Prediction for Projection on Time-Varying Surfaces

by

Adam Daniel Gomes

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, 2016

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

In spatial augmented reality applications, when video projectors display images on time-varying, non-planar surfaces, rather than on flat, rigid surfaces, undesired image distortion may occur. For applications where realism is of the utmost importance, such as surgical simulations, image distortion can significantly detract from the user experience. To combat this, the time-varying surface can be modelled using a mass-spring model, commonly used for simulating deformable objects in computer graphics. The mass-spring model can be formulated into a nonlinear state space equation that describes the dynamics of discrete points making up the surface of the object. Two simulation techniques are used to verify the model and to determine the best approach for real-time simulations.

To project images in real-time onto quickly changing surfaces, an extended Kalman filter (EKF) prediction algorithm is developed to predict the position of the deforming surface, at a specified point in time, T_s seconds, in the future. Using the linearized mass-spring system, the EKF is formulated and tested upon two simulation scenarios. The simulation scenarios include a falling cloth with added process noise, and a cloth perturbed by random viscous forces. Using mean squared error, the results show the EKF predictions and simulation outputs converge within a narrow band. For each scenario, the parameters of the EKF are manually tuned to improve the accuracy of the predictions. Experimental data is collected by measuring the movement of cloth-like materials to verify the effectiveness of the prediction algorithm. Specifically, cloth movement data is captured using infra-red markers and motion capture software. The EKF prediction algorithm is run on the experimental data producing near convergent results between the predictions and the measurements.

When the physical surface is changing noticeably and quickly, compared to the projector's drawing rate, additional distortion may occur. An inter-frame prediction algorithm is developed to further predict the position of discrete points at their corresponding projection times. This is most useful when the prediction algorithm produces predictions slower than the drawing rate of the projector ($T_s > \frac{1}{\text{fps}}$).

When implementing the EKF in real-time, there is a trade-off between speed and accuracy. If the number of discrete points is large, the EKF is required to solve a large system of equations. To combat this, nonlinear optimization techniques are used to find parameters that reduce the number of states while maintaining system dynamics. This results in a sparser, more computationally efficient model with similar physical behaviour to the original system.

Applications for time-varying surface prediction include surgical simulations, projection for entertainment and advertising, and other spatial augmented reality applications.

Acknowledgements

I would like to first thank my advisor Professor David Wang for his guidance, advice and feedback. I would also like to thank my research group, particularly Madeleine Wang for assisting with experimental data collection. Lastly, I would like to thank my friends and family who have helped me through this journey.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

In spatial augmented reality applications, projection of images onto non-rigid surfaces can pose many issues. As the surface geometry is not necessarily stationary, standard projection techniques can fail to create a realistic experience due to improper image mapping. For applications where realism is of great importance, such as surgical simulators, this can effect how well a user can perform their intended task. To use projection in these situations, the position of the deformable object's surface must be tracked. There are a number of publications that have studied tracking and projection onto non-rigid surfaces; however, for quickly changing surfaces, there is no mention of how well these techniques perform. When the surface being projected onto is moving quickly, the computational time of processing images, in addition to surface tracking, may cause distorted images to be projected. To combat this, a prediction scheme can be used to approximate the position of the surface at the time of projection. Using this predicted surface, images can be processed in advance, resulting in a smoother experience for the user.

To implement a prediction scheme for surface tracking, a physically accurate deformable model needs to be used. A large number of deformable models have been studied in the field of computer graphics. These models range from aesthetically pleasing models to physically accurate models built from a theoretical foundation. Using deformable models, tracking of object surfaces has been used in the fashion and movie industries [20]. Since these industries require a high level of realism in their models, tracking is done post production rather than in real-time. Aesthetically pleasing, real-time models have been used for video games; however, these models often times lack physical realism. Even though surface tracking and estimation using deformable models in real-time seems like an obvious area of research in spatial augmented reality, very few papers have studied it. One example is Killpack [42], where he examines and tracks the motion of cloth in conveyor belts. Using

video data and a physical model for cloth, Killpack's goal is to eventually be able to control the surface behaviour of the belt.

An application where time-varying surface prediction would be very useful is in surgical simulators. To make surgical simulators more realistic, the visual experience should closely match what the surgeon sees while performing surgery. Many studies have looked at graphically representing the surface of organs using physically accurate models [22]. When using projector based spatial augmented reality for the visuals, the malleable surface of organs require surface tracking for accurate projection. Surface tracking can be combined with these models to accurately recreate visuals using a projector setup (Figure 1.1). Moafimadani et al. [50] created a haptic device to simulate the tactile sensations that surgeons feel during pedicle screw insertion surgery for scoliosis. To make the simulator more compelling for surgical training, a visual component is being developed using projectors. Since the surgeon applies tremendous amounts of force on the patient during the procedure, the skin will move quite drastically from its resting position. As a result, for the surgical simulator, the surface geometry must be able to change to match the experience of the real surgery. Surface prediction can be applied to the simulator to account for these surface changes, allowing the visuals to match the real life surgery, making the simulator far more suitable for use as a real training device.

Figure 1.1: Surgical simulator projector setup

Another application where time-varying surface prediction can be used is in the entertainment industry, especially during live projection shows. For example, in Figure 1.2, the singer is wearing a long white dress being projected upon by a video. The projection takes

advantage of the dress' geometry, making it a focal point of the performance. The projectors are calibrated to project onto the fixed surface of the dress, and as a result, issues may arise if the performer moves during the show. When the performer moves, the dress, and therefore the display surface, changes shape. The images may no longer align with the geometry, ruining the experience. By using surface estimation, the surface geometry can be continuously accounted for during projection, resulting in a fluid show. Furthermore, additional effects can be added to the performance that take advantage of the changing display surface geometry. Images can be projected onto surfaces changing due to wind, or even onto fluids to create a more immersive atmosphere for the audience.

Figure 1.2: Singer wearing dress being projected onto [4]

The following chapters guide the reader through the contributions and relevant details of this thesis. 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, providing the model formulation, model linearization and simulation results. Chapter 4 reviews common filtering approaches and discusses how suitable these techniques are to deformable model estimation. Chapter 5 presents techniques for reducing the order of the model so that the prediction filter can run more efficiently. Chapter 6 discusses how the extended Kalman filter can be combined with the mass spring model to create a deformable surface prediction filter. The prediction filter is applied to both simulation and experimental scenarios, and the results are discussed. Also in Chapter 6, an additional prediction scheme is developed to compensate for surface movement during image projection. Lastly, Chapter 7 summarizes the results of the experiments and presents topics for future research.

Chapter 2

Background

To create a model for deformable surfaces for augmented reality applications, a number of research areas need to be explored. This chapter provides a review of the techniques used in computer graphics to simulate deformable objects, so that a physically realistic deformable model can be used in augmented reality applications. Also in this chapter, past and current applications of projector based augmented reality, along with their advantages and disadvantages, will be discussed. Works related to non-rigid object projection will also be reviewed. Lastly, projection mapping on three-dimensional surfaces will be discussed.

2.1 Deformable Models

The simulation of solid objects such as rigid bodies, soft bodies or cloth has been an important and active research area in computer graphics since the late 1980's [31]. Early works focused on non-physical models of deformable objects, where purely geometric deformation techniques were employed. These techniques are extremely efficient; however, they rely on the skill of the designer rather than on the physical properties of the object [31]. The designer would generally have to have prior knowledge of how the objects behave, making these techniques just as aesthetic as analytical. One of the main approaches in this field is the free-form deformation (FFD) technique. The FFD technique is a general method for deforming objects in which the shape of object is changed by deforming the space in which the object lies. Linear transformations of the space are composed together to provide complex deformations; however, the user's control of deformations are not very intuitive. Non-physical methods for modelling deformations are limited by the expertise and patience of the user. Deformations are explicitly specified and the system has no knowledge of the

nature of the objects being manipulated. As a result, modelling complex structures with non-uniform shapes are almost impossible using non-physical models.

The most common approaches for deformable body simulations are called the classical dynamic simulation methods (or physical models). These approaches formulate the change of momentum of a system as a function of applied forces, and determine positions through numerical integration of accelerations and velocities. These techniques are well studied, and as a result, are well established in computer graphics. The main methods in this area are the mass spring model method and the finite-element model (FEM) method. The mass spring model method is a technique that has been used widely and effectively for modelling deformable objects. An object is modelled as a collection of point masses connected by springs and dampers in a specific orientation. Newton's Second Law, which governs the motion of masses when forces are applied, determines the position of each point mass after every time-step in the simulation. A more detailed analysis of this method can be found in Chapter 3. Mass spring systems are simple physical models which are easy to construct and very fast to compute. They are so useful that they are still being used for simulations of deformable bodies in new application areas such as virtual and augmented reality systems [51]. One of the major drawbacks of this method is that the point masses are chosen discretely throughout the object. This causes the simulation to be an approximation, rather than a true simulation of continuous bodies. Additionally, it is quite difficult to determine the optimal mass, spring, and damper parameters to best match real-world objects. For most applications, these parameters are chosen so that the visual appearance is pleasing. As a result, mass spring models often say little about the material properties of the object being modelled [51]. However, more recent studies have found close-to-optimal parameter sets for mass spring models using parameter identification techniques or physical properties of the real-world material [61][21][33]. Teschner [61] employs generlized springs that preserve distances, areas and volumes. Bridson[21] presents a physically correct bending model for triangle meshes. Grinspun [33] presents a similar model for simulating discrete shells. Eberhardt [28] and Choi [26] improve the realism by modelling nonlinear material properties. Eberhardt base their spring model on measured cloth data to model hysteresis and Choi approximate cloth buckling using a fifth-order polynomial. Lahey [44] models the bending effect of fabrics including nonlinear elasticity and viscous and Coulomb friction with hysteric effects.

Numerical stiffness can occur in the model when spring constants are chosen to be too large. This is a well known problem in mathematics [11]. A system is numerically stiff if a numerical method is forced to use a steplength which is excessively small in relation to the smoothness of the exact solution [45]. For linear differential equations, numerical solving is unstable if the step time, ΔT, is chosen to be greater than the natural period of the

system. The natural period, T_0 of a linear system is given by

$$T_0 \propto \pi\sqrt{\frac{m}{k}}, \tag{2.1}$$

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

$$k_c \propto m\frac{\pi^2}{T_0^2}. \tag{2.2}$$

The critical stiffness value, k_c, is the spring constant value at which the system becomes numerically unstable at a step time $\Delta T = T_0$. Therefore, the only way to increase the maximum value of k, is to decrease the value of ΔT. As a result, to obtain convergent simulation results, small integration step times need to be chosen, resulting in far more calculations. If integration step times are chosen to be too large, the simulation behaviour will be divergent and unstable. A number of studies have found ways around the numerical stiffness problem. Provot [54] avoided these issues by applying constraints to the movement of each point mass. This technique has been shown to significantly improve the numerical stability of the model; however, it detracts from the physical realism of the model. Ascher [10] proposes using alternate integration techniques, such as implicit integration schemes to improve the likelihood of convergence of the model. Hauth [34], Boxerman [18] and Ascher[10] use both implicit and explicit integration schemes; implicit integration is used on the stiff portions of the model and explicit integration is used on the non-stiff parts. This allows for a balance of stability, from the implicit scheme, and simplicity, from the explicit scheme. Overall, mass spring models have generally been the de facto deformable body simulation technique used in practice due to its ease of formulation and computational efficiency.

Models that treat objects as continuous bodies, rather than discrete points, are called continuum methods. The continuum model of a deformable object considers the equilibrium of a general body acted on by external forces. The equilibrium configuration of an object is the undeformed, resting shape of the object. The object reaches equilibrium when its potential energy is at a minimum. As a result, the deformation is a function of these forces and the object's material properties. The partial differential equation (PDE) governing the dynamics of elastic materials is given by [51],

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

where ρ is the material density, $\mathbf{x}$ is the material position as a function of (x, y, z), σ is the 3×3 stress tensor and $\mathbf{f}_{ext}$ is a vector of externally applied forces as a function of (x, y, z).

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, $\mathbf{x}(t)$, to Equation (2.3) for an entire body, FEM methods are used. The finite element method turns a PDE into a set of algebraic equations that can be solved numerically. The method takes the continuous domain and discretizes 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 approximated by linear combination of these discrete solutions, that is [51],

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

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. However, in the most general case of the FEM, this may not be true. Substituting $\tilde{\mathbf{x}}(t)$ into Equation (2.3) for $\mathbf{x}(t)$ results in a series of equations solving for each $\mathbf{x}_i(t)$. 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 $\mathbf{M}$ is the mass matrix, $\mathbf{D}$ is the damper matrix, $\mathbf{K}$ is the stiffness matrix, $\mathbf{F}$ is the applied force vector and $\mathbf{u}$ is node displacement vector. The matrix $\mathbf{M}$ is generally found

by integrating over the volume of each element, and **D** is a linear combination of both **M** and **K**, in a method called Rayleigh Damping [8]. Often times, to save on computations, diagonal matrices are used for **M** and **D** in a method called mass-lumping. In this method, **M** contains the mass of each node along its diagonal.

Finite element methods provide more physically realistic simulations than mass spring systems with fewer node points. However, due to 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 [31].

More contemporary methods for deformable body simulations include the position-based methods. Position-based methods compute positions directly, based on the solution to a quasi-static problem [13]. This is in contrast to the classical methods which evolve positions through numerical integration of accelerations. Position based methods use constraint functions to limit the behaviour of each particle, while explicit forward Euler integration is used to solve for the positions at the next time step. This method recreates the behaviour of a second-order system, but it needs no acceleration data. The position-based methods are mainly used in applications where performance, controllability and stability are more important than accuracy. These methods have been used in surgical simulations [64], and hair style simulations [56]. Other recent approaches for deformable modelling include the Mesh-free methods [12]. Mesh-free methods are an extension of the continuum approaches for modelling, where surfaces are constructed entirely out of nodes rather than meshes. This allows for discontinuities in the model to have little effect on simulation accuracy, making it possible to solve large classes of problems ranging from stiff structures to fluids. Lastly, there are the Eulerian methods for deformable body simulations. All previously mentioned methods fall under the category of Lagrangian methods, where an object is described as a set of moving points that change position over time [51]. Eulerian methods looks at a stationary set of points and calculates how the material properties stored at those points change over time. The Eulerian methods are mainly used for fluid simulations, as their formulation solves the Navier-Stokes equations [65][24][23]. Using these methods, Zhu [65] were able to model sand as a fluid, Carlson [24] modelled solid objects that can melt into liquids and Carlson [23] were able to model rigid body interactions with fluids. Most contemporary methods have seen little application as they require too much computational power[23]. Although newer techniques can create more visually compelling simulations, they can take hours or days to produce results, and therefore cannot be used for real-time applications.

2.2 Spatial Augmented Reality

Spatial Augmented Reality (SAR) is a technological variation of Augmented Reality (AR) where large, spatially aligned optical elements, such as transparent screens or video projectors, are used as the display medium [17]. Within SAR, projector-based augmentation integrates augmented reality directly into the user's environment, rather than simply into their visual field. To do this, projector based displays apply front-projection to project images directly onto physical objects instead of onto an image plane [17]. Since projectors traditionally project on rectangular flat surfaces, projection on non-planar surfaces require special rendering process. For images to be projected properly onto an object's surface, texture mapping needs to be performed on virtual models that closely resemble the object to be projected upon. Texture mapping allows an image to be "painted" on an object by defining transformations from the desired image coordinates to the coordinates of the virtual object. The texture mapped virtual object is then projected onto the real world object to create the illusion of alternate lighting, texture and material properties. Generally, projector based augmentation works best when the geometry of virtual objects closely match that of the physical objects. To match the geometry of both objects, a CAD model of the real world object is most often used. The textured virtual object is then projected on materials such as styrofoam, cardboard, or any material with neutral colour and texture. Many software packages exist that specialize in projector based SAR such as MadMapper[5] and Derivative TouchDesigner[3]. Projector based spatial displays overcome the shortcomings of AR displays as they provide improved ergonomics, a scalable resolution and easier eye accommodation. On the negative side, projector based spatial displays are prone to shadows and occlusion, they are constrained to the size and shape of physical objects being projected on, and they are difficult to calibrate and keep aligned.

Projector based augmentation has been used in many artistic and advertising ventures. Examples of projector based SAR include the Being There project developed at UNC [48], where researchers built a walk-through of a virtual building by projecting architectural objects such as columns, cupboards and tables onto simple shapes made of styrofoam. The experiment showed that projector augmentation provides a stronger sense of immersion when compared to other displays. More advanced applications of projector based augmentation include projection onto dynamic mechanical models and projection onto deformable surfaces. In [17], the researchers project images on a model of a car to create the illusion of motion. Rather than have the car move in the environment, the projections change to create the illusion of a moving vehicle. By using the dynamic effects of acceleration, the researchers warp images to match the expected shape and colouration of a vehicle in motion. Adding additional effects such as induced shadows and lighting helped improve the

realism of the demonstration. Commercial applications of projection based SAR include the projection of product images on building facades for advertisement. Large companies such as Nokia, Samsung and BMW have used SAR to promote their products in major cities. Other applications of projection based AR include appearance enhancement [38] and creating the illusion of deformations on rigid surfaces [37].

Another area of research within projector based augmented reality is occlusion correction. Occlusion correction studies how to compensate for unwanted optical interference effects and how to artificially create shadows that would exist if the projections were real objects. Sukthankar [59] addresses the problem of undesired shadows by redundantly illuminating the display surface using multiple projectors in different locations. Additionally, the researchers use cameras to identify occlusions as they occur and dynamically adjust the projected image. Punpongsanon [55] uses feature tracking techniques, where occluded features are ignored on each frame, to reduce projection error. Steimle [58] detects body parts, such as hands and fingers, and accounts for them during image projection. On the other hand, often times it is advantageous to create occlusions, such as virtual shadows, to increase the realism of the SAR scenario. For example, to create virtual occlusions, Bimber [16] uses computer controlled projectors to create lighting on a per-pixel basis, allowing for artificial, view-dependent shadows to be "projected" onto real objects.

2.2.1 Non-Rigid Projection Based AR

For the most part, projection based AR research 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 [53]. 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 [43], 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 [62], optical flow [36], and object contour analysis [25]. In a very recent study, [55] uses non-rigid materials such as silly putty, sodium borate and clay as surfaces, painted with infra-red paint, to project images on the changing surface geometry. Their method determines tangential deformations, like kneading and stretching, from feature extraction by tracking the motion of the infra-red markers. However, since the infra-red camera must be placed directly above the surface, only two-dimensional motion can be measured. Also in a very recent study, [58] created a system where sheets of paper can be used as handheld displays. If the paper

is slightly bent, the deformation is estimated using a Kinect, and the projection is warped to fit the new shape of the paper, making it a spatially aware display. This work does not examine tangential deformation, such as stretching, and does not comment on how the speed of deformations affects performance. They do admit that when working with fully flexible materials, only simple deformations can be tracked.

2.2.2 Projection Mapping

Traditionally, projectors are used to create flat and rectangular images, and as a result, rendering algorithms used for LCD panels can be used without any modification. However, in SAR applications, where the display surface can vary geometrically, alternate rending approaches have been proposed. 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.

Display Surface

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. The advantage of defining surfaces this way is that graphics hardware use polygons and the associated normals as geometric primitives (OpenGL). There are many techniques that can be used to generate the display surface. Common techniques for obtaining geometry is by using depth cameras or multiple camera set-ups. Bimber [16] proposes using triangulation techniques based on correspondences extracted from the images from two cameras. Corresponding points are determined by sequentially illuminating points until a three-dimensional point cloud is created. This technique is called the active structured light technique. Other techniques include capturing discrete points of data using markers and interpolating to create a dense mesh, and using video data, combined with machine learning, to approximate the 3D shape of an object.

Projector Model

The projector model can be approximated by a pin-hole model, similar to a pin-hole camera model. As explained by Bimber [16], 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 in homogeneous coordinates [16] 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 external parameters (transformation matrix converting the world coordinate system to the projector coordinate system), and F is the projector's intrinsic matrix defined as

$$F = \begin{bmatrix} \alpha & \gamma & u_0 \\ 0 & \beta & v_0 \\ 0 & 0 & 1 \end{bmatrix}, \tag{2.9}$$

where (u_0, v_0) are the coordinates of the principal point which is the point where the principal plane crosses the optical axis, usually the centre of the image, α and β represent the focal length in terms of pixels, and γ represents the skew between the u and v axes. 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. Assuming the display surface has been computed, the mapping between each projector pixel and each surface point is known. As a result, the projection matrix P can easily be found.

Rendering

To render an image onto a surface, two steps are required [16]. The first step involves producing a texture map of the desired image to be projected. The second step is projecting the texture onto the the polygonal model of the display surface. Projection of a two-dimensional texture onto a three-dimensional model is done through the process of UV mapping. UV mapping defines a coordinate system (u, v) for the texture and a coordinate system (x, y, z) to 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. Additional techniques, such as shading, specifically Phong shading, can be applied to

texture mapping to increase the level of realism. Phong shading is an interpolation technique that computes pixel colours based on the surface normals and a specified reflection model [30]. 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 2.2 shows a 3D printed replica torso projected with a red and white checkerboard pattern rendered upon it using projection based AR (using the TouchDesigner software). The checkerboard pattern itself is shown in Figure 2.1.

Figure 2.1: Checkerboard pattern [2]

Figure 2.2: SAR example of checkerboard rendered onto torso

Chapter 3

Derivation of Mass-Spring-Damper Model

As described by Provot in [54], a realistic model of cloth can be assembled by interconnecting a set of point masses (nodes) with springs and dampers. This model represents a two dimensional surface in three dimensional space. Each point mass in the model is connected to its neighbouring nodes above, below, to the left and to the right with structural springs (and dampers); to its diagonal nodes with shear springs (and dampers), and to the nodes 2 elements away with flexion springs (and dampers). Structural springs keep the mesh together when pure compression or traction (stretching) stresses are applied. Shear springs prevent the model from collapsing diagonally when shear stresses are applied. Flexion springs prevent the cloth from folding onto itself due to pure flexion stresses. Figure 3.1 shows a square 3×3 node cloth system where all connecting springs are drawn. In this arrangement, the structural, shear, and flexion springs can easily be seen emanating from each node.

In this Chapter, the mass spring model will be developed in state space form, will be linearized, and simulated for implementation into estimation filters.

3.1 State Space Formulation

The mass spring cloth model can be described by a first order differential equation of the form,

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

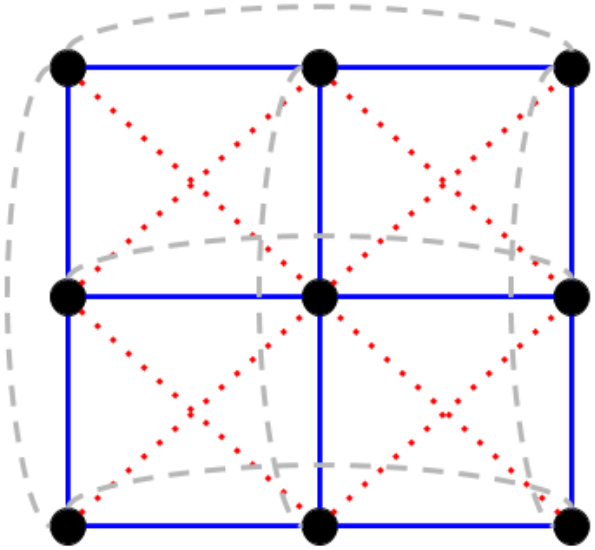


Figure 3.1: Connection of mass nodes with structural springs (blue), shear springs (red dashed), and flexion springs (grey dashed)

Here $x \in \mathbf{R}^{6n}$ is the state vector containing the position $p_i \in \mathbf{R}^3$ and velocity $v_i \in \mathbf{R}^3$ of each point mass (node) i in Euclidean space, for all $i = 1, 2, \ldots, n$, 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 also be written 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 $\dot{\mathbf{p}} = [v_1^T \ldots v_n^T]^T$. n is the number of point masses in the system and $f(\cdot) : \mathbf{R}^{6n} \to \mathbf{R}^{6n}$ is a sufficiently smooth nonlinear state transition function, i.e., $f(\cdot) \in C^1$. The function $f(\cdot)$ is required to be differentiable as the Jacobian of the system is required for both numerical integration and the filter formulation. The mass spring model is a time invariant system, and therefore, the state transition function can

be written as $f(x) = f(x,t)$. The state derivative vector can also be written as

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

Here $a_i(\cdot) : \mathbf{R}^{6n} \to \mathbf{R}^3$ represents the acceleration of each node. The resulting state derivative vector is composed of the point velocities concatenated with the point accelerations. Using Newtonian mechanics, the point accelerations are found to be

$$a_i(x) = \frac{1}{m_i}(m_i\vec{g} + \sum_{j \in \mathcal{A}_i} F_j(p_j, v_j)), \tag{3.5}$$

for $i = 1 \dots n$. In Equation (3.5), F_j is the force applied onto point mass m_i by point mass m_j, and $\vec{g}$ is the acceleration due to gravity (i.e. $\vec{g} = \begin{bmatrix} 0 & -9.81 & 0 \end{bmatrix}^T \text{m/s}$). 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 $\mathcal{A}_i$ by both a spring and damper, the internal force on a node i can be written in the form of a second order differential equation:

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

From this, the force F_j caused by each node j on node i has the form:

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

In Equation (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 portion of Equation (3.7),

$$k_{s_{ij}}(\|p_i - p_j\|_2 - r_{ij})\frac{p_i - p_j}{\|p_i - p_j\|_2},$$

is derived from Hooke's law, Figure 3.2, where the constant $r_{ij} \in \mathbf{R}$ represents the resting length of the spring between nodes i and j. The force caused by the spring is applied in the opposite direction to the vector created between points i and j when stretched, and in the same direction as the vector when compressed. The damping force is always

applied in the opposite direction of the velocity vector created between points i and j. The spring and damper coefficients are assumed to be linear constants for this model. Models with nonlinear spring and damper coefficients may create more physically realistic simulations [26]; however, their characteristics are difficult to determine. Due to the specific arrangement of springs and dampers, the size of the connected node set $\mathcal{A}_i$ can range from node to node due to the positioning of the node. For example, a corner node will always have two fewer structural springs, three fewer shear springs and two fewer flexion springs than a center node in a high order system. It can easily be seen that a single node can be connected to at most 12 other points, and as little as 3 other points. This implies that the size of the connected points set, $\mathcal{A}_i$, can range from 3 to 12 elements.

