptive meshing for cloth animation. *Engineering with Computers*, 20(4):333–341, 2005.

[86] Eric A Wan and Rudolph Van Der Merwe. The unscented kalman filter for nonlinear estimation. In *Adaptive Systems for Signal Processing, Communications, and Control Symposium 2000. AS-SPCC. The IEEE 2000*, pages 153–158. Ieee, 2000.

[87] Yanzhen Wang, Yueshan Xiong, Kai Xu, Ke Tan, and Guangyou Guo. A mass-spring model for surface mesh deformation based on shape matching. In *GRAPHITE*, volume 6, pages 375–380, 2006.

[88] Greg Welch, Gary Bishop, et al. An introduction to the kalman filter. 1995.

[89] Scott Whitney. Vibrations of cantilever beams: Deflection, frequency, and research uses. *Website: Apr*, 23(10), 1999.

[90] Yongning Zhu and Robert Bridson. Animating sand as a fluid. In *ACM Transactions on Graphics (TOG)*, volume 24, pages 965–972. ACM, 2005.