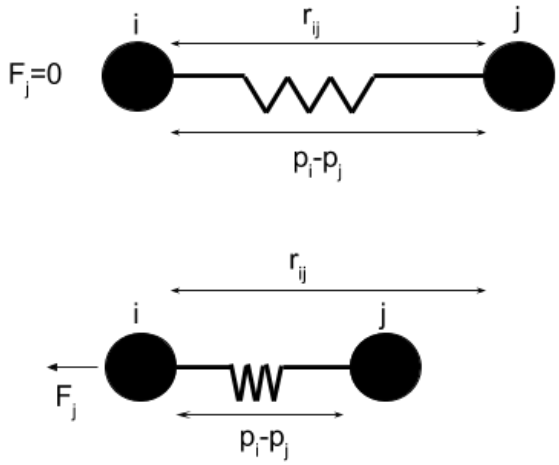


Figure 3.2: Illustration of spring force between two nodes i and j

To illustrate this model derivation, a simple 2×2 node mass spring model is formulated.

The model has the initial conditions at time t_0

$$x_0 = \begin{bmatrix} \mathbf{p}(t_0) \\ \dot{\mathbf{p}}(t_0) \end{bmatrix} \quad \mathbf{p}(t_0) = \begin{bmatrix} p_1(t_0) \\ p_2(t_0) \\ p_3(t_0) \\ p_4(t_0) \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} \quad \dot{\mathbf{p}}(t_0) = \begin{bmatrix} v_1(t_0) \\ v_2(t_0) \\ v_3(t_0) \\ v_4(t_0) \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

as shown in Figure 3.4, and parameters $m_i = 1\text{kg}$ for all four nodes, $k_{s_i} = 100\text{N/m}$ and $k_{d_i} = 0.1\text{N·s/m}$ for all six node connections. The initial conditions imply the rest length, r_{ij}, for all structural springs to be 1m and all shear springs to be $\sqrt{2}$m.

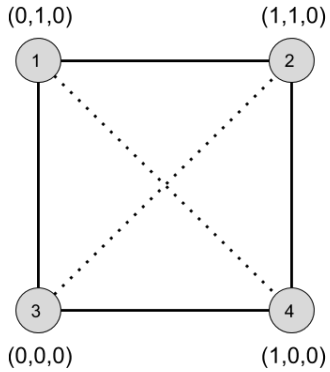


Figure 3.3: Initial positions of 2×2 example mesh

Now assume the model is at time $t_1 > t_0$, where the state vector at time t_1 is

$$x(t_1) = \begin{bmatrix} \mathbf{p}(t_1) \\ \dot{\mathbf{p}}(t_1) \end{bmatrix} \quad \mathbf{p}(t_1) = \begin{bmatrix} p_1(t_1) \\ p_2(t_1) \\ p_3(t_1) \\ p_4(t_1) \end{bmatrix} = \begin{bmatrix} 0 \\ 2 \\ 0 \\ 1 \\ 2 \\ 0 \\ 0 \\ 0 \\ 1 \\ -1 \\ 0 \end{bmatrix} \quad \dot{\mathbf{p}}(t_1) = \begin{bmatrix} v_1(t_1) \\ v_2(t_1) \\ v_3(t_1) \\ v_4(t_1) \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0.5 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}.$$

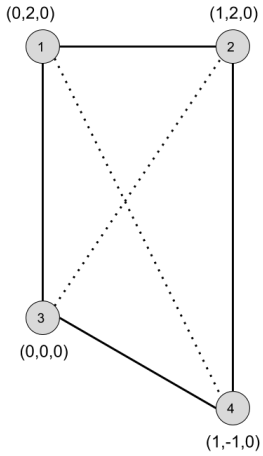


Figure 3.4: Configuration of 2×2 example mesh at time t_1

The state derivative vector at time t_1, $\dot{x}(t_1)$, can be found by using Equation (3.4). The first 12 elements of $\dot{x}(t_1)$ are simply $\dot{\mathbf{p}}(t_1)$ and the next 12 elements are found using Equation (3.5).
Node 1:

$$\mathcal{A}_1 = \{2, 3, 4\},$$

$$F_2 = -k_{s_{12}}(\|p_1 - p_2\|_2 - r_{12})\frac{p_1 - p_2}{\|p_1 - p_2\|_2} - k_{d_{12}}(v_1 - v_2) = \begin{bmatrix} 0.1 \\ 0 \\ -0.05 \end{bmatrix} \text{N}$$

$$F_3 = -k_{s_{13}}(\|p_1 - p_3\|_2 - r_{13})\frac{p_1 - p_3}{\|p_1 - p_3\|_2} - k_{d_{13}}(v_1 - v_3) = \begin{bmatrix} 0 \\ -100 \\ -0.05 \end{bmatrix} \text{N}$$

$$F_4 = -k_{s_{14}}(\|p_1 - p_4\|_2 - r_{14})\frac{p_1 - p_4}{\|p_1 - p_4\|_2} - k_{d_{14}}(v_1 - v_4) = \begin{bmatrix} 55.38 \\ -165.84 \\ -0.05 \end{bmatrix} \text{N}$$

$$a_1(t_1) = \frac{1}{m_1}(m_1\vec{g} + F_2 + F_3 + F_4) = \begin{bmatrix} 55.48 \\ -275.65 \\ -0.15 \end{bmatrix} \frac{\text{m}}{\text{s}^2}$$

Node 2:

$$\mathcal{A}_1 = \{1, 3, 4\},$$

$$F_1 = -k_{s_{21}}(\|p_2 - p_1\|_2 - r_{21})\frac{p_2 - p_1}{\|p_2 - p_1\|_2} - k_{d_{21}}(v_2 - v_1) = \begin{bmatrix} -0.1 \\ 0 \\ 0.05 \end{bmatrix} \text{N}$$

$$F_3 = -k_{s_{23}}(\|p_2 - p_3\|_2 - r_{23})\frac{p_2 - p_3}{\|p_2 - p_3\|_2} - k_{d_{23}}(v_2 - v_3) = \begin{bmatrix} -36.85 \\ -73.51 \\ 0 \end{bmatrix} \text{N}$$

$$F_4 = -k_{s_{24}}(\|p_2 - p_4\|_2 - r_{24})\frac{p_2 - p_4}{\|p_2 - p_4\|_2} - k_{d_{24}}(v_2 - v_4) = \begin{bmatrix} 0 \\ -200 \\ 0 \end{bmatrix} \text{N}$$

$$a_2(t_1) = \frac{1}{m_2}(m_2\vec{g} + F_1 + F_3 + F_4) = \begin{bmatrix} -36.95 \\ -283.32 \\ 0.05 \end{bmatrix} \frac{\text{m}}{\text{s}^2}$$

Node 3:

$$\mathcal{A}_1 = \{1, 2, 4\},$$

$$F_1 = -k_{s_{31}}(\|p_3 - p_1\|_2 - r_{31})\frac{p_3 - p_1}{\|p_3 - p_1\|_2} - k_{d_{31}}(v_3 - v_1) = \begin{bmatrix} 0 \\ 100 \\ 0.05 \end{bmatrix} \text{N}$$

$$F_2 = -k_{s_{32}}(\|p_3 - p_2\|_2 - r_{32})\frac{p_3 - p_2}{\|p_3 - p_2\|_2} - k_{d_{32}}(v_3 - v_2) = \begin{bmatrix} 36.85 \\ 73.51 \\ 0 \end{bmatrix} \text{N}$$

$$F_4 = -k_{s_{34}}(\|p_3 - p_4\|_2 - r_{34})\frac{p_3 - p_4}{\|p_3 - p_4\|_2} - k_{d_{34}}(v_3 - v_4) = \begin{bmatrix} 29.39 \\ -29.29 \\ 0 \end{bmatrix} \text{N}$$

$$a_3(t_1) = \frac{1}{m_3}(m_3\vec{g} + F_1 + F_2 + F_4) = \begin{bmatrix} 66.24 \\ 134.41 \\ 0.05 \end{bmatrix} \frac{\text{m}}{\text{s}^2}$$

Node 4:

$$\mathcal{A}_1 = \{1, 2, 3\},$$

$$F_1 = -k_{s_{41}}(\|p_4 - p_1\|_2 - r_{41})\frac{p_4 - p_1}{\|p_4 - p_1\|_2} - k_{d_{41}}(v_4 - v_1) = \begin{bmatrix} -55.38 \\ 165.84 \\ 0.05 \end{bmatrix} \text{N}$$

$$F_2 = -k_{s_{42}}(\|p_4 - p_2\|_2 - r_{42})\frac{p_4 - p_2}{\|p_4 - p_2\|_2} - k_{d_{42}}(v_4 - v_2) = \begin{bmatrix} 0 \\ 200 \\ 0 \end{bmatrix} \text{N}$$

$$F_3 = -k_{s_{43}}(\|p_4 - p_3\|_2 - r_{43})\frac{p_4 - p_3}{\|p_4 - p_3\|_2} - k_{d_{43}}(v_4 - v_3) = \begin{bmatrix} -29.39 \\ 29.29 \\ 0 \end{bmatrix} \text{N}$$

$$a_4(t_1) = \frac{1}{m_4}(m_4\vec{g} + F_1 + F_2 + F_3) = \begin{bmatrix} -84.77 \\ 385.32 \\ 0.05 \end{bmatrix} \frac{\text{m}}{\text{s}^2}$$

To find the state vector at t_2, say 0.01s after t_1, i.e. $t_2 = t_1 + 0.01$, a numerical integration method, such as first order Euler approximation can be taken:

$$x(t_2) = x(t_1) + 0.01 \cdot \dot{x}(t_1). \tag{3.8}$$

Therefore,

$$x(t_2) = \begin{bmatrix} \mathbf{p}(t_2) \\ \dot{\mathbf{p}}(t_2) \end{bmatrix} \quad \mathbf{p}(t_2) = \begin{bmatrix} p_1(t_2) \\ p_2(t_2) \\ p_3(t_2) \\ p_4(t_2) \end{bmatrix} = \begin{bmatrix} 0 \\ 2 \\ 0.005 \\ 1.01 \\ 2 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1.01 \\ -1 \\ 0 \end{bmatrix} \quad \dot{\mathbf{p}}(t_2) = \begin{bmatrix} v_1(t_2) \\ v_2(t_2) \\ v_3(t_2) \\ v_4(t_2) \end{bmatrix} = \begin{bmatrix} 0.5548 \\ -2.76 \\ 0.5 \\ 0.63 \\ -2.83 \\ 0 \\ 0.66 \\ 1.34 \\ 0 \\ 0.15 \\ 3.58 \\ 0 \end{bmatrix}.$$

The solution at at time t_2 can be approximated using other numerical integration methods as well. Higher order approximations can be solved for by using methods such as the Runge-Kutta integration method.

3.2 Linearization

Since the system from Equation (3.1) is geometrically nonlinear, the system needs to be linearized in order to be integrated into both the EKF formulation and implicit integration schemes. To do this, the Jacobian of the state vector needs to be evaluated at each time step. The Jacobian of a vector function $\vec{f}(x_1, \ldots, x_n)$ is defined as,

$$J = \frac{\partial f_i}{\partial x_j} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \cdots & \frac{\partial f_2}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial x_1} & \frac{\partial f_n}{\partial x_2} & \cdots & \frac{\partial f_n}{\partial x_n} \end{bmatrix}. \tag{3.9}$$

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

$$J(x) = \begin{bmatrix} \frac{\partial \mathbf{v}}{\partial \mathbf{p}} & \frac{\partial \mathbf{v}}{\partial \mathbf{v}} \\ \frac{\partial \mathbf{a}}{\partial \mathbf{p}} & \frac{\partial \mathbf{a}}{\partial \mathbf{v}} \end{bmatrix}, \tag{3.10}$$

where $\mathbf{p}$, $\mathbf{v}$ and $\mathbf{a}$ are $\mathbf{R}^{3n}$ vectors representing the positions, velocities and accelerations

of all nodes respectively.

The Jacobian is evaluated as follows:

The upper left quadrant of $J(x)$ contains the derivatives of the velocity states v_i with respect to the position states p_j, i.e.,

$$\frac{dv_i}{dp_j} = \mathbf{0}_{3\times 3}. \tag{3.11}$$

These derivatives are trivially 0. As a result, the upper left sub-matrix is a $3n \times 3n$ zeros matrix.
The upper right quadrant of $J(x)$ contains the derivatives of states v_i with respect to states v_j. The result is a $3n \times 3n$ identity matrix, $\mathbf{I}_{3\times 3}$. In other words,

$$\frac{dv_i}{dv_j} = \begin{cases} \mathbf{I}_{3\times 3}, & i = j \\ \mathbf{0}_{3\times 3}, & i \neq j \end{cases}. \tag{3.12}$$

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

$$\begin{aligned} \frac{\partial a_i}{\partial v_k} &= -\sum_{j\in A} \frac{\partial}{\partial v_k} k_d (v_i - v_j), \\ \frac{\partial a_i}{\partial v_k} &= -\sum_{j\in A} \frac{\partial}{\partial v_k} k_d v_i - \frac{\partial}{\partial v_k} k_d v_j. \end{aligned} \tag{3.13}$$

Since the derivative is not zero for only certain values of k, namely $k = j$ if $j \in \mathcal{A}$, and if $k = i$. The result for the first case is

$$\begin{aligned} \frac{\partial a_i}{\partial v_j} &= \frac{\partial}{\partial v_j} k_d v_j \\ \frac{\partial a_i}{\partial v_j} &= k_d \mathbf{I}_{3\times 3}. \end{aligned} \tag{3.14}$$

Now taking the derivative of a_i with respect to v_i, the result is,

$$\begin{aligned}\frac{\partial a_i}{\partial v_i} &= -\sum_{j\in A}\frac{\partial}{\partial v_i}k_d v_i \\ \frac{\partial a_i}{\partial v_i} &= -\sum_{j\in A}k_d\mathbf{I}_{3\times 3},\end{aligned} \tag{3.15}$$

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

Finally, for the bottom left quadrant of the Jacobian, the elements result from the taking derivative of accelerations, a_i, with respect to the positions, p_k. The derivation is as follows,

$$\frac{\partial a_i}{\partial p_k} = -\sum_{j\in A}\frac{\partial}{\partial p_k}[k_s(\|p_i-p_j\|_2 - r)\frac{p_i-p_j}{\|p_i-p_j\|_2}] \tag{3.16}$$

To simplify Equation (3.16), let $D(p_i) = (\|p_i-p_j\|_2 - r)$ and $E(p_i) = \frac{p_i-p_j}{\|p_i-p_j\|_2}$, so that $D(\cdot)$ and $E(\cdot)$ are only functions of p_i. Therefore, Equation (3.16) can be rewritten as

$$\frac{\partial a_i}{\partial p_k} = -\sum_{j\in A}\frac{\partial}{\partial p_k}[D(p_i)\cdot E(p_i)]. \tag{3.17}$$

Now, when $k=i$, to take the derivative of $D(p_i)\cdot E(p_i)$ with respect to p_i, the chain rule is applied

$$\frac{\partial a_i}{\partial p_i} = -k_s\sum_{j\in A}[D(p_i)\frac{dE(p_i)}{dp_i} + \frac{dD(p_i)}{dp_i}E(p_i)]. \tag{3.18}$$

Since $D(p_i)$ is a scalar function, its derivative results in a $\mathbf{R}^3$ vector. $E(p_i)$, on the other hand, is a $\mathbf{R}^3 \to \mathbf{R}^3$ vector function, meaning its derivative results in a $\mathbf{R}^{3\times 3}$ matrix. Therefore,

$$\begin{aligned}\frac{dD(p_i)}{dp_i} &= \frac{(p_i-p_j)^T}{\|p_i-p_j\|} \\ \frac{dE(p_i)}{dp_i} &= \frac{\mathbf{I}_{3\times 3}}{\|p_i-p_j\|} - \frac{(p_i-p_j)(p_i-p_j)^T}{\|p_i-p_j\|^3}.\end{aligned} \tag{3.19}$$

Combining Equations (3.18) and (3.19) results in

$$\frac{\partial a_i}{\partial p_i} = -k_s \sum_{j \in A} [(1 - \frac{r}{\|p_i - p_j\|})[\mathbf{I}_{3\times 3} - \frac{(p_i - p_j)(p_i - p_j)^T}{(p_i - p_j)^T(p_i - p_j)}] + \frac{(p_i - p_j)(p_i - p_j)^T}{(p_i - p_j)^T(p_i - p_j)}]. \tag{3.20}$$

When $k = j$, to find the derivative of a_i with respect p_j, notice how 3.16 can be rearranged to

$$\frac{\partial a_i}{\partial p_j} = \sum_{j \in A} \frac{\partial}{\partial p_j} [k_s(\|p_j - p_i\|_2 - r)\frac{p_j - p_i}{\|p_j - p_i\|_2}]. \tag{3.21}$$

Using the same technique from Equations (3.18)-(3.20), the derivative resolves to

$$\frac{\partial a_i}{\partial p_j} = k_s[(1 - \frac{r}{\|p_i - p_j\|})[\mathbf{I}_{3\times 3} - \frac{(p_i - p_j)(p_i - p_j)^T}{(p_i - p_j)^T(p_i - p_j)}] + \frac{(p_i - p_j)(p_i - p_j)^T}{(p_i - p_j)^T(p_i - p_j)}]. \tag{3.22}$$

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

$$J(x) = \begin{bmatrix} \mathbf{0} & \mathbf{I} \\ \mathbf{J}_{ap} & \mathbf{J}_{av} \end{bmatrix}, \tag{3.23}$$

where $\mathbf{J}_{ap} = \mathbf{M}^{-1}\frac{\partial \mathbf{a}}{\partial \mathbf{p}}$ is composed of the matrices from Equations (3.20) and (3.22), $\mathbf{J}_{av} = \mathbf{M}^{-1}\frac{\partial \mathbf{a}}{\partial \mathbf{v}}$ is composed of the matrices from Equations (3.14) and (3.15), and $\mathbf{M}$ is a diagonal matrix with the node mass values along its diagonal.

To gain further intuition about the Jacobian of a mass spring system, a numerical example follows. Using the same initial and final configuration from the numerical example in Section 3.1, the Jacobian about the initial configuration (Figure 3.4) will be found. The parameters of the system are $m_i = 1$kg for all four nodes, $k_{s_i} = 100$N/m and $k_{d_i} = 0.1$N·s/m for all six spring connections. As there are 4 nodes in this system, the Jacobian will be a $\mathbf{R}^{24\times 24}$ matrix. The upper left and right quadrants of the Jacobian are trivially found to $\mathbf{0}_{12\times 12}$ and $\mathbf{I}_{12\times 12}$, respectively. i.e.,

$$\begin{aligned} \frac{d\mathbf{p}}{d\mathbf{v}} &= \mathbf{0}_{12\times 12}, \\ \frac{d\mathbf{v}}{d\mathbf{v}} &= \mathbf{I}_{12\times 12}. \end{aligned} \tag{3.24}$$

The lower right $\mathbf{R}^{12\times 12}$ matrix, $\mathbf{J}_{av} = \mathbf{I}\frac{d\mathbf{a}}{d\mathbf{v}}$, can easily be found using Equations (3.14) and

(3.15),

$$\frac{d\mathbf{a}}{d\mathbf{v}} = \begin{bmatrix} -4k_d\mathbf{I} & k_d\mathbf{I} & k_d\mathbf{I} & k_d\mathbf{I} \\ k_d\mathbf{I} & -4k_d\mathbf{I} & k_d\mathbf{I} & k_d\mathbf{I} \\ k_d\mathbf{I} & k_d\mathbf{I} & -4k_d\mathbf{I} & k_d\mathbf{I} \\ k_d\mathbf{I} & k_d\mathbf{I} & k_d\mathbf{I} & -4k_d\mathbf{I} \end{bmatrix} = \begin{bmatrix} -0.4\mathbf{I}_{3\times3} & 0.1\mathbf{I}_{3\times3} & 0.1\mathbf{I}_{3\times3} & 0.1\mathbf{I}_{3\times3} \\ 0.1\mathbf{I}_{3\times3} & -0.4\mathbf{I}_{3\times3} & 0.1\mathbf{I}_{3\times3} & 0.1\mathbf{I}_{3\times3} \\ 0.1\mathbf{I}_{3\times3} & 0.1\mathbf{I}_{3\times3} & -0.4\mathbf{I}_{3\times3} & 0.1\mathbf{I}_{3\times3} \\ 0.1\mathbf{I}_{3\times3} & 0.1\mathbf{I}_{3\times3} & 0.1\mathbf{I}_{3\times3} & -0.4\mathbf{I}_{3\times3} \end{bmatrix}. \tag{3.25}$$

Lastly, the lower left $\mathbf{R}^{12\times12}$ matrix, $\mathbf{J}_{ap} = \mathbf{I}\frac{d\mathbf{a}}{d\mathbf{p}}$, can be found using Equations (3.20) and (3.22). The result is

$$\frac{d\mathbf{a}}{d\mathbf{p}} = \begin{bmatrix} -150 & 50 & 0 & 100 & 0 & 0 & 0 & 0 & 0 & 50 & -50 & 0 \\ 50 & -150 & 0 & 0 & 0 & 0 & 0 & 100 & 0 & -50 & 50 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 100 & 0 & 0 & -150 & -50 & 0 & 50 & 50 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -50 & -150 & 0 & 50 & 50 & 0 & 0 & 100 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 50 & 50 & 0 & -150 & -50 & 0 & 100 & 0 & 0 \\ 0 & 100 & 0 & 50 & 50 & 0 & -50 & -150 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 50 & -50 & 0 & 0 & 0 & 0 & 100 & 0 & 0 & -150 & 50 & 0 \\ -50 & 50 & 0 & 0 & 100 & 0 & 0 & 0 & 0 & 50 & -150 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}. \tag{3.26}$$

If the system is modelled with many nodes, the Jacobian can be constructed into a sparse matrix. Since each point can be connected to a maximum of 12 other points, due to the size of the set $\mathcal{A}$, each row in the sub-matrices $\mathbf{J}_{ap}$ and $\mathbf{J}_{av}$ can have at most 13 non-zero elements. The sparse structure of the Jacobian is an important attribute when simulating using implicit integration techniques. Implicit techniques often incorporate Newton solvers [19] that require the inverse of the Jacobian of a system. A sparse Jacobian allows for fewer computations, as sparse solvers are more computationally efficient than finding matrix inverses. The sparse matrix derivation for a rectangular mass spring model can be found in Appendix A.

3.3 Model Simulation

To obtain the state vector $x(t)$, the equation given in Equation (3.4) needs to be solved. An approximate solution to this equation can be solved using a numerical integration algorithm. A number of studies have looked at the convergence of numerical integration algorithms on mass spring cloth models [26]. Both implicit and explicit integration schemes have been studied for simulating cloth motion. Explicit integration is called *explicit* because it provides explicit formulas for the solution at the next time step. Implicit integration is called *implicit* because the solution is implicitly given as the solution of a system of equations[51]. Explicit integration algorithms such as the Euler forward method and Runge-Kutta method, among others, compute the next step of integration using only past information, making implementation straightforward. Explicit algorithms, however, can be numerically unstable if the integration step time is not chosen carefully. Additionally, the choice of step time can alter how quickly the effects of forces propagate over the cloth material. Implicit integration algorithms, on the other hand, use information at the terminus of its step time. Implicit integration, as a result, has a number of advantages over explicit integration methods. Firstly, because terminus information is being used, the system will be unconditionally numerically stable. This is ideal for systems which are numerically stiff. Secondly, because of the stability of the method, larger time steps can be used without the worry of unstable solutions. This allows the implicit methods to achieve the same level of accuracy as explicit methods, with fewer integration steps. Therefore, implicit approaches are more suitable for real-time applications. For deformable model simulations, implicit integration schemes allow for more robust parameter choices. From [54], 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. On the negative side, implicit techniques often times have no analytic solution. Instead, solution finding algorithms, such as a Newton solver, need to be run concurrently to find a solution. These solvers often times need the Jacobian of the system to be solved for at every time step, making them quite computationally intensive. For systems which are difficult to model, the Jacobian may need to be approximated, resulting in additional simulation error.

The fixed step-time Runge-Kutta method and the Euler backwards method are compared in this chapter to show the similarities between explicit and implicit simulation methods. Additionally, they are also compared to show that the implicit scheme, due its computational efficiency, is the better choice as the integration method for model compression in Chapter 5.

Runge-Kutta Method

Given the system,

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

and time step ΔT, the fixed step-time Runge-Kutta solves the next integration step $x[k+1]$ using the current solution $x[k]$ in the following way [60]:

$$x[k+1] = x[k] + \frac{\Delta T}{6}(k_1 + 2k_2 + 2k_3 + k_4), \tag{3.27}$$

where,

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

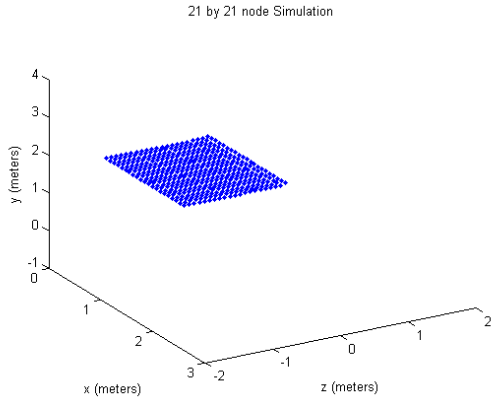
This process is continued until the termination of the simulation. The fixed step-time Runge-Kutta algorithm, as a result, provides an approximation of the state vector $x[k]$ at each step k. Here,

$$x[k] = x(t_0 + k\Delta T) \qquad k = 0, 1, \ldots \tag{3.28}$$

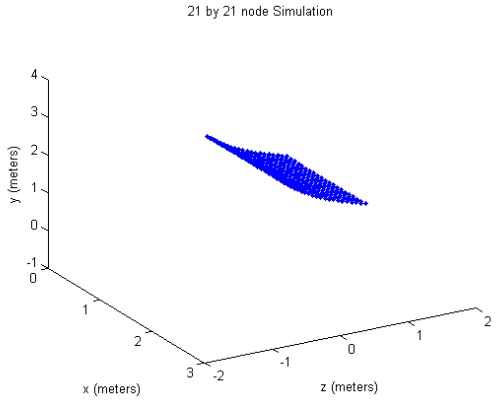
Figure 3.5 shows a series of snapshots of simulation data, solved with the Runge-Kutta algorithm, for a falling 21×21 node cloth model anchored horizontally along the top row of nodes. The solution was solved for at an integration step time of $\Delta T = 0.001$s. For this simulation, the model parameters (mass, spring constants, dampers) were arbitrarily chosen to be the same for each node and spring/damper connection. The masses were chosen to be 0.025kg for each node making the total mass of the cloth 11.025kg, the spring constants were chosen to be 300N/m, and the damper constants were chosen to be 0.08N·s/m. Both the total length and width of the cloth are 1.5m, evenly separated between nodes.

Euler Backward Method

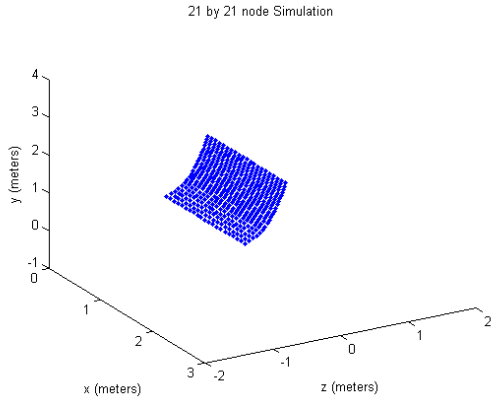
A simple implicit integration scheme is the Euler backward method. As described by [26], it allows for stability at larger time steps. As a result, it is significantly faster than explicit simulation methods due to the reduced number of computations. The method begins by



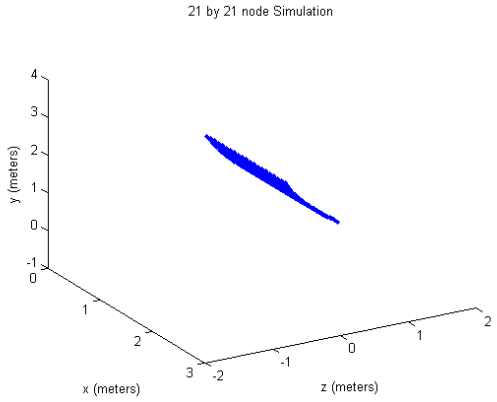
(a) Initial cloth position, $t = 0$s



(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 3.5: Runge Kutta simulation of falling 21×21 node anchored cloth

first defining the change in state

$$\Delta x = \begin{bmatrix} \Delta p \\ \Delta v \end{bmatrix} = \begin{bmatrix} p(t_0 + \Delta T) - p(t_0) \\ v(t_0 + \Delta T) - v(t_0) \end{bmatrix}, \tag{3.29}$$

and solving the first order approximation,

$$\begin{bmatrix} \Delta p \\ \Delta v \end{bmatrix} = \Delta T \begin{bmatrix} \Delta v + v_0 \\ \Delta a + a_0 \end{bmatrix}. \tag{3.30}$$

Here Δa is the change in acceleration and is a function of p and v as in Equation (3.5). Since $a(x)$ is nonlinear, the first order Taylor approximation is used to aid in solving for Δv,

$$\Delta v \approx a(p_0, v_0) + \frac{\partial a}{\partial p}\Delta p + \frac{\partial a}{\partial v}\Delta v. \tag{3.31}$$

Rearranging for Δv in Equation (3.31), and replacing Δp with $\Delta T(\Delta v + v_0)$, Δv is found to be the solution to the linear system,

$$(\mathbf{I} - \Delta T^2 \frac{\partial a}{\partial p} - \Delta T \frac{\partial a}{\partial v})\Delta v = \Delta T a(p_0, v_0) + \Delta T^2 \frac{\partial a}{\partial p} v_0 \tag{3.32}$$

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

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

Using the solution Δv, $p(t_0 + \Delta T)$ is solved for by taking,

$$p(t_0 + \Delta T) = \Delta T(\Delta v + v_0)$$

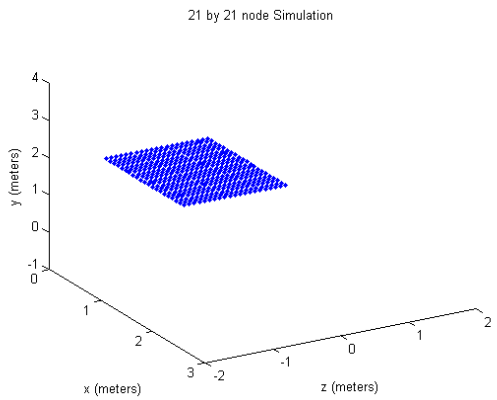
and $v(t_0 + \Delta T)$ is solved for by taking,

$$v(t_0 + \Delta T) = \Delta v + v_0.$$

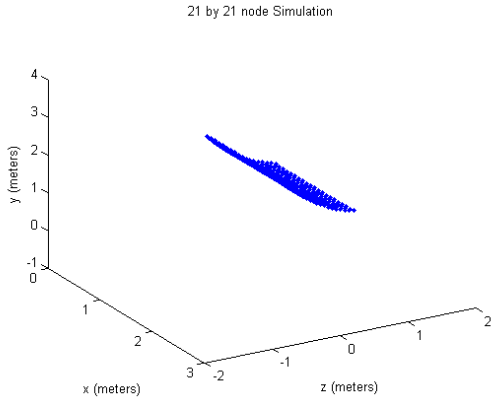
These steps are completed at each integration time step until the termination of the simulation.

Figure 3.6 shows snapshots of the forward Euler method applied to a 21×21 node cloth model falling while anchored along a row of nodes. The solution was solved for at an integration step time of $\Delta T = 0.01$s. For this simulation, the model parameters are chosen

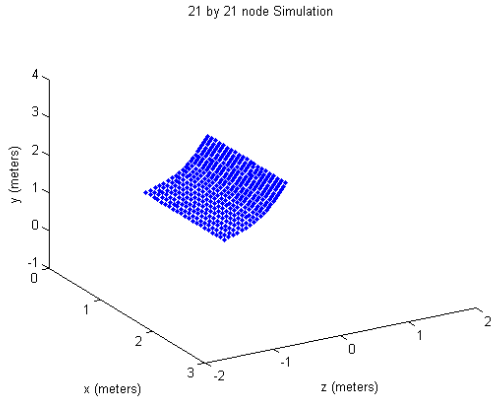
to be the same as the Runge-Kutta simulation.



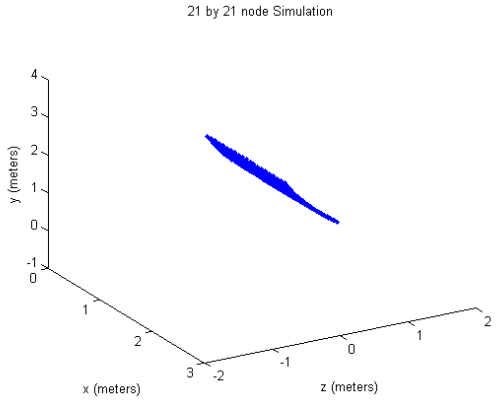
(a) Initial cloth position, $t = 0$s



(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 3.6: Implicit Euler Backwards simulation of falling 21×21 node anchored cloth

A comparison of the Runge-Kutta simulation and Euler backwards simulation is shown in Figure 3.7. The plot shows the average between the nodes of each cloth for each time step, i.e.,

$$e[k] = \frac{1}{n} \sum_{i=1}^{n} \|p_i^{rk}[k] - p_i^{eb}[k]\|_2, \quad k = 0, 1, \ldots. \tag{3.33}$$

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.

Both methods produce very similar results, as the difference between associated nodes is at most 3.5cm for a 10 second simulation. However, the difference does increase as the simulations progress. This is expected as approximation error increases in each method as the simulations progress. Comparing the error between the two methods when the integration step times are the same, e.g. $\Delta T = 0.001$, the result is 0 for the entire simulation (Figure 3.8). This shows that both methods produce the same results with the same step time, as long as the methods are still numerically stable. As the Runge-Kutta does not produce stable results at $\Delta T = 0.01$, the results at this step time cannot be compared. Since real world systems do not behave exactly like the mass spring model, small differences between simulation results are not significant enough to discard either method.

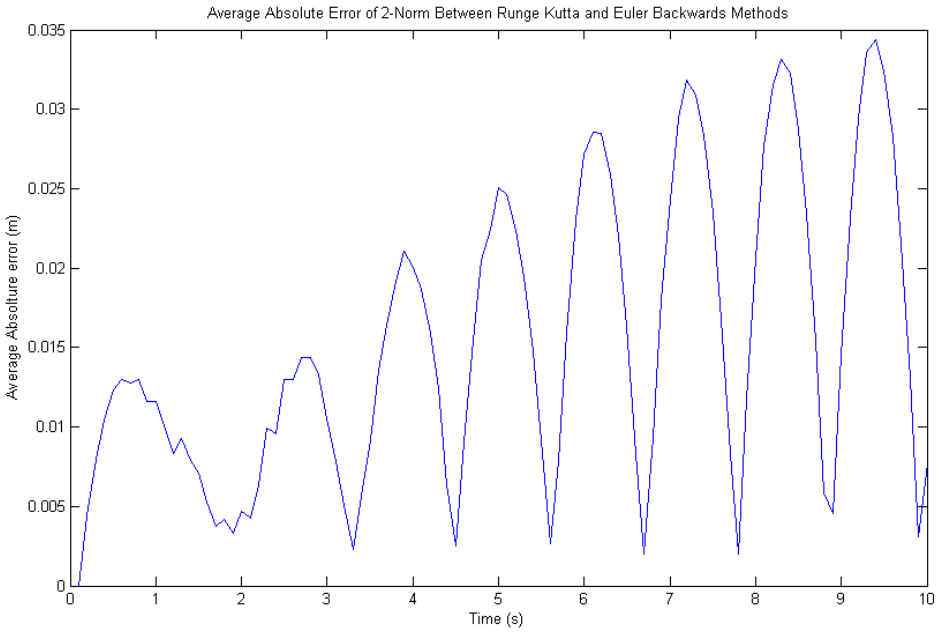


Figure 3.7: Comparison of Runge-Kutta at $\Delta T = 0.001$s and Euler Backwards Methods at $\Delta T = 0.01$s

Using the mass spring model derived in this chapter, a filter can be built that makes estimates of an object's future positions. The next chapter will review filtering approaches that produce optimal estimation results assuming certain conditions. The efficiency of an estimation filter, however, is inversely proportional to the number of states in the system. In this case, the number of nodes needs to be reduced to allow for predictions to be run

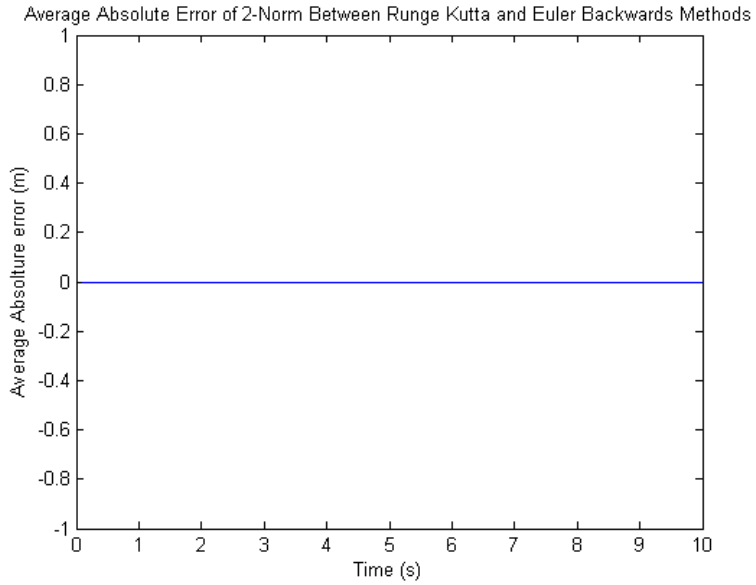


Figure 3.8: Comparison of Runge-Kutta at $\Delta T = 0.001$s and Euler Backwards Methods at $\Delta T = 0.001$s

in real-time. To reduce the number of nodes in the model, optimal spring, damper and mass parameters can be found that match the behaviour of the sparse desired model to a dense actual model. Chapter 5 will present this technique and provide results of model compression.

Chapter 4

Review of Filtering Approaches

When measuring the movement of a time-varying surface, there will inherently be noise in the position data due to sensor noise. Additionally, since the mass spring model is used as the dynamic model of real world surface, there will be modelling error that will affect the system. As a result, filtering techniques are used to obtain an accurate estimate of system states when the measurement data contains statistical noise or other inaccuracies. Generally, these techniques attempt to find an estimate that minimizes the error between the actual state value and the estimated states. This chapter will focus on filtering techniques that minimize the squared error between actual state values and state estimates, i.e.,

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

where $\hat{x}$ is the state estimate and x is the actual state value. This chapter will review common filtering approaches for state estimation: the least squares estimator, the Kalman filter, the extended Kalman filter, and the unscented Kalman filter.

4.1 Least Squares Estimation

The least squares method is a method of finding a state estimate that minimizes the squared discrepancy between the actual value of the state and the state estimate. The method was independently discovered by both Gauss and Legendre in the early 19th century for applications in astronomy. The least squares method has been applied to numerous application areas for both state estimation and parameter identification [40]. The classic least squares method is explained as follows [35].

Given a sequence of n noisy measurements $\{y_i\}_{i=1}^n \in L_2$ and random variable $Z \in L_2$, where L_2 is the space of square Lebesgue-integrable functions, i.e., a random variable Z is square integrable if

$$\mathbf{E}[Z^2] < \infty, \tag{4.2}$$

the least squares estimate $\hat{Z}$ of Z is the solution

$$\hat{Z} = f(y_1, y_2, \ldots, y_n), \tag{4.3}$$

to the minimization problem

$$f(y_1, y_2, \ldots, y_n) = \arg\min_g \{\mathbf{E}[(Z - g(y_1, y_2, \ldots, y_n))^2] |\ g : \mathbf{R}^n \to \mathbf{R}\}. \tag{4.4}$$

Here, $\mathbf{E}[\cdot]$ is the expected value function, and $g(\cdot)$ is a nonlinear function applied to the set of noisy measurement data. In other words, the least squares method finds a random variable $\hat{Z}$ that minimizes the expected value of the squared difference between Z and $\hat{Z}$. It is well known that the solution to this problem is given as

$$f(y_1, y_2, \ldots, y_n) = \mathbf{E}[Z|y_1, y_2, \ldots, y_n], \tag{4.5}$$

where $\mathbf{E}[Z|y_1, y_2, \ldots, y_n]$ is the conditional expectation of Z given $y_1, y_2, \ldots, y_n$ [35]. When $Z, y_1, y_2, \ldots, y_n$ are all random variables chosen from a Gaussian distribution, the conditional expectation of Z given $y_1, y_2, \ldots, y_n$, and therefore the solution to the least squares problem, becomes an affine function of the measured data. More specifically,

$$\hat{Z} = \mathbf{E}[Z|y_1, y_2, \ldots, y_n] = \sum_{i=1}^{n} \alpha_i y_i + \beta_i, \tag{4.6}$$

where $\alpha_i, \beta_i \in \mathbf{R}$.

Furthermore, when the estimator function $f(\cdot)$ is in the form of a linear function (Equation (4.6) with $\beta_i = 0$ for all $i = 1, \ldots, n$), $\hat{Z}$ it is called the linear least squares estimate of random variable Z. The linear least squares estimate has a closed form solution of the form,

$$\hat{Z} = \mathbf{E}[ZY]^T \mathbf{E}[ZZ^T]^{-1} Y, \tag{4.7}$$

where $Y = [y_1 y_2 \ldots y_n]^T$ and $\mathbf{E}[Z] = \mathbf{E}[y_i] = 0\ \forall i = 1 \ldots n$.

A straightforward example of the linear least squares estimator is the following: Given

a linear system defined by the equation

$$Az = y, \tag{4.8}$$

where $A \in \mathbf{R}^{n \times n}$ defines the linear system, $z \in \mathbf{R}^n$ is the system state vector, and $y \in \mathbf{R}^n$ is the output vector, the linear least squares estimate, $\hat{z}$, is the solution to the optimization problem

$$\arg\min_{\hat{z}} \|A\hat{z} - y\|_2^2. \tag{4.9}$$

The solution to this problem is closed formed, and can be solved by setting the derivative of Equation (4.9) to zero, i.e.,

$$\begin{aligned} &\frac{\partial \|A\hat{z} - y\|_2^2}{\partial z} = \frac{\partial (A\hat{z} - y)^T (A\hat{z} - y)}{\partial z} = 0, \\ &2A^T A\hat{z} - 2A^T y = 0, \\ &\hat{z} = (A^T A)^{-1} A^T y. \end{aligned} \tag{4.10}$$

If not all measurement data is available immediately, the linear least squares estimator can be solved in a recursive fashion. Equation (4.11) gives the recursive algorithm for finding the linear least square estimate for the current time step given the previous state estimate and the current measurement.

$$\hat{Z}_k = \hat{Z}_{k-1} + \frac{\mathbf{E}[Z_{k-1}\tilde{y}_k]}{\mathbf{E}[\tilde{y}_k^2]}\tilde{y}_k \tag{4.11}$$

Here, $\tilde{y}_k = y_k - P(y_k | L_{k-1})$, where P is the projection operator that finds the minimum projection of y_k onto the subspace L_{k-1}, the space spanned by $y_1, \ldots, y_{k-1}$.

As mentioned in Equation (4.6), when all inputs are Guassian random variables with non-zero means, the solution to the least squares problem takes the form of an affine function. The least squares estimate $\hat{Z}$ has the following solution,

$$\hat{Z} = P((Z - \mathbf{E}[Z]) | L_k^c) + \mathbf{E}[Z] \tag{4.12}$$

where L_k^c is the subspace spanned by $y_1 - \mathbf{E}[y_1], \ldots, y_k - \mathbf{E}[y_k]$.

The following three algorithms discussed in this chapter find the linear least squares

state estimate given a dynamic system of the form

$$\begin{aligned} \dot{x} &= f(x) \\ y &= h(x), \end{aligned} \tag{4.13}$$

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, named after one if its primary developers, Rudolph E. Kalman, is an algorithm that finds a affine least squares state estimate of a linear dynamic system given noisy measurement data. The Kalman filter, derived in [41], is a recursive, efficient algorithm requiring only the previous state estimate, rather than the entire history of states, to calculate the current state estimate. Under certain circumstances, the Kalman filter provides the optimal state estimate for a linear system. The Kalman filter has numerous applications in technology including guidance and navigation of vehicles, signal processing, and econometrics. Due to its popularity and effectiveness, there are many available resources solely focused on the applications of the Kalman filter. The remainder of this section describes the relevant equations and conditions required to apply the Kalman filter.

Suppose a linear, dynamic system is given in the form

$$\begin{aligned} x[k+1] &= F_k x[k] + w_k \\ y[k] &= H_k x[k] + v_k, \end{aligned} \tag{4.14}$$

where $F_k \in \mathbf{R}^{n \times n}$ is the state transition matrix, $H_k \in \mathbf{R}^{n \times n}$ is the observation matrix, $x[k] \in \mathbf{R}^n$ is the state vector, $y[k] \in \mathbf{R}^m$ is the output vector, $w_k \in \mathbf{R}^n$ is the process noise vector, and $v_k \in \mathbf{R}^m$ is the measurement noise vector, all at time step k. Since the system is in a recursive form, the recursive affine least-squares estimator can be used to find the state estimate vector $\hat{x}_{k|k}$ at time step k, as described in Equation (4.6). The Kalman filter finds the optimal least squares state estimate when the noise vectors are chosen from a Gaussian distribution. In practice, however, the Kalman filter does not require noise vectors w_k and v_k to be Gaussian random variables to produce convergent results.

The Kalman filter algorithm can be written in a number different forms; for this thesis, the two step approach is used. The two steps are composed of the prediction step and the update step. The prediction step procedure is listed in Equation (4.15).

$$\begin{aligned} \hat{x}_{k|k-1} &= F_k \hat{x}_{k-1|k-1} \\ P_{k|k-1} &= F_k P_{k-1|k-1} F_k^T + Q_k. \end{aligned} \tag{4.15}$$

Here, $\hat{x}_{k|k-1}$ is the state prediction given the previous $k-1$ measurements, $P_{k|k-1}$ is the state covariance matrix given the previous $k-1$ measurements, and Q_k is the process noise covariance matrix. The prediction step uses the previous state estimate 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. For deformable surface prediction, the *a priori* state estimate is used as the state prediction, since only previous data is available at the any given time step.

The Kalman update step equations are shown in Equation (4.16).

$$\begin{aligned} \tilde{y}_k &= y[k] - H_k \hat{x}_{k|k-1} \\ S_k &= H_k P_{k|k-1} H_k^T + R_k \\ K_k &= P_{k|k-1} H_k^T S_k^{-1} \\ \hat{x}_{k|k} &= \hat{x}_{k|k-1} + K_k \tilde{y}_k \\ \Sigma_{k|k} &= (I - K_k H_k) P_{k|k-1} \end{aligned} \tag{4.16}$$

Here $\tilde{y}_k$ is the measurement residual, which is the error between the measured output and the predicted output, S_k is the residual 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.

In most applications, the prediction and update steps alternate. The state prediction advances until the next observation is made, which allows the update step to be applied. The algorithm continues repeatedly until termination. To begin the Kalman filter algorithm, an initial state vector $\hat{x}_{0|0}$ and state covariance matrix $P_{0|0}$, along with noise covariance matrices Q_k and R_k need to be supplied. The selection of $\hat{x}_{0|0}$, $P_{0|0}$, Q_k and R_k greatly affect the performance of the Kalman filter, and in general, they are difficult to find. These parameters can be estimated using a priori statistical knowledge about the noise and errors in the system; however, often times, they are simply tuned until the best results are observed [57].

As the standard Kalman filter can only be applied to linear systems, the algorithm needs to be reworked to handle nonlinear dynamical systems. The extended Kalman filter

and the unscented Kalman filter are variations of the standard Kalman filter that handle nonlinearities with varying amounts of success. These methods are reviewed and discussed in the remainder of this chapter.

4.3 Extended Kalman Filter

Although the Kalman filter is used in a vast number of application areas, it is only applicable to linear systems and is only optimal when the systems are subject to Gaussian process and measurement noises. The extended Kalman filter (EKF) is an extension of the standard Kalman filter, designed to handle systems with nonlinear dynamics. It enables the estimation of state variables through the linear approximation of the system's dynamics. Essentially, the EKF follows the same methods as the standard Kalman Filter after linearizing the system at every time step. The rest of this section describes the EKF and its implementation on nonlinear systems [9].

Given a nonlinear dynamic system of the form

$$\begin{aligned} x[k+1] &= f(x[k]) + w_k \\ y[k] &= h(x[k]) + v_k, \end{aligned} \tag{4.17}$$

where $f : \mathbf{R}^n \to \mathbf{R}^n$ and $h : \mathbf{R}^n \to \mathbf{R}^m$ are sufficiently smooth functions, $x[k]$ is the state vector at time step k, w_k is the process noise at time step k, $y[k]$ is the output at time step k, and v_k is the measurement noise at time step k. Additionally, for optimality, 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:

$$\begin{aligned} \mathbf{E}[w_k w_k^T] &= Q_k \\ \mathbf{E}[v_k v_k^T] &= R_k \\ \mathbf{E}[w_k v_k^T] &= 0. \end{aligned} \tag{4.18}$$

Since the standard Kalman filter uses only linear operators in its prediction and update steps, the functions f and h need to be linearized. Since the functions f and h are assumed to be smooth, and as a result, differentiable, the Jacobian of f and h can be evaluated at each time step k. The Jacobians of f and h are used in place of the state transition matrix $F[k]$ and observation matrix $H[k]$ in the standard Kalman filter, respectively. Equation

(4.19) lists the Jacobians of f and h.

$$\begin{aligned} F_{k-1} &= \frac{\partial f}{\partial x}|_{\hat{x}_{k-1|k-1}} \\ H_k &= \frac{\partial h}{\partial x}|_{\hat{x}_{k|k-1}} \end{aligned} \tag{4.19}$$

The EKF recursively predicts the states at time step k by using the measurement data available at the previous time step, $k-1$. This prediction is denoted as $\hat{x}_{k|k-1}$ and the associated state covariance at time k given the $k-1$ measurement is denoted as $P_{k|k-1}$. The new state prediction $\hat{x}_{k|k-1}$ is calculated by applying the state transition function f to the most recent state estimate $\hat{x}_{k-1|k-1}$. The Kalman prediction step is shown in Equation (4.20).

$$\begin{aligned} \hat{x}_{k|k-1} &= f(\hat{x}_{k-1|k-1}) \\ P_{k|k-1} &= F_{k-1}P_{k-1|k-1}F_{k-1}^T + Q_{k-1} \end{aligned} \tag{4.20}$$

To update the state prediction to the current state estimate, $\hat{x}_{k|k}$ the EKF completes the steps listed in Equation (4.21).

$$\begin{aligned} \tilde{y}_k &= y_k - h(\hat{x}_{k|k-1}) \\ S_k &= H_k P_{k|k-1} H_k^T + R_k \\ K_k &= P_{k|k-1} H_k^T S_k^{-1} \\ \hat{x}_{k|k} &= \hat{x}_{k|k-1} + K_k \tilde{y}_k \\ P_{k|k} &= (\mathbf{I} - K_k H_k) P_{k|k-1} \end{aligned} \tag{4.21}$$

After the current measurement is made, the residual $\tilde{y}$ is computed and is multiplied by the Kalman gain K_k to correct for error in the state prediction. The Kalman gain term is also used to update the state covariance $P_{k|k}$.

To begin the EKF recursion, an initial state estimate $\hat{x}_{0|0}$ and an initial covariance matrix $P_{0|0}$ must be supplied.

Since the EKF computes a first order approximation of the nonlinear system, performance limitations may arise due to the linearization. Unlike the standard Kalman Filter, which is guaranteed to converge given a linear model with Gaussian, independent noise, the EKF may not converge. Generally, the EKF performs best when the error caused by linearization is smaller than other uncertainties in the system. Higher order approximations of the nonlinear system may reduce model error. However, they only provide significant improvements when measurement noise is small. Additionally, the choice of initial vari-

ables $\hat{x}_{0|0}$ and $P_{0|0}$, as well as well as noise covariance matrices Q_k and R_k, could greatly effect the performance of the EKF. Choosing these parameters, or parameter tuning, is a highly researched component of Kalman filters and the choice of parameters is application dependent.

4.4 Unscented Kalman Filter

When there are nonlinearities in the dynamic state model, the EKF provides an approximate solution for the optimal term $\hat{x}_{k|k-1}$ as simply a function of the mean value of the prior estimates. Additionally, since the model is linearized by taking the Jacobian of the system, the covariance matrices are propagated through a first order approximation of the system. These approximations may introduce errors into the *a posteriori* state estimate, and as a result, cause sub-optimal or even divergent behaviour. The unscented Kalman filter (UKF)[63] addresses these issues by providing a third order approximation, when the noise variables are Gaussian distributed, and a second order approximation, if not. This allows for a more accurate estimation of the *a posteriori* state estimate's mean and covariance. The UKF utilizes the unscented transformation (UT), which is a method for calculating the statistics for a random variable undergoing a nonlinear transformation. Since it is difficult to find the statistical properties of a nonlinearly transformed random variable, the UT uses a set of carefully chosen sample points, or sigma points, to estimate the mean and covariance of the random variable.

The UKF algorithm begins by defining augmented state estimate, $\mathbf{x}^a_{k-1|k-1}$, and state covariance matrix, $\mathbf{P}^a_{k-1|k-1}$, as

$$\begin{aligned} \mathbf{x}^a_{k-1|k-1} &= \begin{bmatrix} \hat{x}^T_{k-1|k-1} & 0 \end{bmatrix}^T \\ \mathbf{P}^a_{k-1|k-1} &= \begin{bmatrix} P_{k-1|k-1} & 0 \\ 0 & Q_k \end{bmatrix}. \end{aligned} \tag{4.22}$$

A set of $2L+1$ sigma points $\chi^i_{k-1|k-1}$ are chosen based on the augmented state estimate and augmented covariance matrix:

$$\begin{aligned} \chi^0_{k-1|k-1} &= \mathbf{x}^a_{k-1|k-1} \\ \chi^i_{k-1|k-1} &= \mathbf{x}^a_{k-1|k-1} + (\sqrt{(L+\lambda)\mathbf{P}^a_{k-1|k-1}})_i, \quad i = 1, \ldots, L \\ \chi^i_{k-1|k-1} &= \mathbf{x}^a_{k-1|k-1} + (\sqrt{(L+\lambda)\mathbf{P}^a_{k-1|k-1}})_{i-L}, \quad i = L+1, \ldots, 2L, \end{aligned} \tag{4.23}$$

where $(\sqrt{(L+\lambda)\mathbf{P}^a_{k-1|k-1}})_i$ is the ith column of the matrix $\sqrt{(L+\lambda)\mathbf{P}^a_{k-1|k-1}}$. The matrix $\mathbf{B}$ is the square root of a matrix $\mathbf{A}$, $\sqrt{\mathbf{A}}$, if $\mathbf{B}\mathbf{B}^T = \mathbf{A}$. Each χ^i has weights associated with their state and covariance values, W^i_s and W^i_c, respectively, and are defined as:

$$\begin{aligned} W^0_s &= \frac{\lambda}{L+\lambda} \\ W^0_c &= \frac{\lambda}{L+\lambda} + 1 - \alpha^2 + \beta \\ W^i_s &= W^i_c = \frac{\lambda}{2(L+\lambda)} \end{aligned} \tag{4.24}$$

where $\lambda = \alpha^2(L+\kappa) - L$. α determines the spread of the sigma points around $\mathbf{x}^a_{k-1|k-1}$, κ is a secondary scaling parameter and β is used to incorporate prior knowledge of the distribution of $\mathbf{x}^a_{k-1|k-1}$.

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

$$\chi^i_{k|k-1} = f(\chi^i_{k-1|k-1}). \tag{4.25}$$

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

$$\begin{aligned} \hat{x}_{k|k-1} &= \sum_{i=0}^{2L} W^i_s \chi^i_{k|k-1} \\ P_{k|k-1} &= \sum_{i=0}^{2L} W^i_c (\chi^i_{k|k-1} - \hat{x}_{k|k-1})(\chi^i_{k|k-1} - \hat{x}_{k|k-1})^T. \end{aligned} \tag{4.26}$$

The update step is takes the predicted state and covariance values, and once again creates augmented variables

$$\begin{aligned} \mathbf{x}^a_{k|k-1} &= \begin{bmatrix} \hat{x}^T_{k|k-1} & 0 \end{bmatrix}^T \\ \mathbf{P}^a_{k|k-1} &= \begin{bmatrix} P_{k|k-1} & 0 \\ 0 & R_k \end{bmatrix}. \end{aligned} \tag{4.27}$$

New sigma points $\chi^i_{k|k-1}$ are found by replacing $\mathbf{x}^a_{k-1|k-1}$ and $\mathbf{P}^a_{k-1|k-1}$ in Equation (4.23) with $\mathbf{x}^a_{k|k-1}$ and $\mathbf{P}^a_{k|k-1}$, respectively. The sigma points are then passed through the observation model $h(\cdot)$, such that

$$\gamma^i_k = h(\chi^i_{k|k-1}). \tag{4.28}$$

These γ_k^i values are combined with their associated weights to produce the predicted measurement and covariance matrices:

$$\begin{aligned}
\hat{y} &= \sum_{i=0}^{2L} W_s^i \gamma_k^i \\
P_{y_k y_k} &= \sum_{i=0}^{2L} W_c^i (\gamma_k^i - \hat{y}_k)(\gamma_k^i - \hat{y}_k)^T \\
P_{x_k y_k} &= \sum_{i=0}^{2L} W_c^i (\gamma_k^i - \hat{x}_{k|k-1})(\gamma_k^i - \hat{y}_k)^T.
\end{aligned} \tag{4.29}$$

The Kalman gain K_k is now defined as

$$K_k = P_{x_k y_k} P_{y_k y_k}^{-1}, \tag{4.30}$$

and the estimated state vector and covariance matrix are found to be

$$\begin{aligned}
\hat{x}_{k|k} &= \hat{x}_{k|k-1} + K_k(y_k - \hat{y}_k) \\
P_{k|k} &= P_{k|k-1} - K_k P_{y_k y_k} K_k^T.
\end{aligned} \tag{4.31}$$

For state transition functions which are difficult to differentiate, or non-differentiable, the EKF can not be used as the algorithm requires the Jacobian of the system. As can be seen in the steps of the UKF, the Jacobian of the system does not need to be explicitly calculated for the UKF to be implemented. Furthermore, the overall number of computations are the same order as the EKF. As a result, the UKF has largely replaced the EKF in a number of nonlinear control applications including ground and air navigation [29].

For the mass spring model derived in Chapter 3, the only filtering techniques reviewed in this chapter that can be used are the EKF and UKF, due to the model's nonlinearity. Since the model is differentiable, with the Jacobian having an analytic solution, the EKF can be easily applied. Due to the ease of implementation and effectiveness of the EKF, it will be the primary focus for prediction filtering in this thesis. With more time, the UKF can be implemented with the mass spring model, possibly increasing the accuracy of state predictions and estimations. Other techniques, such as particle filtering methods, can be used to estimate state values with varying degrees of effectiveness.

Chapter 5

Model Compression

Although modelling mass spring systems with a large number of point masses increases the realism of the model, the cost of running the EKF increases significantly due to the large number of state variables. The update step of the EKF algorithm, specifically the Kalman gain calculation,

$$K_k = P_{k|k-1}H^T(HP_{k|k-1}H^T + R_k)^{-1}, \tag{5.1}$$

requires the inverse of a $6n \times 6n$ matrix. Matrix inversion is often a $\mathcal{O}(n^3)$ computation time operation, implying that the number of operations increase cubically with the number of mass nodes. Furthermore, even solving the linear system implicitly, i.e., solve for K_k in

$$K_k(HP_{k|k-1}H^T + R_k) = P_{k|k-1}H^T, \tag{5.2}$$

is a computationally expensive task, as the matrix $HP_{k|k-1}H^T + R_k$ does not have any beneficial structure. Therefore, finding a reduced ordered system, with similar physical behaviour as the original system, will result in a less computationally expensive EKF model, while maintaining realism. This would allow for close to real-time prediction of cloth movement. This chapter will explore techniques of compressing dense state models to models with fewer states, while keeping physical meaning, for more efficient prediction.

5.1 Data Driven Compression

To reduce the order of a mass spring system, a data driven approach can be used. In other words, a sparse model can be found by minimizing the error between the outputs of

a "measured" dense system and the "desired" sparse system,

$$\min \int_0^t \|Hz(\tau) - \mathbf{A}Hx(\tau)\| d\tau. \tag{5.3}$$

Here, $x(t)$ is a $\mathbf{R}^m$ time series vector of the higher order system states, $z(t)$ is a $\mathbf{R}^n$ time series vector of the lower order system states where $m > n$, H is the observation model, and $\mathbf{A}$ is a $\mathbf{R}^{n \times m}$ matrix that selects the corresponding states of the higher order system to compare to the lower order system. Equation (5.3) can be written as a minimization problem of the squared error between the two systems, where the minimization variable is the parameter vector θ. In this case, the parameter vector θ would contain all spring constants, damper constants and masses of the reduced order system. The general minimization problem is defined as:

$$\begin{aligned} \min_{\theta} \quad & \int_0^t \|Hz(\tau) - \mathbf{A}Hx(\tau)\|_2^2 d\tau \\ \text{s.t.} \quad & \dot{z}(t) = f(z, \theta), \end{aligned} \tag{5.4}$$

where θ is a $\mathbf{R}^{2s+n}$ parameter vector and s is the total number of spring connections in the lower order system. For a rectangular mass spring model with n_1 rows and n_2 columns, i.e. $n_1 \times n_2$ nodes, the total number of spring connections s is found to be

$$s = n_1(n_2-1)+n_2(n_1-1)+2(n_1-1)(n_2-1)+n_1 \cdot \max\{0, n_2-2\}+n_2 \cdot \max\{0, n_1-2\}. \tag{5.5}$$

The parameter vector θ contains three types of parameters: spring constants k_s, damper constants k_d, and masses m. Written in a vector:

$$\theta = \begin{bmatrix} k_{s_1} \\ \vdots \\ k_{s_s} \\ k_{d_1} \\ \vdots \\ k_{d_s} \\ m_1 \\ \vdots \\ m_n \end{bmatrix}. \tag{5.6}$$

Due to the outputs being in discrete time, the minimization term in Equation (5.4) can be approximated by

$$\sum_{i=0}^{N} \|Hz[i] - \mathbf{A}Hx[i]\|_2^2, \tag{5.7}$$

where N is the number of samples, and $x[i] = x(t_0 + i\Delta T)$ where ΔT is the step time of the integration that solved $x[i]$ and $z[i]$. By the nature of this minimization, there may not be a unique parameter vector θ that minimizes Equation (5.4). In other words, there may not be a unique compressed realization of the model. This can be seen by looking at a simple one dimensional example. Given two mass-spring-damper systems, the parameters of a single mass-spring-damper system can be found, such that the behaviour of the system matches the behaviour of a given two mass-spring-damper system. Figure 5.2 shows a two mass-spring-damper system (a) and its analogous electrical circuit topology (b). Figure 5.1 shows a single mass-spring-damper system (a) and its equivalent electrical circuit topology (b). The equivalent parameters between the mechanical and electrical models are shown in Table 5.1 [1].

Table 5.1: Electrical and mechanical parameters

Electrical	Mechanical
Voltage e	Velocity v
Current i	Force F
Capacitance C	Mass $1/M$
Inductance L	Stiffness $1/K$
Resistance R	Damping $1/D$

In both circuits, the Norton equivalents are taken so that there is a single current source connected to a load impedance. This allows for the impedances and currents of each circuit to be compared to one another. By inspection, it is obvious that the impedance of Figure 5.2b is a function of all mass, spring, and damper parameters (all passive circuit elements). As a result, the Norton equivalent impedance has the form

$$Z_2 = g_2(L_1, R_1, C1, L_2, R_2, C_2) + jp_2(L_1, R_1, C_1, L_2, R_2, C_2), \tag{5.8}$$

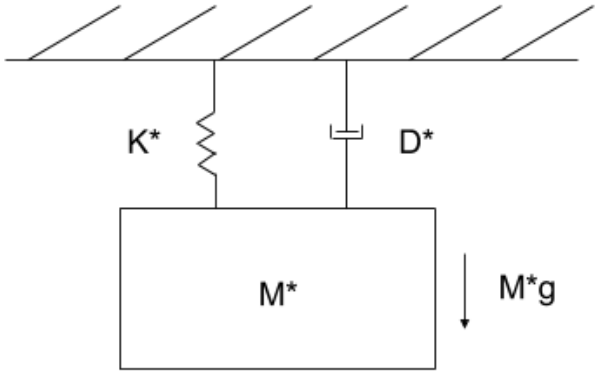
where j is the imaginary number, $g_2(\cdot)$ is a function representing the real part of Z_2, and $p_2(\cdot)$ is a function representing the imaginary part of Z_2 . The Norton equivalent impedance

of Figure 5.1b has the form

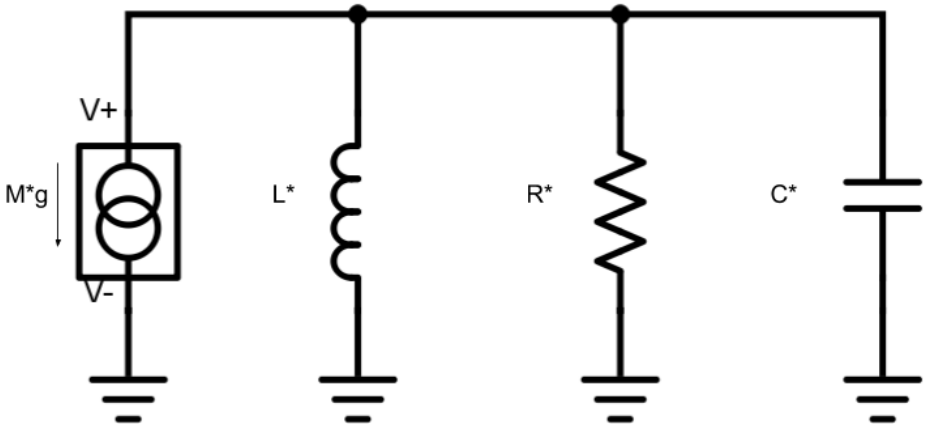
$$Z_1 = g_1(L^*, R^*, C^*) + jp_1(L^*, R^*, C^*). \tag{5.9}$$

Since both the real and imaginary parts of Z_1 and Z_2 are functions of all their respective passive elements, there is no unique choice of L^*, R^*, C^*, and therefore, K^*, D^*, M^* in the single mass-spring-damper system that produces the same impedance as the two mass-spring-damper system. There are an infinite number of combinations of components that produce the same impedance result. Therefore, in the three dimensional mesh case, it is more than likely that there is no unique solution to Equation (5.4). As a result, numerical optimization techniques can be used to find a "more" optimal solution than just arbitrarily choosing parameters.

Equation (5.4) is a optimization problem with nonlinear constraints, i.e. $\dot{x} = f(x, \theta)$, and as a result nonlinear optimization methods are required. As previously explained, Equation (5.4) is most likely non-convex, having multiple minima, and a global minimum that is difficult to find. Therefore, convex solvers cannot be used. Since the gradient of the minimization problem, with respect to minimization variable, θ, is not easily solvable, a gradient descent solver is not applicable. To obtain a solution to the minimization problem, a heuristic based, nonlinear optimization algorithm can be used. Heuristic methods can find global optimum under certain conditions. For example, in simulated annealing, the probability of finding the global optimum approaches 1 as time increases. This is not helpful as the time required will likely exceed the time required for a complete search of the space [32]. Under normal conditions, heuristic optimization techniques may find the optimum of a function, but are not guaranteed to converge to an optimal, or even close to optimal solution. Heuristic approaches do, however, attempt to move away from local minima to find a global minimum. Heuristic based nonlinear optimization techniques include simulated annealing, genetic algorithms and swarm algorithms, among others. A number of these techniques have been used for data driven parameter identification for mass spring models. Data driven parameter identification, as discussed in Section 6.4, is a method of finding mass and spring constants that allow a simulation model to match experimental data. Genetic algorithms [14] and evolutionary algorithms [47] have been used in data driven parameter identification producing positive results. However, they require large amounts of preprocessing to create population sets. The method of simulated annealing was used in [27] to get good nodal approximations of deformable bodies. Each method has trade-offs regarding computation time and optimal convergence. In [46], the author mentions data driven approaches only work well for identifying parameters of smaller meshes. Therefore, other approaches, such as finding analytic expressions for solutions, have been researched. Studies that try to match the elastic properties of a mass spring model to

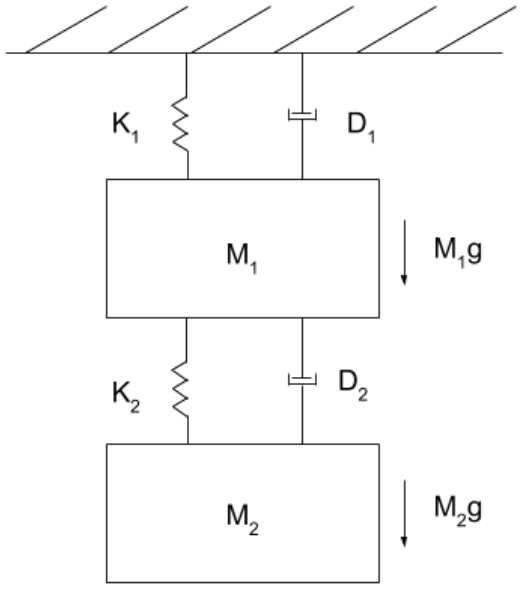


(a) Single mass-spring-damper system

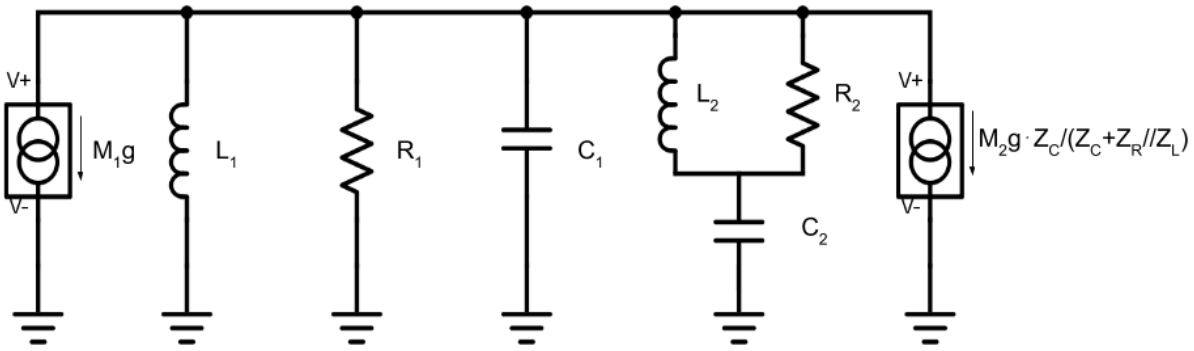


(b) Electrical circuit equivalent

Figure 5.1: Electrical analogy of single mass-spring-damper system



(a) Two mass-spring-damper system



(b) Electrical circuit equivalent

Figure 5.2: Electrical analogy of two mass-spring-damper system

those of a real deformable object, such as [61], often require the two volumes to be equal. In the case of mass spring model compression, it would imply the sum of the masses of the compressed model to be equal to the sum of the masses of the original model. However, for the purpose of this study, it is not necessary for the mass of the sparse model to be equal to the mass of the original model. This is because the behaviour of the model is more important, for the purpose of surface projection, than volume matching.

For this study, Equation (5.4) is solved for using the simulated annealing heuristic algorithm. Simulated annealing is used due to its fast computation time compared to other heuristic based algorithms. Large sets of solutions do not have to be preprocessed, as in genetic and evolutionary algorithms, since creating a population is an expensive task. Instead, during each iteration, only one, likely more optimal, solution is found. The simulated annealing algorithm mimics the physical concept of annealing materials. Annealing is the thermal process of achieving a low-energy state in a solid, making the material less fragile. By cooling the material at a specific rate, the structural properties of the material change. Generally, the material is heated until it reaches its *annealing temperature*, and the temperature is even throughout. The material is then cooled slowly to minimize the internal energy [49].

The simulated annealing algorithm begins by taking an initial guess of the parameter vector, θ_0 and finding its associated cost. In this case, the associated cost, $C(\theta)$, is set to be the minimization term

$$C(\theta) = \sum_{i=0}^{N} \|Hz[i] - \mathbf{A}Hx[i]\|_2^2. \tag{5.10}$$

The implicit backwards Euler method is used to compute the vector $z[i]$, as it is far more computationally efficient than explicit integration methods. After finding the initial cost term, $C(\theta_0)$, zero-mean Gaussian noise, W, is added to each parameter in θ_0 to create the new candidate solution θ_1,

$$\theta_1 = \theta_0 + W. \tag{5.11}$$

The cost of the candidate solution is then calculated, $C(\theta_1)$, and is compared to the the cost of the initial guess. If $C(\theta_1) \leq C(\theta_0)$, then θ_1 is taken as the new optimal solution. If $C(\theta_1) > C(\theta_0)$, θ_1 is chosen as the new optimal solution with a probability

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

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 the so-called 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 continually repeated until a termination condition is reached. After reaching a termination condition, such as a minimum value for T, the final solution θ_N is obtained. θ_N 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.

The simulated annealing algorithm is used here to compress a 21×21 mesh into a 5×5 mesh. The 21×21 mesh is also compressed into a 11×11 mesh, to study how the number of states in the sparse model effects the model behaviour. It is expected that with fewer nodes, matching the dynamics of the original system will be more difficult. For the first iteration of the simulated annealing algorithm, the initial parameter vector θ_0 begins with all k_s being the same value, all k_d being the same value, and all m being the same value. Figure 5.3 shows the cost of each solution of the 5×5 mesh compression over the iterations of the simulated annealing algorithm. The results show the cost of the final solution, $C(\theta_N^5)$, is approximately 66% less than the cost of the initial solution. The associated mass, spring and damper distributions for θ_N^5 are shown in Figure 5.4. The costs of the 11×11 compression solutions are shown in Figure 5.5 and the parameter distributions of θ_N^{11} is shown in Figure 5.6.

The results in Figure 5.7 show that the denser 11×11 mesh has far less error when compared to the original system than 5×5 model. Here, the mean squared error between the compressed and original systems is plotted. The 11×11 model produces 10 times less error than the 5×5 model throughout the simulation. This also holds true when comparing the cloth motion visually. The low inertia of the sparser 5×5 model, due to the lighter mass, causes sluggish behaviour over time. Although the 11×11 model is also slightly sluggish, its dynamics compare far more favourably to the 21×21 configuration. To obtain less sluggish results from the 5×5 model, the cost function in Equation (5.4) is replaced with a function that gives exponentially larger weighting to future data. The is done to keep the two sets of data closer together for a longer period of time. The new minimization problem is given in Equation (5.12).

$$\begin{aligned} \min_{\theta} \quad & \int_0^t e^{\alpha\tau} \|Hz(\tau) - \mathbf{A}Hx(\tau)\|_2^2 d\tau \\ \text{s.t.} \quad & \dot{z}(t) = f(z, \theta). \end{aligned} \tag{5.12}$$

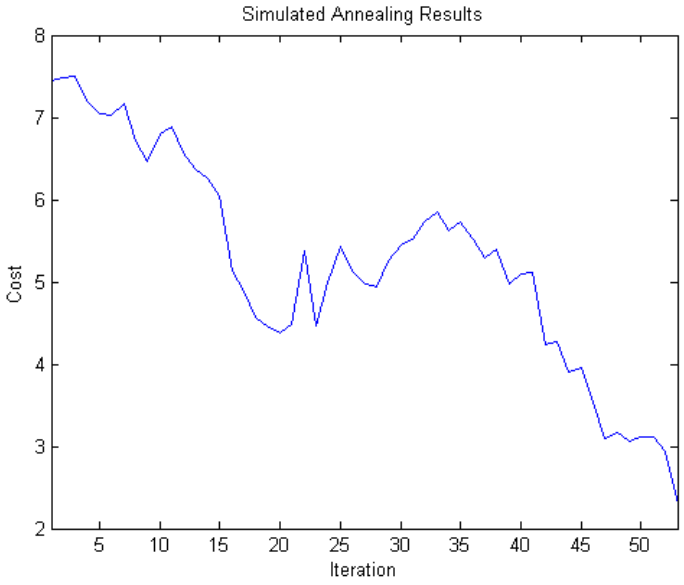


Figure 5.3: Simulated annealing costs for compression to 5×5 model

Here α is a scaling factor which determines the weighting of future time data. The associated cost function for optimization is

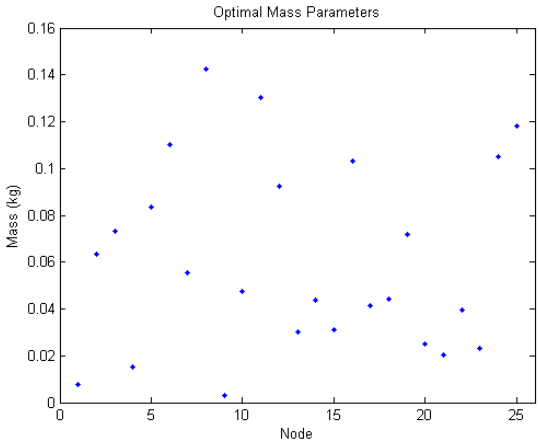
$$C(\theta) = \sum_{i=0}^{N} e^{\alpha i \Delta T} \|Hz[i] - \mathbf{A}Hx[i]\|_2^2, \tag{5.13}$$

where ΔT is the step time of the integration that solved $x[i]$ and $z[i]$.

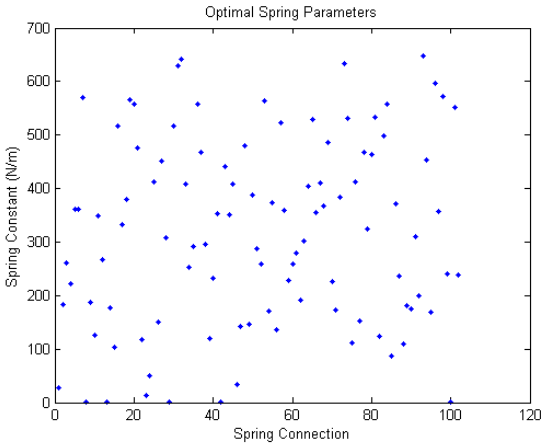
It can also be seen from the circuit analogy in Figure 5.2, that the current source, and therefore the external force, changes when the Norton equivalent circuit is found. The overall force applied to the new mass becomes the sum of the two current sources, i.e.

$$F_{ext} = I_{eq0} = M_1\vec{g} + M_2\vec{g}\frac{Z_C}{Z_C + Z_R//Z_L}, \tag{5.14}$$

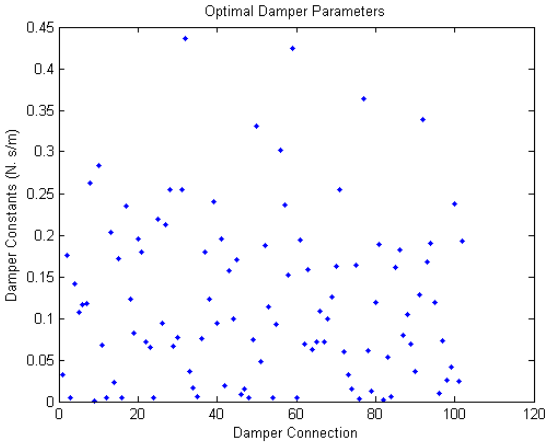
where $Z_{(\cdot)}$ is the impedance of the specified circuit element, and $\vec{g}$ is the acceleration due to gravity. This shows that even in the one dimensional linear case, the new force is a linear combination of external forces on the two original masses. In the case where more masses are linked together, this would likely imply that the force on the compressed system masses are a linear combination of all external forces. Extending this to the mass spring model,



(a) Solution masses



(b) Solution spring constants



(c) Solution damper constants

Figure 5.4: Final parameter vector θ_N values for 5×5 compression

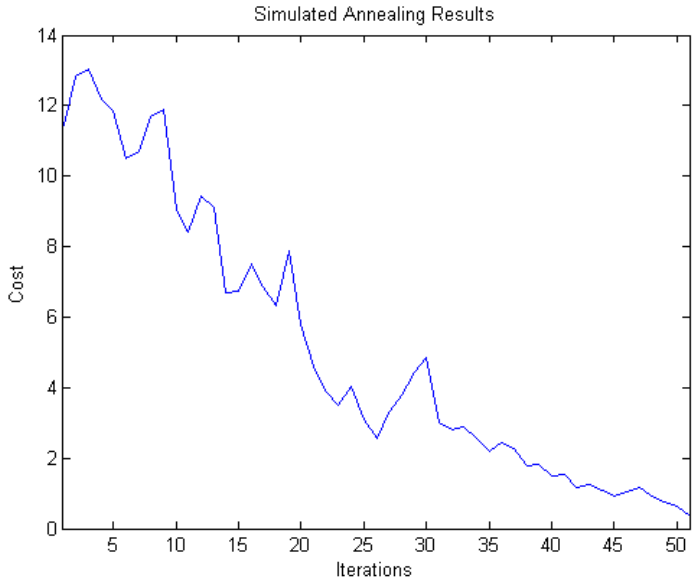
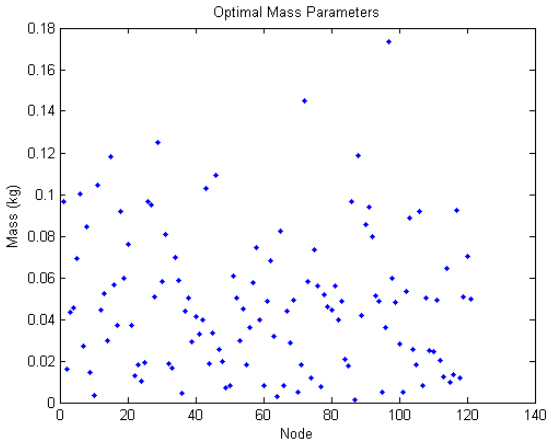


Figure 5.5: Simulated annealing costs for compression to 11×11 model

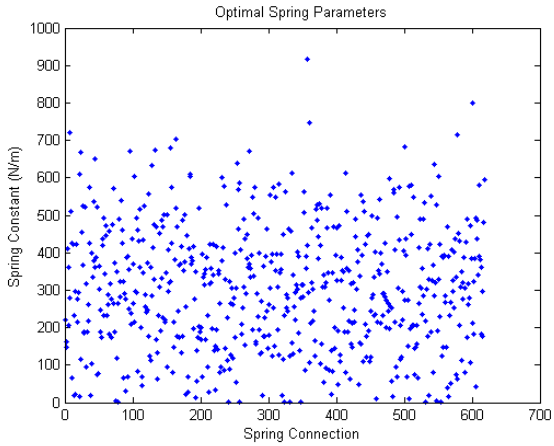
each node in the compressed system would likely have an additional amount of external force applied onto it. This additional force is some combination of the external forces applied to the surrounding nodes of the original system. As the exact set of nodes from the original model that effect the nodes of the compressed model is unknown, new scaling parameters G_i for $i = 1 \ldots n$ are added to the parameter vector θ. These parameters are applied as follows

$$F_{ext_i} = G_i m_i \vec{g}, \tag{5.15}$$

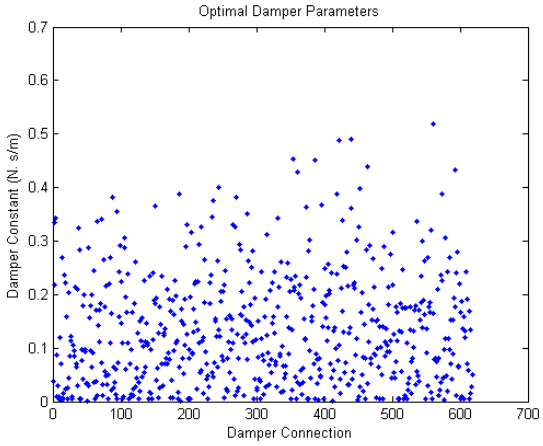
as the only external force on the falling cloth model is the force due to gravity. The new parameter vector θ now contains $2(s+n)$ elements: s spring constants, s damper constants, n masses, and n scaling terms. The results of the simulated annealing with the new cost function (Equation (5.13)), where $\alpha = 0.001$, and extra parameters are shown to in Figure 5.8. Here, the 21×21 model is compressed to a 5×5 model where the initial parameter vector θ_0 contains identical values for all spring constants, damper constants, masses, and scaling factors, respectively. The distribution of parameters in θ_N are shown in Figure 5.10. The final simulated annealing cost is, again, significantly lower than the initial cost (by 94%) when the parameters were chosen to be equal. The mean squared error between the compressed and original model is shown in Figure 5.9. Also shown in Figure 5.9 is the mean squared error of the 11×11 model using the parameters from Figure 5.5. The



(a) Solution masses



(b) Solution spring constants



(c) Solution damper constants

Figure 5.6: Final parameter vector θ_N values for 11×11 compression

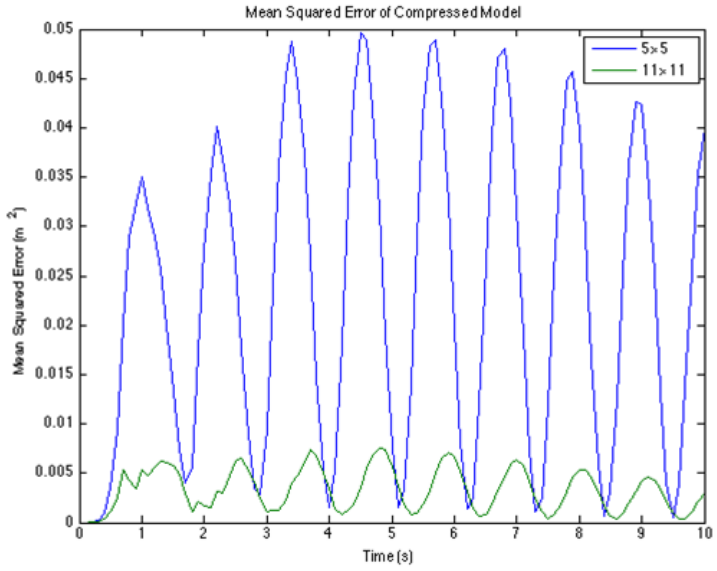


Figure 5.7: Mean squared error between compressed models and the original model

11×11 compression results do not use the new parameter vector and cost function. These are the same results as the first iteration of the simulated annealing algorithm.

Using the extra parameters and new cost function, the error reduced by a factor of 20 compared to solution found using the old cost function. As can be seen by comparing Figures 5.7 and 5.9. Additionally, the compression to a 5×5 model now produces better results than the compression to a 11×11 model using the previous cost function. The error associated with the new 5×5 model is now less than 50% of the error associated with the 11×11 model.

The problem with using this solution for general model compression, is that it specific to only one scenario. As a result, the solution may be overtrained to a single model. To determine whether this solution has been overtrained, the parameter set is applied to a model with different initial conditions. Figure 5.11a (red line) shows the error of using this parameter set with the new 5×5 simulation. The error here is quite large compared to the original simulation. To combat this, a new cost function compensating for both simulation scenarios is used. For simulation scenarios S_1 and S_2, the associated cost function becomes

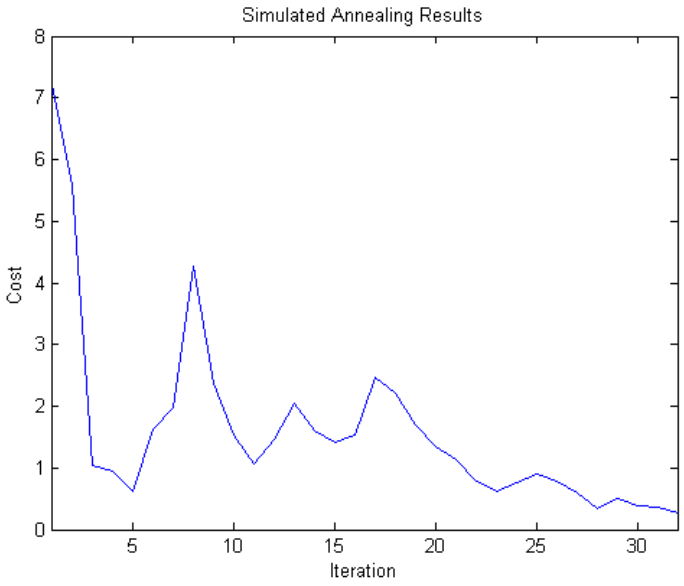


Figure 5.8: Updated simulated annealing costs for compression to 5×5 model

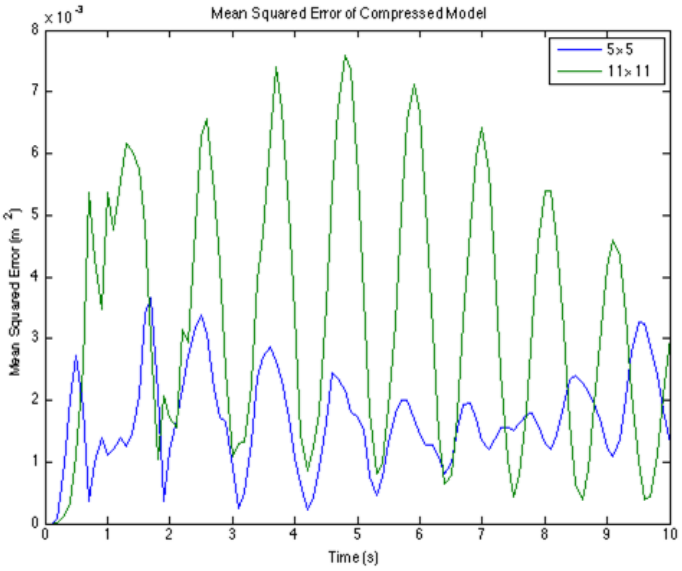
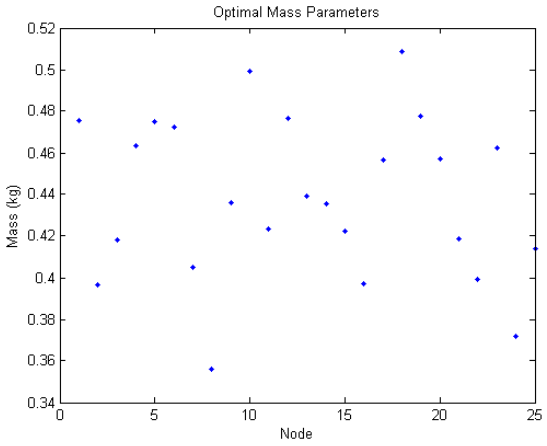
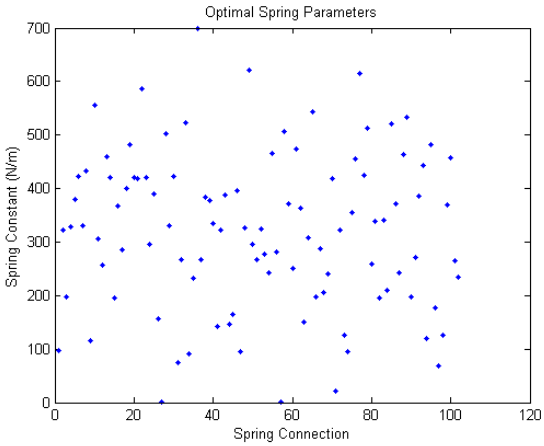


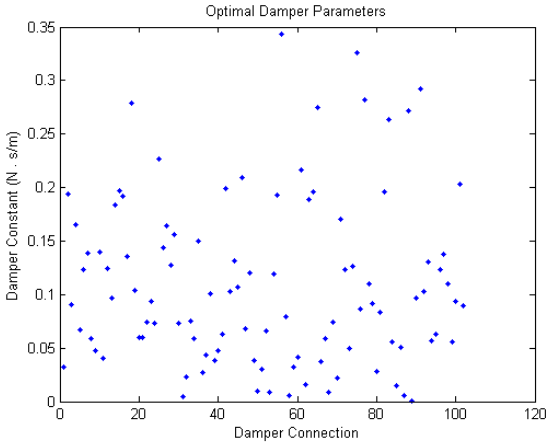
Figure 5.9: Mean squared error between compressed models and the original model with new cost function



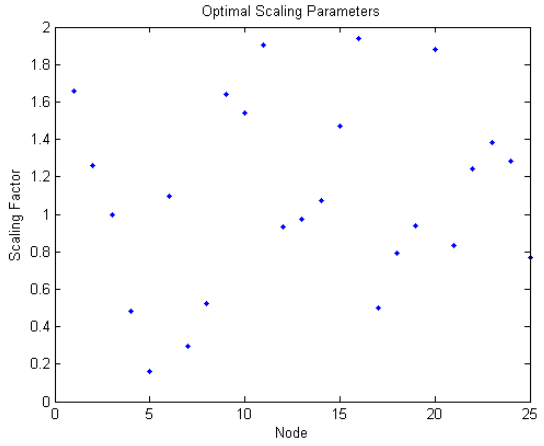
(a) Solution masses



(b) Solution spring constants



(c) Solution damper constants



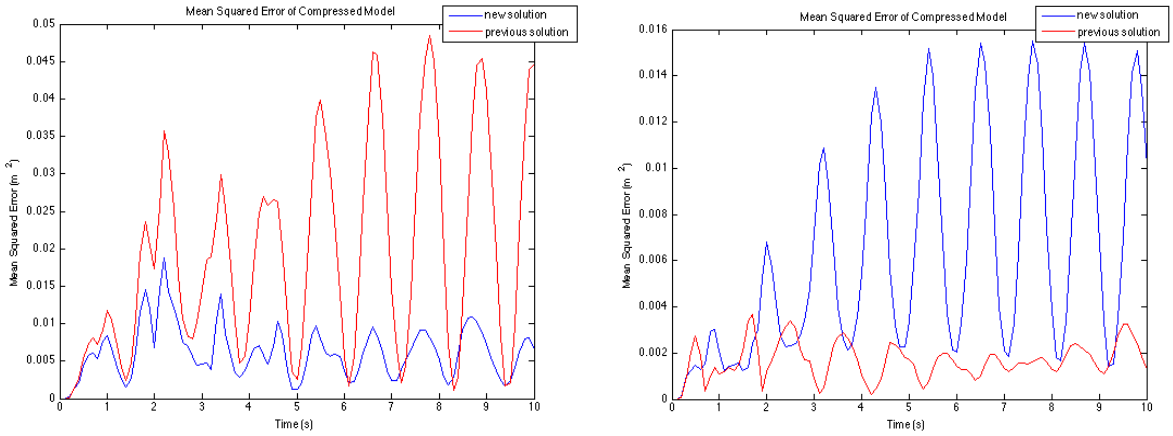
(d) Solution scaling factors

Figure 5.10: Updated final parameter vector θ_N values for 5×5 compression

the sum of the two cost functions $C_{new}(\theta) = C_{S_1}(\theta) + C_{S_2}(\theta)$, or

$$C_{new}(\theta) = \sum_{i=0}^{N} [e^{\alpha i \Delta T} \| H z_{S_1}[i] - \mathbf{A} H x_{S_1}[i] \|_2^2 + e^{\alpha i \Delta T} \| H z_{S_2}[i] - \mathbf{A} H x_{S_2}[i] \|_2^2]. \tag{5.16}$$

Using this new solution, the error of the second scenario compared to the error using the old solution, is shown in Figure 5.11a. As can be seen, the error is nearly 4 times lower using this new solution. The results of using the new solution in the original model is shown in Figure 5.11b (note the difference in scale between Figures 5.11a and 5.11b). Although the error using this parameter set is larger than using the previous parameter set, the error is still significantly lower than using the solution from Figure 5.6. Therefore, to prevent overtraining, the cost function for simulated annealing needs to incorporate multiple models. Applying more than two models to the cost function will likely result in an even more general solution.



(a) New scenario error plots

(b) Original scenario error plots

Figure 5.11: Error plots when using two scenario cost function

The results of the model compression in this chapter justify that a reduced order model can be used in the EKF almost as effectively as a larger model. By using heuristic optimization techniques, the behaviour of sparse models can almost exactly match the behaviour of the dense, original model. It is, therefore, reasonable to assume that when working with dense state models, these models can be reduced to smaller models that have similar dynamic characteristics. As a result, for the last chapter of this thesis, reduced order mass spring models will be used to validate the EKF as a surface predictor. The models used

in Chapter 6 will all have fewer than 25 nodes, compared to the 441 nodes used for cloth simulations in Chapter 3.

Chapter 6

Estimation Filtering Applied to Model

In this chapter, the extended Kalman filter, described in Section 4.3, is applied to the mass spring model, derived in Chapter 3, to create a prediction filter for deformable surface motion. The EKF is first formulated using the Jacobian of the mass spring model found in Section 3.2 and is applied to two different simulation scenarios. These scenarios use the model simulation techniques from Section 3.3 with added disturbances in the form of Gaussian noise and random viscous forces. For both simulation scenarios, the noise covariance parameters of the EKF are tuned to minimize the mean-squared error between the predictions and the simulation outputs. The EKF is applied to real experimental data of cloth movement to predict its motion. The results are discussed, as well as strategies for finding mass, spring, and damper parameters to best match the model to the experimental data.

For spatial augmented reality applications, where a projector is projecting images on moving surfaces, the EKF predictor may not be fast enough to compensate for inter-frame perturbations. As a result, an inter-frame prediction technique is derived to compensate for any of these disturbances.

6.1 Formulation of Filter

To combine the EKF and the deformable mass spring model, the model needs to be linearized using a first order approximation. This implies that the Jacobian derived in section

3.2 must be used in place of the state transition matrix described in Section 4.3. In other words, given the state transition function

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

the state transition matrix, F_k, must be

$$F_k = J(\hat{x}_{k|k}) = \frac{\partial f}{\partial x}|_{\hat{x}_{k|k}}. \tag{6.1}$$

The prediction step of the EKF can therefore be written as:

$$\begin{aligned} \hat{x}_{k|k-1} &= f(\hat{x}_{k-1|k-1}) \\ P_{k|k-1} &= J(\hat{x}_{k-1|k-1})P_{k-1|k-1}J(\hat{x}_{k-1|k-1})^T + Q_{k-1}, \end{aligned} \tag{6.2}$$

where the initial conditions are $\hat{x}_{0|0} = x[0]$ and $P_{0|0} = \mathbf{0}$, assuming the initial state is given, and $Q_k = \alpha\mathbf{I}\ \ \forall k \in \mathbf{Z}_+$, where $\alpha \in \mathbf{R}_+$ (positive real numbers). A diagonal matrix is chosen for Q_k as the process noise vectors are assumed to be independent random vectors. Here, the system is linearized about the current state estimate, and therefore, is less accurate for deviations far from the current state estimate.

Since the mass spring model is a continuous time state space model, having the form

$$\frac{dx}{dt} = f(x), \tag{6.3}$$

the model needs to be discretized to allow for the use of the discrete time EKF. To discretize the model, a first order Euler approximation is used, i.e.,

$$x[k+1] = x[k] + T_s f(x[k]), \quad x[0] = x(t_0), \tag{6.4}$$

or in the state prediction step of Equation (6.2),

$$\hat{x}_{k|k-1} = \hat{x}_{k-1|k-1} + T_s f(\hat{x}_{k-1|k-1}). \tag{6.5}$$

In Equations (6.4) and (6.5), T_s is the sampling time for the discretization of Equation (6.3). As the EKF uses this discretized model in its formulation, it results in the EKF providing a state prediction every T_s seconds.

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. The update step is as follows:

$$\begin{aligned}
\tilde{y}_k &= y[k] - H_k \hat{x}_{k|k-1} \\
S_k &= H_k P_{k|k-1} H_k^T + R_k \\
K_k &= P_{k|k-1} H_k^T S_k^{-1} \\
\hat{x}_{k|k} &= \hat{x}_{k|k-1} + K_k \tilde{y}_k \\
P_{k|k} &= (\mathbf{I} - K_k H_k) P_{k|k-1}.
\end{aligned} \tag{6.6}$$

Comparing Equation (6.6) to Equation (4.21), the output function $h_k(x)$ is a linear operator selecting only the position states of x at time step k. Therefore, $h_k(x) = H_k x$, where

$$H_k = H = \begin{bmatrix} \mathbf{I}_{3n \times 3n} & \mathbf{0}_{3n \times 3n} \end{bmatrix} \qquad \forall k = 0, 1, \ldots \tag{6.7}$$

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, $R_k = \beta \mathbf{I} \;\; \forall k = 0, 1, \ldots, \beta \in \mathbf{R}_{++}$ (strictly positive numbers). A diagonal matrix is chosen for R_k as the measurement noise vectors are assumed to be independent random vectors.

The initialization of the EKF recursion is shown in Equations (6.8)-(6.11). First, a state prediction is made using the current state estimate vector, $\hat{x}_{k|k}$, initialized with the given initial conditions. The state covariance prediction is then made by linearizing the model about the current state estimate. The measured outputs are then combined with the prediction results to obtain an updated state estimate and state covariance matrix. The process is continually repeated until termination.

$$\text{Prediction Step 1: } \begin{cases} \hat{x}_{1|0} = \hat{x}_{0|0} + T_s f(\hat{x}_{0|0}) \\ P_{1|0} = J(\hat{x}_{0|0}) P_{0|0} J(\hat{x}_{0|0})^T + Q_0 \end{cases} \tag{6.8}$$

$$\text{Measure Step 1: } \begin{cases} x[1] = x[0] + T_s f(x[0]) \\ y[1] = H x[1] \end{cases} \tag{6.9}$$

$$\text{Update Step 1: } \begin{cases} \tilde{y}_1 = y[1] - H \hat{x}_{1|0} \\ S_1 = H P_{1|0} H^T + R_1 \\ K_1 = P_{1|0} H^T S_1^{-1} \\ \hat{x}_{1|1} = \hat{x}_{1|0} + K_1 \tilde{y}_1 \\ P_{1|1} = (\mathbf{I} - K_1 H) P_{1|0} \end{cases} \tag{6.10}$$

$$\text{Prediction Step 2: } \begin{cases} \hat{x}_{2|1} = \hat{x}_{1|1} + T_s f(\hat{x}_{1|1}) \\ P_{2|1} = J(\hat{x}_{1|1}) P_{1|1} J(\hat{x}_{1|1})^T + Q_1 \end{cases} \tag{6.11}$$

$$\vdots$$

6.2 Results

To determine the effectiveness of the extended Kalman filter when applied to the mass spring model, two simulation scenarios are chosen for EKF implementation. In the first scenario, the EKF is applied to a square, 5×5 node cloth configuration with Guassian noise added to the derivative state vector $\dot{x}$. The Gaussian noise is of zero-mean with variance of $0.001 \frac{\text{m}^2}{\text{s}^2}$ added to the velocity states and $0.001 \frac{\text{m}^4}{\text{s}^4}$ added to the acceleration states. Since the noise is added directly into $\dot{x}$, the noise acts as process noise in the system, i.e. $w_k \sim N(0, 0.001)$ and are independent and identically distributed for all $k = 1 \ldots$. The parameters for the mass spring model are chosen arbitrarily and are as listed in Table 6.1.

Table 6.1: 5×5 Node Model Parameters

Parameter	Value
Total Length	1.5m
Total Width	1.5m
Spring Constants k_{s_i}	300N/m
Damper Constants k_{d_i}	0.08N·s/m
Masses m_i	0.441kg

For the above scenario, each node has the same mass, and all node connections have the same damper and spring constants. Since the model is a square 5×5 mesh, the nodes are evenly spaced about the length and width of the cloth. This results in a 0.375m separation between horizontally and vertically adjacent nodes. During the simulation, the cloth starts at an initial position parallel to the ground, and is dropped while the top nodes are kept at a fixed position; similar to anchoring a flag to a horizontal pole. The motion of the cloth is like that of a pendulum.

For the EKF formulation of the system, the values chosen for the noise covariance matrices, R_k and Q_k, are $\mathbf{I}$ and $0.001\mathbf{I}$, respectively, for all $k \in \mathbf{Z}_+$. The process noise covariance matrix Q_k was chosen to be $0.001\mathbf{I}$ as this is exactly the variance of the noise added to the vector $\dot{x}$. Since the noise added to the system is in fact w_k, and Q_k is

the covariance of w_k, this is the obvious choice of parameter. The observation covariance matrix R_k, on the other hand, was chosen arbitrarily. The initial state covariance matrix was set to be $P_{0|0} = \mathbf{0}$, since the initial states of the EKF model were set equal to the initial states of the simulation model, i.e. $\hat{x}_{0|0} = x[0]$. Figure 6.1 shows snapshots of the EKF position state predictions overlaid on the measured outputs for four time instances of the Runge-Kutta simulation. The blue dots represent the measured positions of the cloth nodes (simulation outputs), while the red dots represent the EKF predicted positions of the cloth nodes. Figure 6.1a shows that there is no difference between outputs and predictions at the start of the EKF recursion because the initial configuration of the EKF is set to the same positions as the simulation model. As the EKF recursion progresses, the error is quite large at first (Figure 6.1b); however, it visually decreases through the completion of the recursion (Figures 6.1c and 6.1d).

Figure 6.2 shows the mean-squared error between the EKF predicted position states and the simulation position states for each time step of the EKF. The mean squared error is defined as,

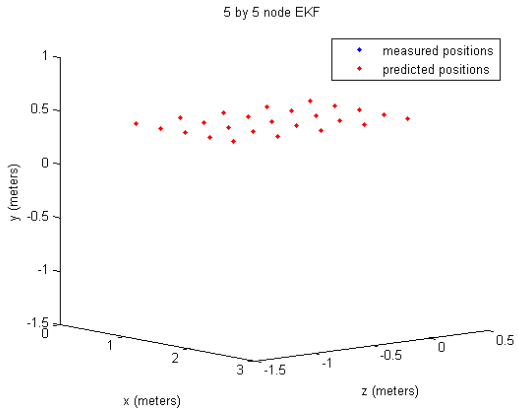
$$\mathbf{e}[k] = \frac{1}{n}(\hat{\mathbf{p}}[k] - \mathbf{p}[k])^T(\hat{\mathbf{p}}[k] - \mathbf{p}[k]), \tag{6.12}$$

where n is the number of position states, $\hat{\mathbf{p}}[k]$ are the predicted position states and $\mathbf{p}[k]$ are the measured position states. Since the Kalman filter finds an estimate which minimizes the squared error between the state estimates and the expected states, it is expected that $\mathbf{e}[k]$ will decrease as k increases. It can be seen from Figure 6.2 that the mean squared error converges to approximately $5 \times 10^{-4}\text{m}^2$ after running the EKF for 10 seconds. Just as in Figure 6.5, the same trend of error reduction can be seen in this plot. The error decreases almost exponentially over time, with intermittent spikes in error near the start of the EKF recursion. By taking the square root of the mean-squared error, i.e. root mean-squared error, an approximation of the average distance between each predicted node's position and the associated simulation node's position is be made. Therefore, the predicted nodes were on average 3.9cm away from the measured values in each direction after 10 seconds of running the EKF. This represents an error that that is 2.6% of both the length and width of the cloth.

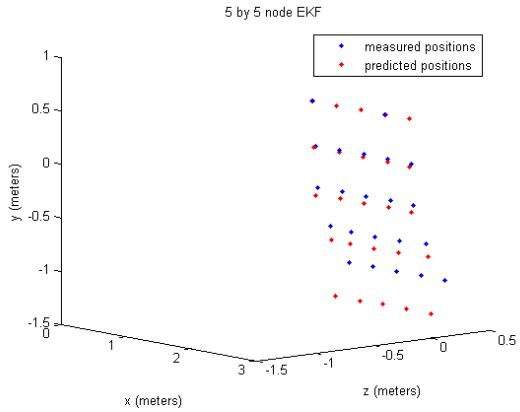
To obtain a more physical representation of the effectiveness of the EKF predictor, the Euclidean error between the EKF prediction and the measured position of a node i is computed. The Euclidean error is given by

$$e_i[k] = \|\hat{p}_i[k] - p_i[k]\|_2, \tag{6.13}$$

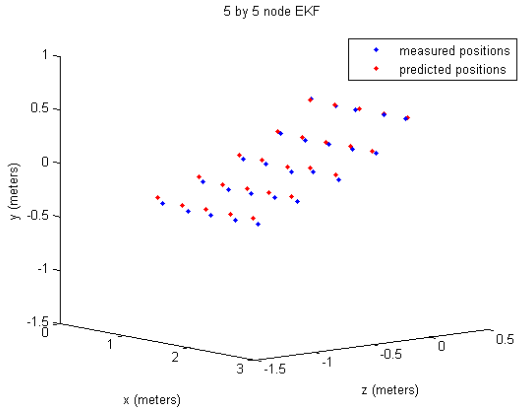
where $\|\cdot\|_2$ is the 2-norm operator, and gives the distance between the predicted node and



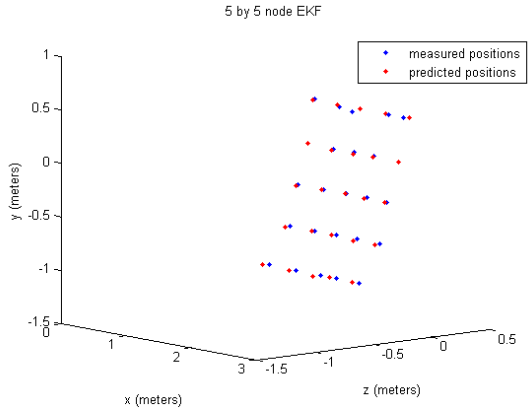
(a) Initial cloth position



(b) Cloth position at 0.8 seconds



(c) Cloth position at 5 seconds



(d) Cloth position at 10 seconds

Figure 6.1: EKF applied to 5×5 node mass spring cloth model with added zero-mean Gaussian noise

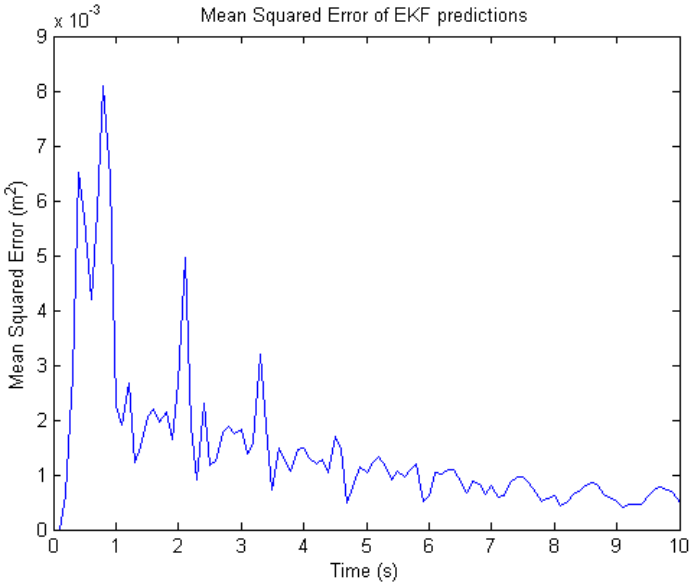


Figure 6.2: Mean squared error between EKF predictions and simulation outputs over time

measured node. For perfect prediction, the distance between predicted node and measured node should be zero, implying error $e_i[k] = 0$. Figure 6.3 shows the Euclidean error of the EKF prediction for a single node over time. The bottom corner node of the cloth is analyzed in Figure 6.3, as it has the largest peak in Euclidean error. This is the worst case node for this EKF implementation. Near the start of the EKF recursion, at 0.8 seconds, the error is very large, with the distance between the predicted node and measured node being of over 30cm. As the recursion continues, the error decreases significantly, to approximately 5cm after 10 seconds. This shows that the position of worst case node can be predicted reasonably accurately after allowing the algorithm to run for some time.

The second simulation scenario that the EKF is implemented on is when the mass spring surface is acted upon by a random viscous force defined as

$$F_i = k_{vis_i}(\hat{n}_i \cdot (u - v_i))\hat{n}_i. \tag{6.14}$$

Here, $\hat{n}_i$ is the unit vector normal to the surface at a specific node i, u is a random $\mathbf{R}^3$ vector representing the viscous entity's velocity in meters per second, such as wind [54], and v_i is the velocity of node i. The viscous force is applied to each node (Figure 6.4), excluding the anchoring nodes, in addition to all other internal and external mesh forces. In this simulation scenario, the cloth is once again anchored along the top row of nodes,

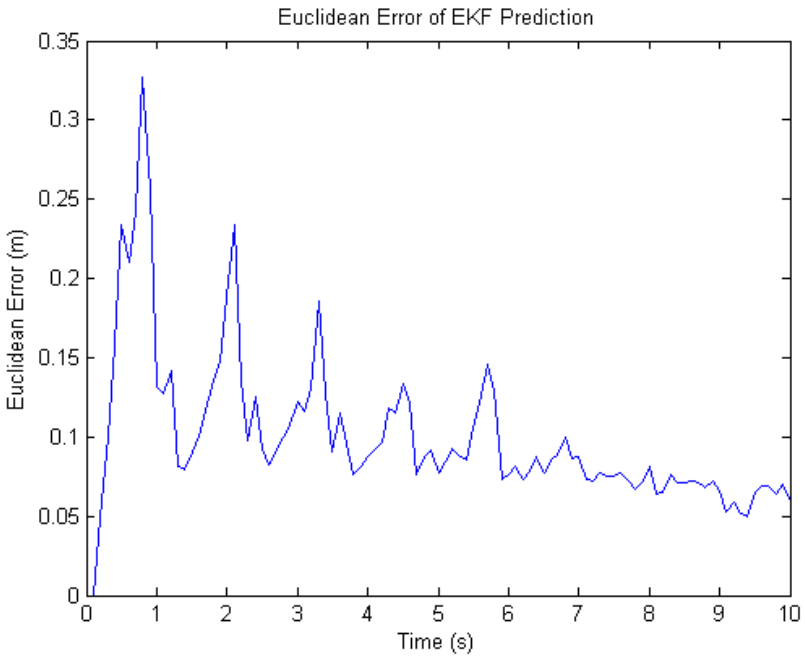


Figure 6.3: Euclidean error of EKF prediction of single node over time

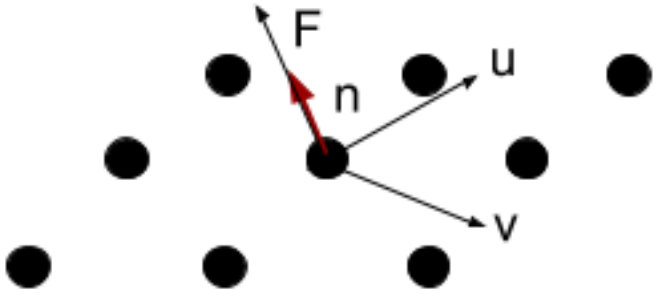


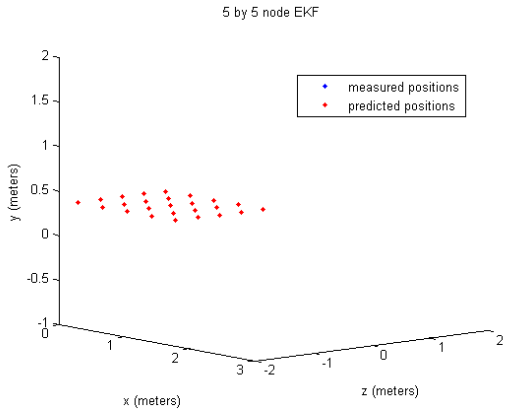
Figure 6.4: Random viscous force applied to cloth node

similar to hanging a flag on a pole, and dropped while a viscous force is applied. Using the same masses, cloth dimensions, and spring and damper constants as the first scenario, the simulation outputs and EKF predictions of the viscous force model are shown in Figure 6.5. In this simulation, the random input u is chosen to be $\begin{bmatrix}\alpha & 0 & \beta\end{bmatrix}^T$ where both α and β are random variables chosen from a uniform distribution $U(1\text{m/s}, 10\text{m/s})$. The constants k_{vis_i} are chosen to be 1 for all nodes i, and EKF parameters R_k and Q_k are chosen to be $\mathbf{I}$ and $100\mathbf{I}$ respectively. The large value of Q_k is chosen to compensate for the modelling difference between the EKF model and the simulation model. Since the EKF does not include the viscous forces in its model formulation, the effects of the viscous forces are assumed to be included in the modelling/process noise w_k. The initial state covariance matrix $P_{0|0}$ is set to $\mathbf{0}$ since the initial state estimate $\hat{x}_{0|0}$ is set to the same value as the initial state measurement $x[0]$, as can be seen from Figure 6.5a.

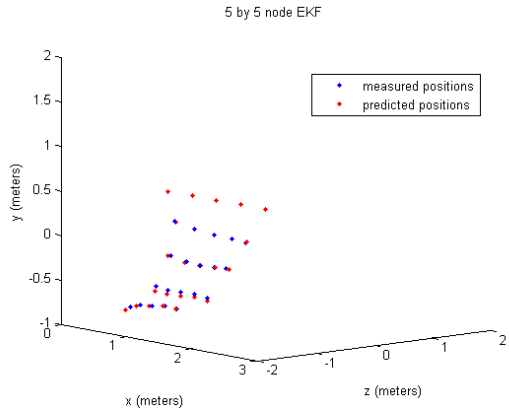
Just as in the the first scenario, the EKF predictor progressively improves as the simulation progresses (Figures 6.5b-6.5d). This is also confirmed by the EKF mean squared error plot shown in Figure 6.6. Each marker has, on average, $6 \times 10^{-4}\text{m}^2$ of error in each direction near the beginning of the EKF recursion. However, the error approaches zero after 10 seconds of running the EKF. Figure 6.7 shows the Euclidean error between a single predicted node and the associated measured output node, for the worst case scenario. The prediction error exponentially decreases from 10cm after 10 iterations (0.1 seconds) of the EKF recursion to less than 0.5cm after 100 iterations (10 seconds). Since the speed of the viscous force is not very large, the EKF performed well predicting the motion of the cloth. When the viscous force is chosen from a larger distribution, i.e. the viscous velocity u_i is chosen from a uniform distribution between $\begin{bmatrix}1 & 0 & 1\end{bmatrix}^T$ and $\begin{bmatrix}100 & 0 & 100\end{bmatrix}^T$ at every time-step, the EKF has a poorer performance predicting the position of the cloth as seen in Figure 6.8. The peak mean squared error is nearly 80 times larger compared to the slower viscous velocity case, at 0.05m^2. Additionally, the Euclidean error of every node (Figure 6.9) is very large; near or above 1m of error for some nodes during the EKF recursion. The EKF prediction error still exponentially decreases to a point where it converges to $5 \times 10^{-4}\text{m}^2$. The predictions settle after the initial transient effects from the large external forces dissipate. This shows that the EKF still predicts very accurately after running for a certain amount of time.

6.3 Parameter Tuning

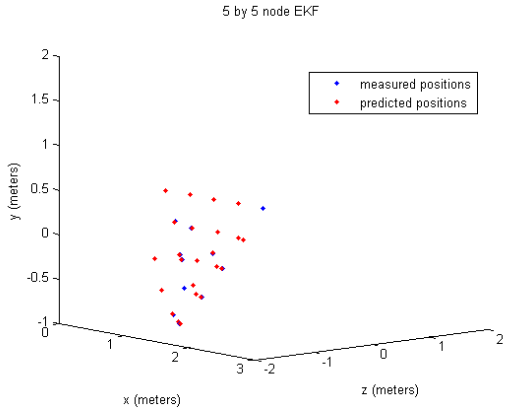
A well known drawback of the standard Kalman filter and the EKF is that they both require a priori knowledge of the process and measurement noise statistics [57] to produce



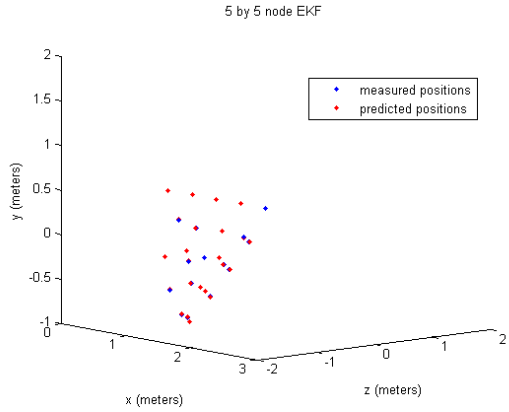
(a) Initial cloth position



(b) Intermediate cloth position



(c) Intermediate cloth position



(d) Intermediate cloth position

Figure 6.5: EKF predictions of 5×5 node mass-spring cloth model with random viscous force applied

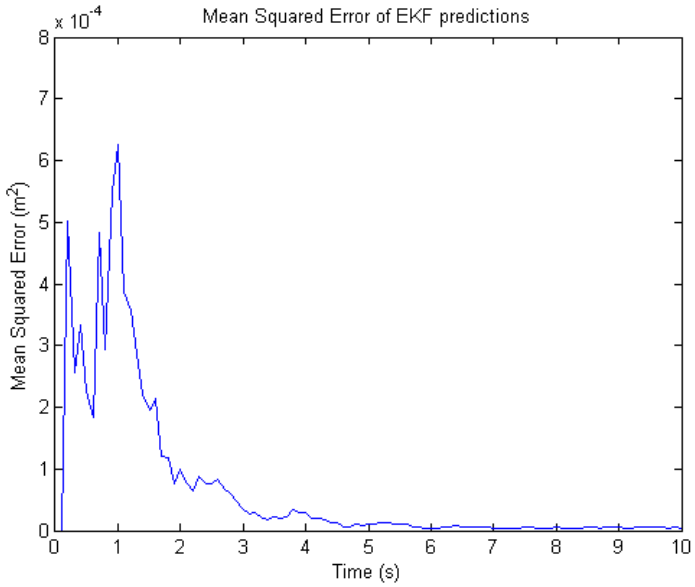


Figure 6.6: Mean squared error between EKF predictions and simulation outputs over time for random viscous force simulation

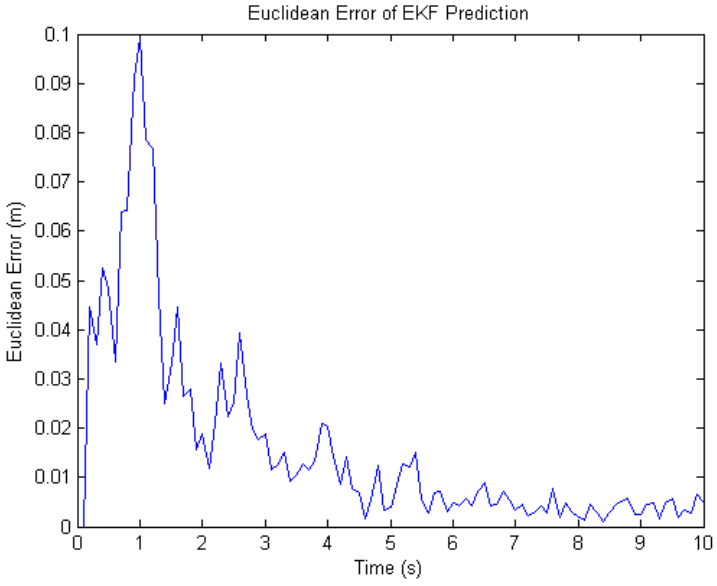


Figure 6.7: Euclidean error of EKF prediction of single node over time for random viscous force simulation

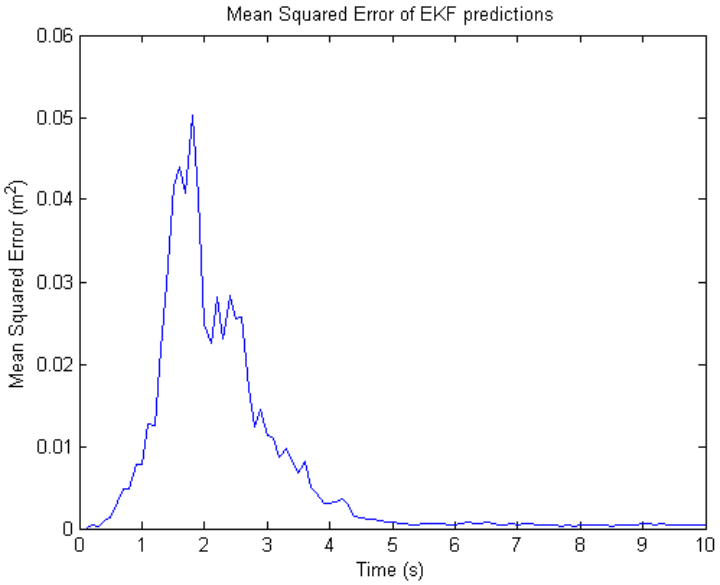


Figure 6.8: Mean squared error between EKF predictions and simulation outputs over time for large random viscous force simulation

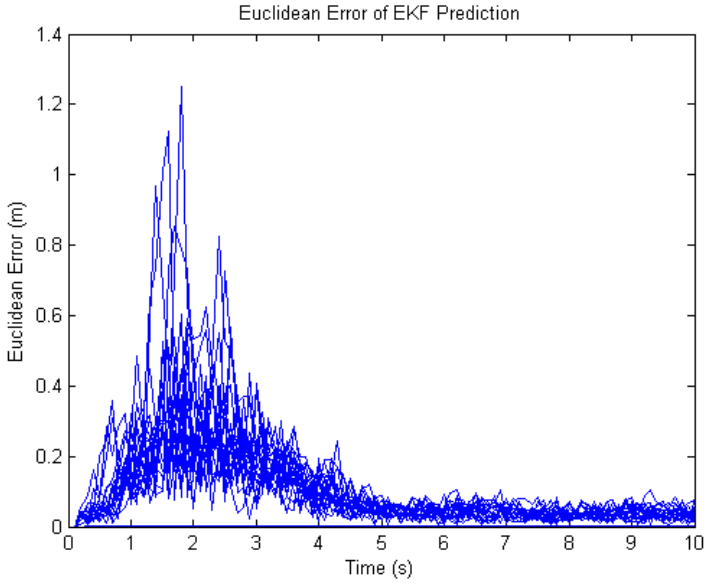
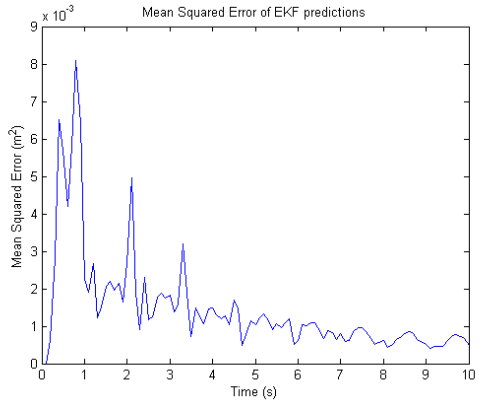


Figure 6.9: Euclidean error of EKF prediction of every node over time for large random viscous force simulation

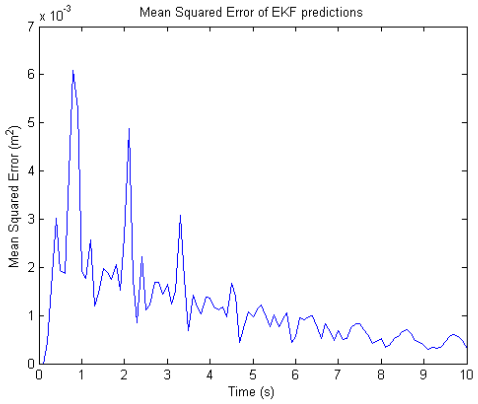
optimal results. In most practical applications, information about these statistics are difficult to obtain, and as a result, the covariance matrices need to be tuned to improve the estimation results. For the simulation scenarios in Section 6.2, the values of the noise covariance matrices, Q_k and R_k, along with the initial state covariance matrix, $P_{0|0}$, were chosen intuitively and arbitrarily. A more optimal set of parameters can be found that produce better prediction results from the EKF. Therefore, a trial and error approach is used select values of Q_k and R_k that minimize the EKF prediction error in a least squares sense.

Intuitively, for the scenario where Guassian noise is added directly into the position and velocity states, the best choice for the process noise covariance matrix Q_k is the variance of the added Gaussian noise itself. In this scenario, the simulation model is injected with zero-mean, Gaussian distributed process noise, w_k, with variance of $0.001\frac{\text{m}^2}{\text{s}^2}$ and $0.001\frac{\text{m}^4}{\text{s}^4}$ for velocity and acceleration states, respectively. Since the noise was added directly into the state derivative vector, this would imply the process noise covariance matrix is $Q_k = 0.001\mathbf{I} \quad \forall k \in \mathbf{Z}_+$. In other words, the diagonal terms are equal to the variance of the noise, while the off diagonal terms are equal to 0, as the random variables are independent. To find the best value for R_k, Q_k is kept fixed at $0.001\mathbf{I}$, while the value of R_k is varied to find the lowest mean squared error between predictions and simulation outputs. The results of this technique are shown in Figure 6.10. As R_k decreases below $\mathbf{I}$, the error consistently decreases until $R_k = 0.00001\mathbf{I}$. At this point, the peak mean squared error is approximately $3.5 \times 10^{-3}\text{m}^2$, which is lower than with any other tested choice of R_k. The small value of R_k can be attributed to the fact that the noise was only added as process noise, not as measurement noise. As a result large values of R_k are not necessary.

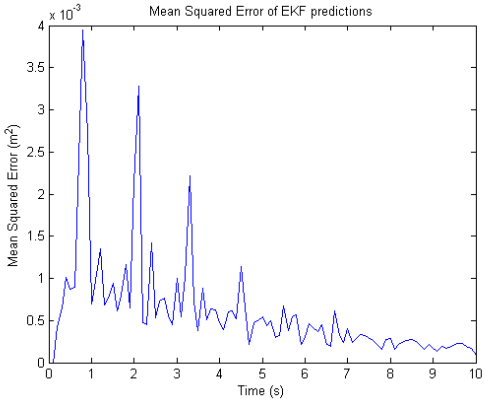
For the scenario where a random viscous force is applied to the mass spring model, there are no available statistics from which the parameters Q_k and R_k can be obtained. Tuning parameters for the EKF in this scenario, as like in most applications of EKFs, will result in parameters unique to the specific situation. As a result, finding the best parameters is an exercise in trial-and-error. Intuitively, since the viscous force is not included in the EKF model formulation, the elements of Q_k need to compensate for the discrepancies in modelling. Larger values of Q_k are required to compensate for larger uncertainties in the EKF model. Therefore, viscous forces with large velocities would likely require Q_k to have larger diagonal elements to compensate for the variance between the EKF and simulation models. To tune the EKF covariance matrices, R_k is first fixed and Q_k is varied. After finding the best choice for Q_k, R_k is then varied while Q_k remains fixed. Figure 6.11 shows a series of mean squared error plots of the EKF predictions, where R_k is kept fixed at $\mathbf{I}$, and Q_k is varied from $\mathbf{I}$ to $1000\mathbf{I}$. As expected, as Q_k increases, the mean squared error decreases in the early stages of the EKF recursion. However, there is a point of diminishing



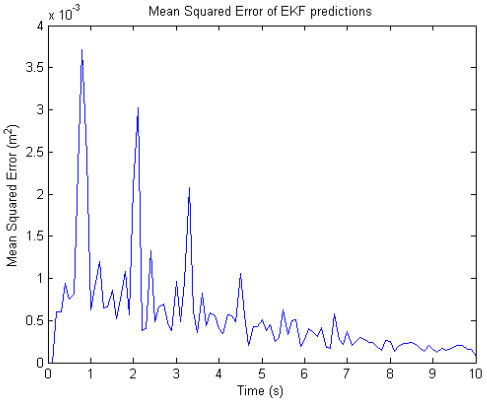
(a) EKF with $Q_k = 0.001\mathbf{I}$ and $R_k = 1\mathbf{I}$



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



(c) EKF with $Q_k = 0.001\mathbf{I}$ and $R_k = 0.0001\mathbf{I}$



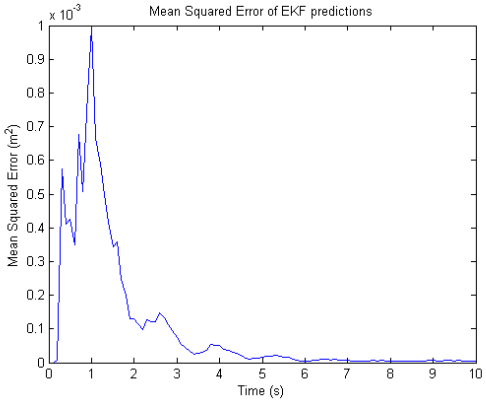
(d) EKF with $Q_k = 0.001\mathbf{I}$ and $R_k = 0.00001\mathbf{I}$

Figure 6.10: EKF MSE error for varying R_k in process noise scenario

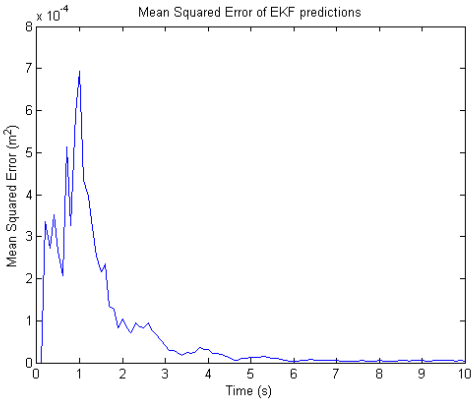
returns. For example, setting $Q_k = 100\mathbf{I}$ and $Q_k = 1000\mathbf{I}$ produce very similar prediction results. Therefore, Q_k can be set to $100\mathbf{I}$, and R_k is varied. When varying R_k, Figure 6.12, increasing the diagonal values past 1 decreases the initial prediction error. However, the peak error increases by nearly 15%. Furthermore, decreasing the diagonal elements below 1 does not result in significant changes in EKF prediction error. As a result, tuning the noise covariance matrices Q_k and R_k to $100\mathbf{I}$ and $\mathbf{I}$, respectively, produce the best prediction results. Using more sophisticated parameter tuning approaches, such as the Autocovariance Least-Squares technique [52], would likely result in values of Q_k and R_k that further increase the accuracy of the EKF.

6.4 Experimental Results

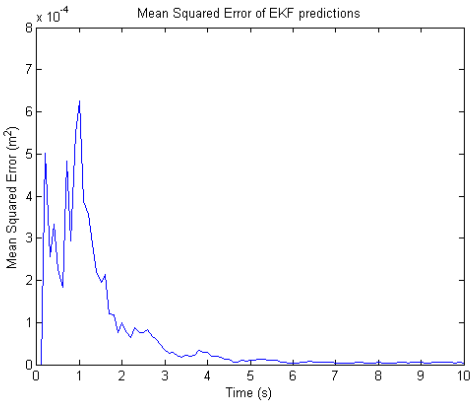
In order to validate both the mass spring model and the EKF formulation, implementation on real world data is required. A cloth-like material would be the obvious choice of surface to implement the algorithm on, as the simulation models from Section 3.3 behave like cloth. As a result, a towel is used for experimental data collection. Since the position of discrete points on the object is used as the measurement variable in the EKF, a sensor system that can capture positional data is required for data collection. A number of different techniques can be used for capturing positional data, such as image processing techniques or 3D camera systems, however, for greater data accuracy, a motion capture system is used in this experiment. The NaturalPoint OptiTrack system [6] is an infra-red camera based motion capture system system that provides positional data, both translational and rotational, within millimetre precision. The OptiTrack system measures the position of specific sized infra-red markers by triangulating each marker with multiple cameras in a known configuration. For this experiment, a four camera configuration is used to measure the position of 12.7mm diameter infra-red markers (Figure 6.13c). The markers are placed on the towel to match the initial positions of the mass nodes in the model. For example, if a 5×5 node mass spring model is used, 25 markers will placed on the real object at same locations as the point masses in the model. The four camera configuration used for the experiment is shown in Figure 6.13a, and the towel, fitted with 20 markers in a 5×4 rectangular configuration, is shown in Figure 6.13b. The towel, along with the markers, have a combined mass of 0.34 kg, with dimensions of $0.51\text{m} \times 0.8\text{m}$. Each marker is placed 0.17m away from an adjacent marker in the horizontal direction and 0.2m away from an adjacent marker in the vertical direction. Here the horizontal and vertical directions are with respect to the view of the cameras. The OptiTrack system captures the position and orientation of each marker at a rate of 100 times per second, implying new data is available



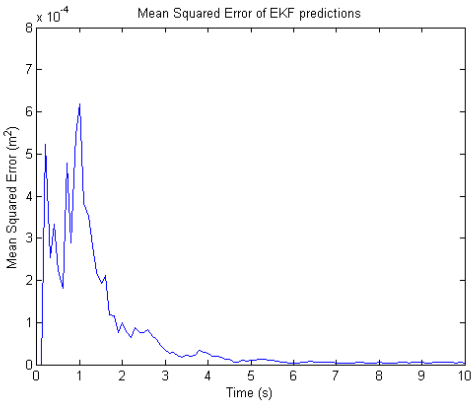
(a) EKF MSE with $Q_k = \mathbf{I}$ and $R_k = \mathbf{I}$



(b) EKF MSE with $Q_k = 10\mathbf{I}$ and $R_k = \mathbf{I}$

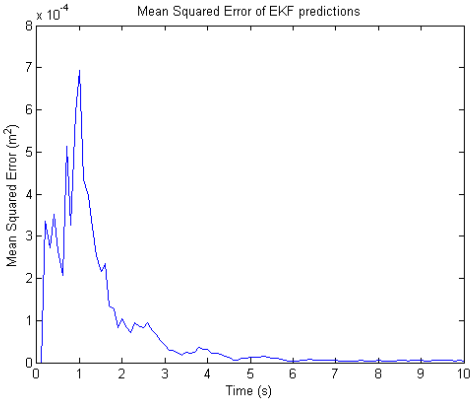


(c) EKF MSE with $Q_k = 100\mathbf{I}$ and $R_k = \mathbf{I}$

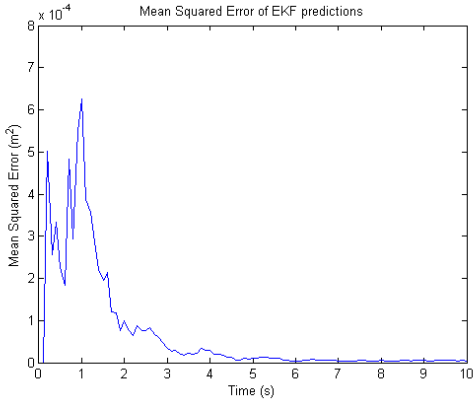


(d) EKF MSE with $Q_k = 1000\mathbf{I}$ and $R_k = \mathbf{I}$

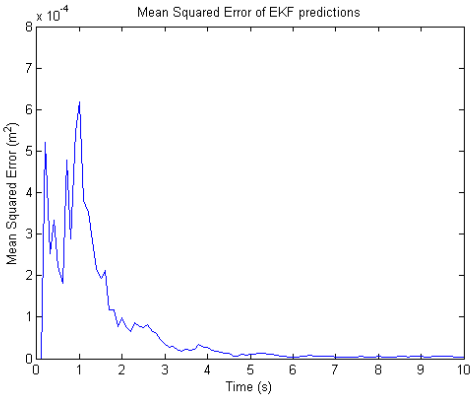
Figure 6.11: EKF MSE prediction mean squared error for varying Q_k



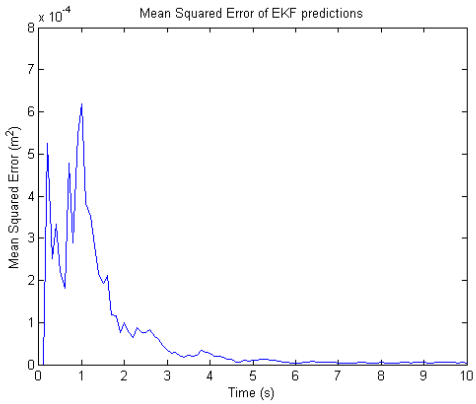
(a) EKF MSE with $Q_k = 100\mathbf{I}$ and $R_k = 10\mathbf{I}$



(b) EKF MSE with $Q_k = 100\mathbf{I}$ and $R_k = \mathbf{I}$



(c) EKF MSE with $Q_k = 100\mathbf{I}$ and $R_k = 0.1\mathbf{I}$



(d) EKF MSE with $Q_k = 100\mathbf{I}$ and $R_k = 0.01\mathbf{I}$

Figure 6.12: EKF MSE prediction mean squared error for varying R_k

every 0.01 seconds. This allows the EKF prediction algorithm to be run every 0.1 seconds, just as in the simulation scenarios in Section 6.2. The data collected from the Optitrack system needs to be post-processed because the structure of the data is not suited for the EKF algorithm. As a result, the EKF is run offline, post data collection. However, with more development time, a real time algorithm could be developed in conjunction with a different sensor system.

During the data collection phase, the cloth is initially placed approximately 2.2m away from the calibrated cameras, so that all markers are viewable by all four cameras. The cloth is anchored along the top row of markers, just as in the simulation scenarios in Section 3.3, by a custom made fixture shown in Figure 6.13d. A small fan is placed behind to the cloth to provide external forces to the cloth. The fan rotates sideways, producing oscillatory forces on the cloth. The cloth begins in a resting state, and when the fan is turned on, the cloth movement is carefully observed so that self-occlusions do not occur. In other words, all markers must be viewable at all times in at least two cameras. This allows for a constant stream of 20 marker positions to be available throughout experiment. Data is collected for 10 seconds, and is stored in a comma separated value file formatted by the OptiTrack software. This file contains time stamp of each frame, the position of each marker in (x, y, z), the orientation of each marker (roll, pitch, yaw, as well as quaternions), as well as additional information about the cameras. Using Matlab, the data is parsed such that only time and positional data are available to the EKF algorithm.

For the EKF formulation, the initial state estimate $\hat{x}_{0|0}$ is set to the initial configuration of the towel with the velocity states set to zero. In other words,

$$\hat{x}_{0|0} = \begin{bmatrix} y(0) \\ \mathbf{0}_{3n\times 1} \end{bmatrix}. \tag{6.15}$$

The initial state covariance matrix $P_{0|0}$ is set to the zero matrix, $\mathbf{0}_{6n\times 6n}$, since the cloth model begins at the same position as the actual towel (the first frame of recorded data). The mass parameters for the model are chosen by evenly distributing the mass of the towel and markers over all 20 nodes. This results in $m_i = 0.017$kg for all $i = 1 \dots 20$. The spring and damper parameters were chosen arbitrarily for this experiment to be $k_{spring_j} = 300$N/m and $k_{damper_j} = 0.08\text{N} \cdot \text{s/m}$ for all $j = 1 \dots 77$; the same as values chosen for the simulation scenarios in Section 6.2. Here, the number of spring connections are solved using Equation (5.5). The arbitrary choice of spring and damper parameters may induce additional modelling error into the system; however, this will be compensated by the choice of process noise covariance matrix Q_k. For the initial implementation of the EKF to the

(a) Four camera configuration

(b) Towel with 20 infra-red markers

(c) 12.7mm infra-red marker

(d) Cloth hanging fixture

Figure 6.13: Experimental data collection equipment

real cloth, the covariance matrices were chosen be:

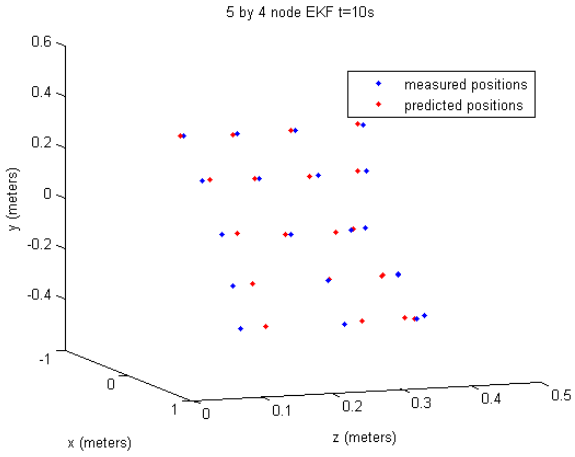
$$Q_k = 100\mathbf{I}$$
$$R_k = \mathbf{I}.$$

The noise covariance matrices chosen for this configuration are the same as the ones tuned for the viscous force simulation in Section 6.3. Due to the difficulty of modelling the applied forces in the experiment, the movement in the cloth is assumed to be attributed to random viscous-like forces. As a result, the covariance matrices are chosen to be the same as those tuned for the random viscous force simulation model. The results of the EKF, Figure 6.15, show there are large peaks in mean squared error periodically, nearly $7 \times 10^{-4}\text{m}^2$ approximately every 5 seconds. This obviously results from the oscillatory motion of the fan providing the external force. After the peaks in error, the EKF predictor nearly converges to the measured value, since there is less than $2.5 \times 10^{-5}\text{m}^2$ of error 7.5 seconds into the recursion. Figure 6.14 shows the positions of the predicted nodes overlaid onto the measured positions of the real data. The peak in error in Figure 6.15 occurs approximately 5 seconds into the EKF recursion, which can visually be seen in Figure 6.14c. Comparing the results at 0.1 seconds, 2 seconds, 5 seconds, and 10 seconds into the EKF recursion, it can be seen that the EKF predictions deviate the most from the measured data at 5 seconds, and the least at 2 seconds 6.14b; which is to be expected from results of the MSE plot. The predictions begin to deviate again at 10 seconds, Figure 6.14d, as the external force increases. These results show that even though the external forces are not modelled in the EKF, the EKF still produces near convergent predictions after the external forces are applied.

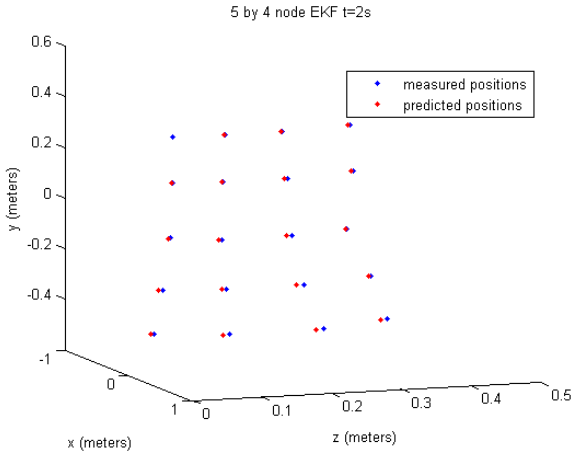
Instead of arbitrarily choosing the spring constants, k_{s_i}, there are a number of spring parameter identification techniques that allow the mass spring model to more closely resemble its real-life counterpart. As discussed in [46], these techniques include data driven parameter identification and analytic parameter identification from isotropic linear elastic reference models. Data driven parameter identification uses nonlinear optimization techniques to find parameter sets that minimize some form of error between the simulation model and real cloth data. As described in [46], the objective function

$$G(\theta) = \sum_i \|\mathbf{p}_i^{ref} - \mathbf{p}_i(\theta, \mathbf{f}^{ref})\|_2^2, \tag{6.16}$$

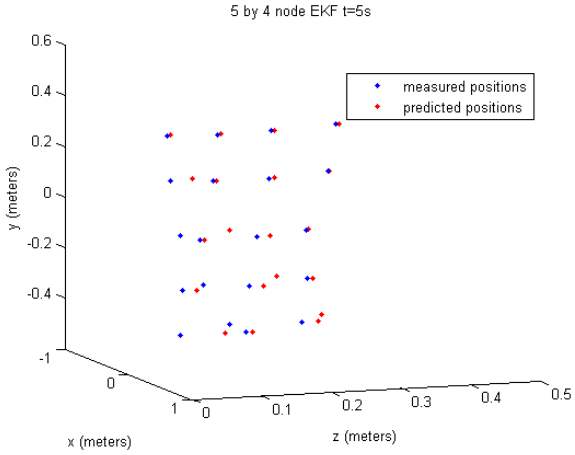
where $\mathbf{p}_i(\theta, \mathbf{f}^{ref})$ are the equilibrium positions of the simulated cloth for a specific spring parameter vector θ and given external force $\mathbf{f}^{ref}$, and $\mathbf{p}^{ref}$ are the vertex positions of a reference model, must be minimized to obtain the optimal parameter vector $\hat{\theta}$. Heuristic



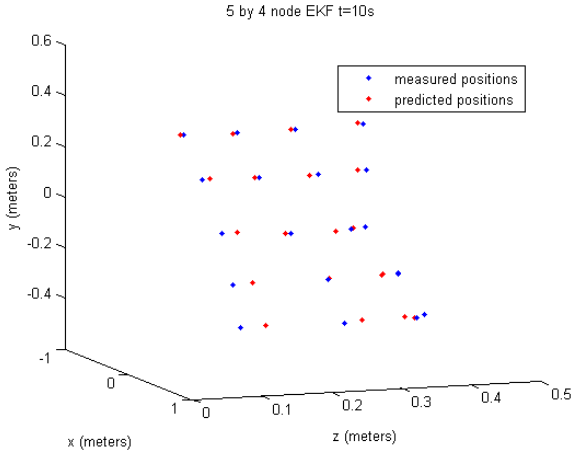
(a) Cloth position at 0.1s



(b) Cloth position at 2s



(c) Cloth position at 5s



(d) Cloth position at 10s

Figure 6.14: EKF applied to 5×4 node experimental data

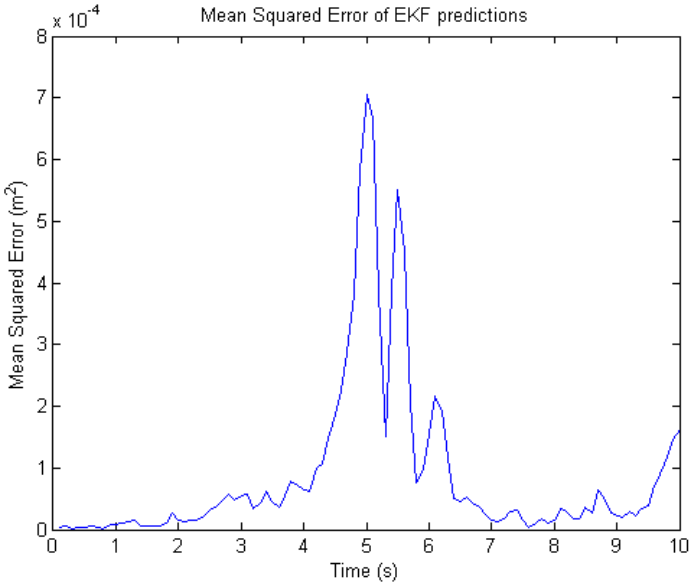


Figure 6.15: Mean squared error between EKF predictions and measured cloth positions over time

based optimization techniques such as simulated annealing and genetic algorithms can be used to find a global minimizer for $G(\theta)$. Data driven parameter identification can give near optimal results for lower order systems relatively quickly, however, as the number of nodes increase, these techniques become computationally expensive. Additionally, the solution is very sensitive to the choice of reference deformations, due to the nonlinearity of the mass spring system. Further, the reference deformations themselves may be difficult to model when being applied as external forces to the mass spring system. Lloyd [46] proposes using analytical expressions for identifying the system parameters (only spring constants) for mass spring cloth systems by equating the finite element model equations of the cloth, to a mass spring model, linearized about the same equilibrium point as FEM model. The analytic solutions use the elastic properties of the material being analyzed. Specifically, the Poisson ratio and the Young's modulus of the material is used. Poisson's ratio is the ratio between the change in expansion and the change in compression for small changes in a material. For a material stretched along the axial direction, Poisson's ratio is defined as

$$\nu = -\frac{d\varepsilon_{trans}}{d\varepsilon_{axial}}, \tag{6.17}$$

where ε_{trans} is the strain in the transverse direction (negative for stretching) and ε_{axial} is the strain in the axial direction (positive for stretching). For most materials, the Poisson's ratio is between 0 and 0.5. Young's modulus is the relationship between the stress and strain in a material. More specifically,

$$E = \frac{FL_0}{A_0 \Delta L}, \tag{6.18}$$

where F is the force exerted on the object, A_0 is the cross-sectional area through which the force is applied, ΔL is the amount by which the length of the object changes, and L_0 is the length of the object.

There only exists an exact solution to this problem for two dimensional equilateral triangle mesh formulations at a Poisson ratio of $\frac{1}{3}$. For the more general case of 3 dimensional tetrahedral based meshes an approximation is made. The most optimal solution to the problem, in a least squares sense, that is the squared difference between the elements of the stiffness matrices for the FEM model and the mass spring model, is

$$k_{(i,j)} = \sum_{\mathcal{A}} \frac{2\sqrt{2}}{25} lE. \tag{6.19}$$

Here l is the Euclidean length between nodes i and j and the Poisson ratio is assumed to be $\frac{1}{4}$. Lloyd [46] states that this is the only Poisson's ratio at which a mass spring model can approximate the reference FEM model in an optimal way.

Assuming that the Poisson's ratio for the real cloth is $\frac{1}{4}$, and using a Young's modulus between 7GPa and 30Gpa, which is similar to the value for cotton fabric, the spring constants $k_{(i,j)}$ are calculated and implemented in the EKF. As [46] does not propose optimal values for damper constants nor node masses, they are again chosen to be $k_{damper_{(i,j)}} = 0.08\text{N} \cdot \text{s/m}$ and $m_i = 0.017\text{kg}$, respectively. Using these derived values for the spring constants, the EKF algorithm did not produce successful results. The large spring constant values, resulting from the very large Young's modulus, caused the state covariance matrix to be close to singular, as evidenced by the large condition number

$$\kappa(P) = \frac{|\lambda_{\max}(P)|}{|\lambda_{\min}(P)|}, \tag{6.20}$$

shown in Figure 6.16. The condition number is consistently greater than 1×10^{20} for all iterations of the EKF. Large spring values cause the system to be numerically unstable, and as a result, the analytic solution does not produce a useful outcome. If the elastic

properties of the material were chosen more carefully, or if the solver was more precise, the analytic solution may have produced better results. This will be the subject of future research.

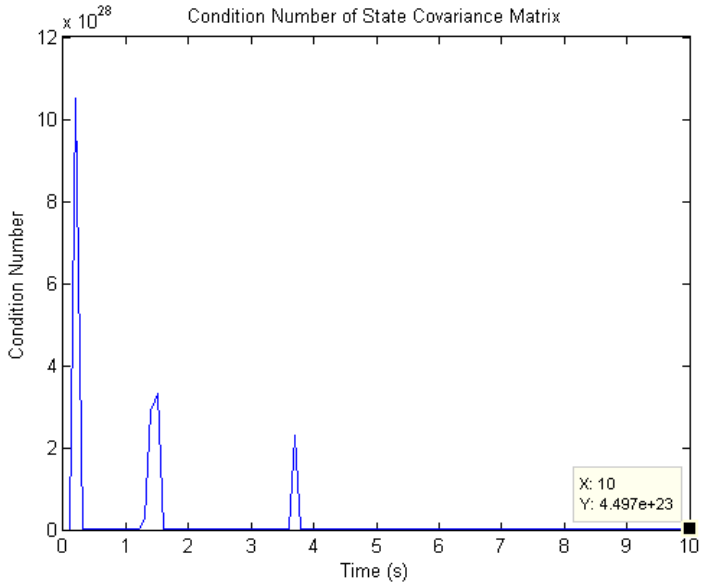


Figure 6.16: Condition number of state covariance matrix $P_{k|k}$ during EKF recursion

Since the analytic parameter ID did not return useful results, a simulated annealing data driven parameter identification technique is used on a simpler scenario to determine the best model parameter for EKF implementation. The scenario used for the parameter identification is dropping the same 5×4 node towel at a slight angle, with gravity being the only external force. This removes the need to model additional forces in the simulation model. Using the same approach as the model compression from Chapter 5, a parameter vector is found that best matches the simulation model to the real world measurements. For this case, the simulated annealing algorithm finds a solution that solves the minimization problem:

$$\begin{aligned} \min_{\theta} \quad & \int_0^t e^{\alpha\tau} \|Hz(\tau) - Hx(\tau)\|_2^2 d\tau \\ \text{s.t.} \quad & \dot{z}(t) = f(z, \theta), \end{aligned} \tag{6.21}$$

where $x(t)$ is the time series of measured states, $z(t)$ is the time series of simulated states, H is the observation model, θ is the parameter vector and $\alpha = 0.001$. The final parameter vector θ_N is applied to the EKF model to predict the motion of the original real world

scenario. This is the scenario where the towel is perturbed by a wind force from a portable fan. As can be seen by the mean squared error in Figure 6.17, the new prediction error is much lower than the prediction error in Figure 6.15. The peak error has decreased by nearly a factor of 4, while the transients near the 6 second mark have been significantly reduced. Once again, if more cloth scenarios are implemented into the cost function of the simulated annealing algorithm, a more general parameter set may be found. Using an expanded cost function will essentially train the EKF to run well on any scenario the specific object, in this case the towel, is subject to.

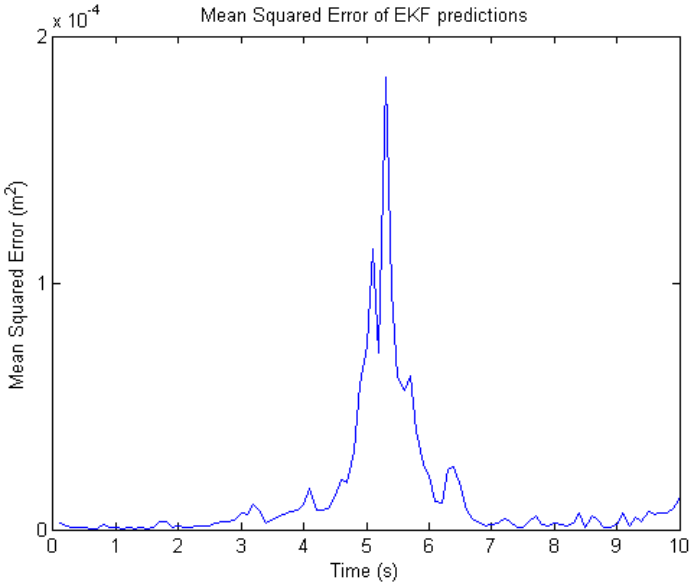


Figure 6.17: Mean squared error between EKF predictions and measured cloth positions over time with new parameters

6.5 Projector Compensation

For spatial augmented reality applications, where deformable objects are required to be projected upon, it is imperative to consider the speed at which the object's surface is moving. Since video projectors draw images at specified rates, surfaces that move noticeably and quickly, with respect to the drawing rate of the projector, may incur additional image distortion when the frames have finished being drawn. To compensate for the effects

of surface movement during the drawing of frames, an inter-frame prediction method is proposed.

Video projectors generally draw at rates between 24 frames per second (fps) and 60fps, as specified by the video file format; however, newer video encodings allow for frame rates up to 300fps. Within each frame, projectors draw images based on the specified display resolution and the source file's scanning format. The display resolution of a projector, or any display monitor, is given by its pixel dimensions. The pixel dimensions of a display are simply the physical number of columns and rows of pixels which create the display. For example a 1920×1080 display would have 1080 rows of 1920 pixels each. Modern, high definition projectors generally have resolutions ranging between 1280×720 and 1920×1080 pixels. However, ultra high definition displays can have much larger pixel counts. The scanning format of a video file describes how a video is "painted" on a display. There are two common methods for scanning video images: progressive scanning and interlaced scanning. Progressive scanning is method of displaying images in which each row of pixels is drawn in sequence. This is in contrast to interlaced scanning where every other row of pixels is drawn, in an alternating fashion. More specifically, the interlaced scanning method reads a signal containing two "fields" of data, odd-numbered rows and even-numbered rows, captured at two different times, and alternates drawing each field during each frame. This allows the display to appear to be running at twice its normal frame rate. Progressive scanning is far more common in modern displays, as it allows motion to appear smoother and is free of the visual artifacts which can be caused by interlacing. However, broadcasting still distributes interlaced signals, as progressive signals require twice the bandwidth. Concatenating the number of horizontal lines in a display with the letter associated with the scanning type (p for progressive and i for interlaced) is the standard naming convention for describing the resolution and scanning of a display. For example, a 1920×1080 high definition display with progressive scanning is labelled as a 1080p display (1080 horizontal progressively scanned lines).

For a 1080p display drawing images at the standard frame rate of 24fps, an entire frame is drawn every $\frac{1}{24}$s. Now, since each of the 1080 horizontal lines are drawn sequentially, due to progressive scanning, each line is drawn at a rate of 25.9kHz. For a 5×5 node mass spring model, with evenly spaced nodes, 270 rows of pixels are projected between each horizontal row of nodes. As a result, each horizontal row of nodes is drawn $\frac{270}{25900} = 0.0104$ seconds after the previous row of nodes. Here, horizontal rows of nodes refer to rows that are horizontal from the point of view of the projector. Considering the sampling rate of the EKF is T_s, if the cloth's position changes significantly during inter-sample periods, there may significant error between the prediction and the position of the cloth at the Kalman update step. To compensate for this, an interpolation approach is used. As the

cloth is moving, the EKF solves for an estimate of the velocity states, and using an Euler approximation, the inter-sample position of every node is calculated. This estimation is based on the assumption that drawing horizontally is instantaneous, as each pixel in a row of 1920 pixels is drawn in $\frac{1}{25900}$ seconds.

Using the previous state estimate $\hat{x}_{k|k}$ and the corresponding time step, $m\Delta T$, where m is the row number, and the time step ΔT is calculated based on the frame rate and the number of rows of nodes, assuming the top row of nodes is the top of the display, in the following way:

$$\Delta T = \frac{1}{\text{frame rate} \times (\#\text{rows} - 1)}, \tag{6.22}$$

the inter frame prediction can be computed. First the state estimate vector is split into a position estimate vector $\hat{p}_{k|k}$ and a velocity estimate vector $\hat{v}_{k|k}$. The position estimates are then reordered, such that the elements are ordered based on their horizontal position with respect to the projector. More specifically, the first i elements of the position vector would contain the positional information about the first horizontal row of nodes with respect to the projector, the next j elements would contain the positional information about the second horizontal row of nodes with respect to the projector, and so on (Figure 6.18).

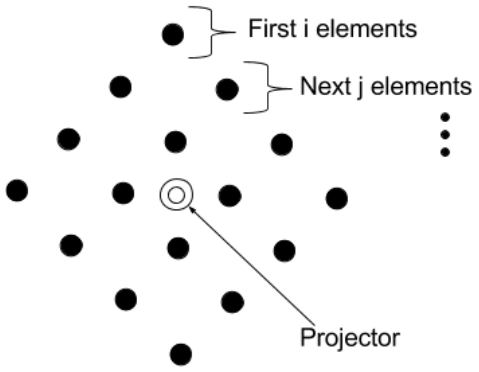


Figure 6.18: Orientation of cloth with respect to projector for inter-frame prediction

After reordering the position estimates, the state estimates are passed through the state transition function $f(\cdot)$. This returns the derivative of the state estimates, and as a result, the velocities to obtain the next position vector. This velocity vector is also reordered, in

the same way the position estimates are ordered, so that each node's position aligns with its velocity. The velocity vector is then multiplied by a $3n \times 3n$ matrix defining the time at which each row of of the object is predicted. The result is added to the position estimates to obtain the inter-frame position predictions. This process is shown in Equation (6.23).

$$\begin{bmatrix} \hat{p}'_{k|k(1)} \\ \vdots \\ \hat{p}'_{k|k(i)} \\ \hat{p}'_{k|k(i+1)} \\ \vdots \\ \hat{p}'_{k|k(i+j)} \\ \vdots \\ \hat{p}'_{k|k(n-l)} \\ \vdots \\ \hat{p}'_{k|k(n)} \end{bmatrix} = \begin{bmatrix} \hat{p}_{k|k(1)} \\ \vdots \\ \hat{p}_{k|k(i)} \\ \hat{p}_{k|k(i+1)} \\ \vdots \\ \hat{p}_{k|k(i+j)} \\ \vdots \\ \hat{p}_{k|k(n-l)} \\ \vdots \\ \hat{p}_{k|k(n)} \end{bmatrix} + \begin{bmatrix} 0\Delta T \times \mathbf{I}_{3i\times 3i} & & & \\ & 1\Delta T \times \mathbf{I}_{3j\times 3j} & & \\ & & \ddots & \\ & & & (n-1)\Delta T \times \mathbf{I}_{3l\times 3l} \end{bmatrix} \cdot \begin{bmatrix} \mathbf{I}_{3n\times 3n} \\ \mathbf{0}_{3n\times 3n} \end{bmatrix}^T f(\hat{x}_{k|k}) \quad (6.23)$$

Figure 6.19 shows the results of an inter-frame prediction where a projector is projecting at a frame rate of 24fps on a 5×5 node cloth surface. The inter-frame prediction is compared to the the previous state estimate, the next EKF prediction and the next measured output. The plot shows the position of each row of nodes as a line. The top lines represents the position of nodes at the beginning start of drawing the frame, and the bottom line represents the last row of nodes at the end of drawing the frame. The inter-frame prediction is at the same position as the previous state estimate for the first row of nodes, and progressively gets closer to the next prediction and measurement at each successive row. Since the frame rate of the projection is 24fps, the last row of cloth is predicted 0.04 seconds in the future using the inter-frame prediction. This is faster than the EKF sampling time of 0.1 seconds, implying that the EKF prediction would not have been able to compensate for any perturbations.

In the scenario where the frame rate is chosen to be $\frac{1}{T_s}$, where T_s is the sampling time of the EKF, the last row of the inter-frame prediction will match the position of the next EKF prediction. If the frame rate of the projection is chosen to be 10fps, possibly due to computational power, it ensures the drawing time of a single frame to be 0.1 seconds,

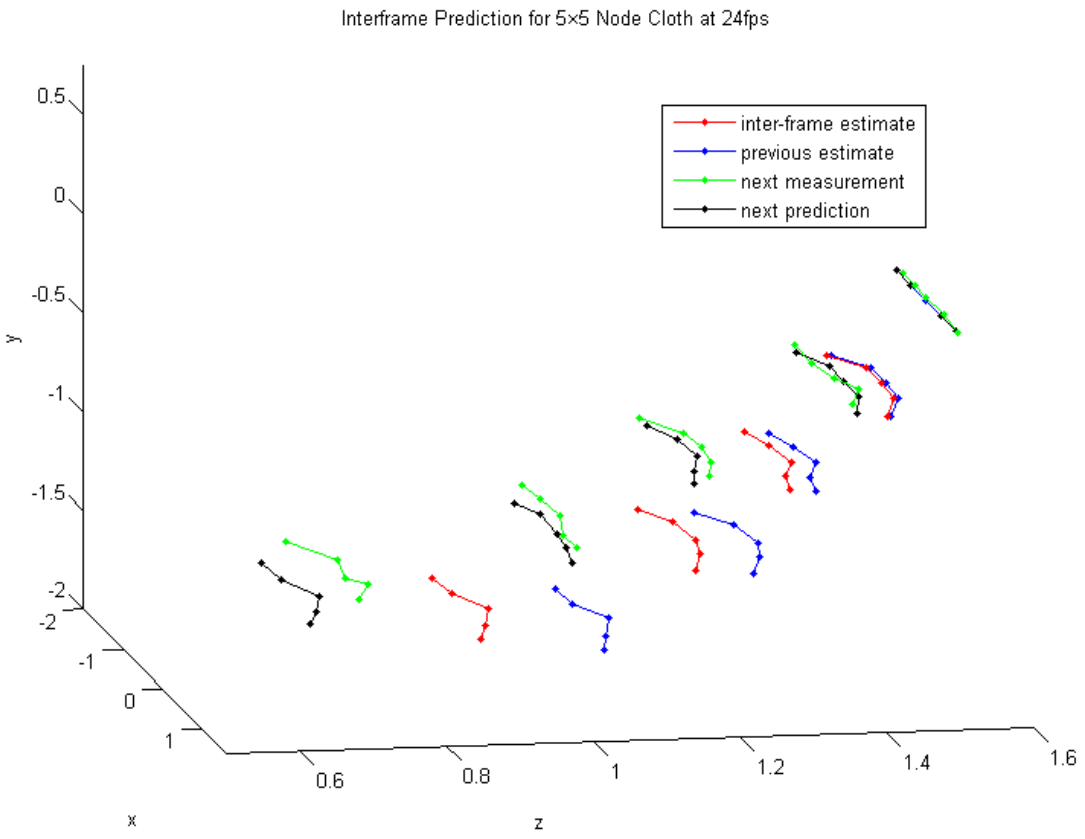


Figure 6.19: Inter-frame prediction of 5×5 node cloth at a projection frame rate of 24fps.

the same as the sampling rate of the EKF from the previous example. Figure 6.20 shows the results of inter-frame prediction when the frame rate is set to 10fps. It can be seen that the position of the first horizontal line of the inter-frame prediction is at the same position as the previous estimate. This is due to the anchoring of nodes in the model. However, even if it was not anchored, it would match the previous estimate due to $\hat{p}'_{k|k(1)} = \hat{p}_{k|k(1)} + 0\Delta T \times \hat{v}_{k|k(1)}$. Also, due to the frame rate of the projector, the last horizontal row of the inter-frame prediction is at the same position of the final row of the next EKF prediction. With this frame rate chosen, the final row equation of the inter-frame prediction (from Equation (6.23)) is exactly the EKF equations for the state prediction.

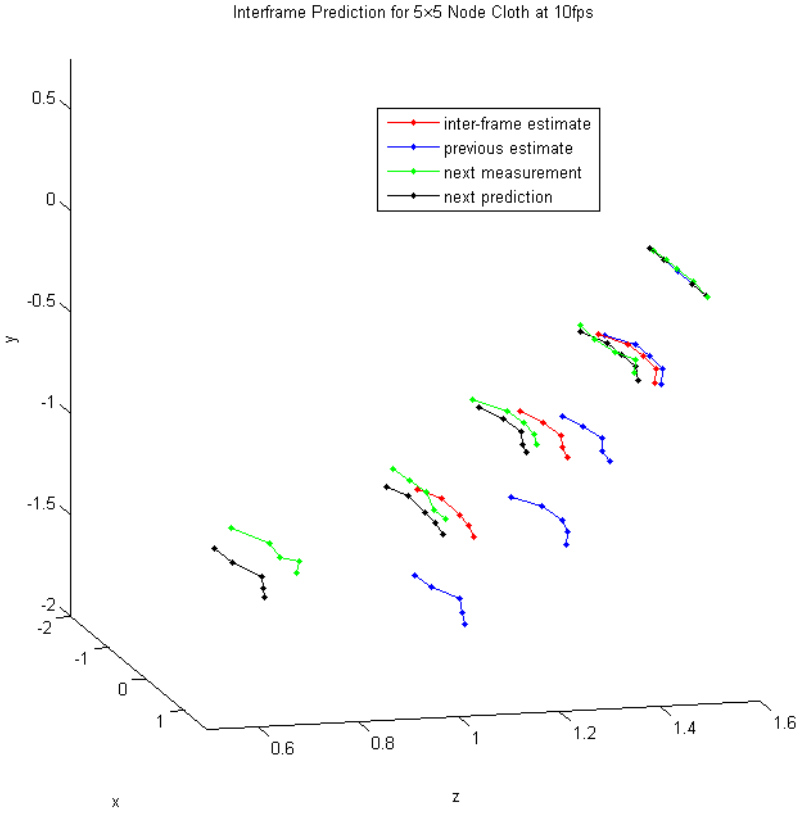


Figure 6.20: Inter-frame prediction of 5×5 node cloth at a projection frame rate of 10fps

To validate the inter-frame prediction algorithm, the algorithm is applied to the experimental data from Section 6.4. The results for the algorithm at a single instance of time, t_0, are shown in Figure 6.21. The results assume that the projector is running at a frame rate

of 10fps. Since the data is collected at a rate of 100Hz by the Optitrack motion capture system, the accuracy of the inter-frame prediction can be verified. The red lines in Figure 6.21 represent the measured positions of each row of nodes, of the 5×4 configuration, at the time they are projected upon. The top row of nodes are shown at $t = t_0$, the second row of nodes are shown at $t = t_0 + 0.025$, and so on. The times are rounded to the nearest hundredth of a second as to match the time-steps from the measured data. The green lines show the position of each row of nodes from the previous measurement, the starting time of the projection. This is what the surface would look like if no prediction was run. Lastly, the inter-frame prediction of each row of nodes is shown in blue. It can easily be seen that the position of the nodes further down the cloth are better predicted using the inter-frame prediction than using just the outputs at the time the projection begins. This result shows that using inter-frame prediction can reduce image distortion during projection, as it better matches the true position of the surface compared to just using measurement data.

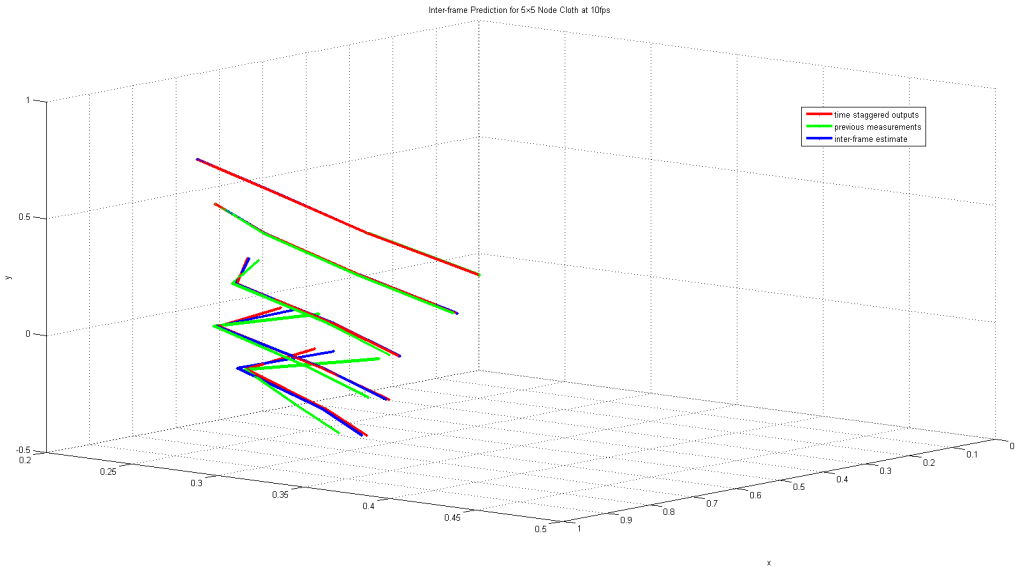


Figure 6.21: Inter-frame prediction of experimental data

Chapter 7

Conclusions and Future Works

The purpose of this thesis was to validate the use of the extended Kalman filter (EKF) for predicting the position of time-varying surfaces. More specifically, by combining a well known model for simulating deformable bodies with the EKF, a real-time prediction algorithm was created. Further, for surface movement during image projection, an additional prediction scheme was developed to minimize error at the the time of projection.

In Section 3.1, the mass spring model was was explained and derived. Interconnected mass-spring-damper systems are used to reproduce the dynamics of deformable bodies, and can be represented using a nonlinear state space model. The dynamics of the system are produced by forces from springs and dampers, along with any external forces. The model is verified by simulating with two integration schemes. The first scheme is an explicit integration algorithm, the Runge-Kutta method. The Runge-Kutta method, as with all explicit methods, uses 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, making it computationally demanding. The second scheme is an implicit integration scheme, the backwards Euler method. This method uses both the previous and the next data point to compute the next output, making it inherently stable. As a result, larger integration step times can be used, making it applicable for real-time applications. It is less accurate than the Runge-Kutta algorithm when the integration step-time is larger than the Runge-Kutta's integration step-time. However, they produce equivalent results when the same step-time is chosen (assuming the Runge-Kutta is numerically stable).

Since the efficiency of the EKF algorithm is inversely proportional to the number of states in the model, reducing the number of states allows the algorithm to be run faster,

and as a result, in real-time. In Chapter 5, state compression is completed by identifying parameters that minimize the difference in behaviour between the original dense model and a desired sparse model. As the system does not seem to have a unique minimizing parameter set, a heuristic based minimization algorithm is used to find a solution. Solution sets were found for compressing a 21×21 model to both a 11×11 and a 5×5 model. Generally, when fewer states are used in the compressed model, the results are less accurate. However, by introducing a new cost function for simulated annealing, and additional parameters, the behaviour of sparse models can be improved. When searching for the optimal parameter vector, the simulated annealing algorithm was first run on only one scenario. This resulted in the overfitting of data. Instead, another cost function which finds a parameter vector that minimizes the error between multiple scenarios is used. Although this produced slightly worse results when applied to the original scenario, it did produce a more general parameter set. By including many scenarios into the cost function, it is likely that a general solution can be found. The results of this section show that dense models can in fact be compressed, with little loss in behaviour, so that prediction can be run in real-time.

In Chapter 6, the EKF is formulated using the linearization of the mass spring model. The model is linearized by finding the Jacobian at a specific operating point, and is used as the state transition matrix in the EKF algorithm. The EKF produced convergent results in two simulation scenarios: one with added process noise and the other with external random viscous forces. These perturbations were assumed to be modelling errors in the system, and were compensated for by tuning the noise covariance matrices Q_k and R_k. To validate the effectiveness of the EKF, experimental data was collected and tested upon. Using an Optitrack motion capture system, the position of discrete nodes on a towel being perturbed was recorded. The EKF was able to predict the position of nodes 0.1 seconds into the future with convergent results. An obvious next step for this algorithm is real-time implementation. The EKF ran on the recorded data fast enough to show real-time implementation is possible; however, the data collection method needs improvement for the system to be usable. Optimal choices for model parameters (spring constants, damper constants, masses) were also discussed in this thesis. Finding more precise values of model parameters would likely improve the accuracy of the predictions. Using previously studied analytic values for spring parameters did not produce useful results. As a result, a data driven system ID approach was used. By using a simulated annealing algorithm, a new parameter set was found that reduced the prediction error by a factor of 4. An additional prediction method was also developed to compensate for motion during the projector's drawing phase. The position of each node is estimated at their specific time of drawing so that image distortion can be minimized. The results show that using the inter-frame

prediction method provides an improvement over using just measurement data, as it more closely approximates the position of the surface at the time of image projection.

While the results of using the mass spring model are visually appealing, it does lack in physical accuracy when compared to newer methods. Many studies have improved the model by using nonlinear parameters or introducing material properties to the model [28] [26]. Future work includes using these modified mass spring systems as the base physical model for prediction. Furthermore, if more computational power is available, the model can be abandoned entirely for a more physically accurate model. With technological advancement in computational power, models for deformable body motion, such as Eulerian models, can be used in real-time. This will allow for the most accurate predictions that fully encapsulate the properties of the physical object.

Since a heuristic approach was used to find the minimizing parameter set in Chapter 5, there is no guarantee that the solution is the global minimum. Furthermore, the solutions are specific to the simulation scenarios they were solved for. This means that for every situation, a new parameter set has to be found. This is computationally expensive, and as a result, not advised. Using multiple scenarios in the cost function was studied, providing more general results; however, this requires a great amount of preprocessing. Current research looks at how to use system identification techniques to find optimal mass spring parameters. Techniques such as approximate analytic expressions can be used to find minimizing parameters quickly and without any preprocessing. Future work, therefore, involves applying these techniques to model compression, in real-time, to determine the optimal parameters for EKF implementation.

In Chapter 6, a marker based motion capture system is used to measure position data of surfaces. Marker-based motion capture systems are highly accurate, but are expensive and very prone to occlusions. Camera based systems combined with computer vision techniques, such as feature selection, can be used instead to capture the position of surfaces in real-time. Future work includes using the Microsoft Kinect, for example, to measure the position of a deformable surface. Although this will likely cause an increase in sensor noise in the system, the estimation filter should be able to compensate. Further, if the accuracy of the EKF is not high enough, other estimation filter algorithms could be applied to improve the accuracy of predictions. The unscented Kamlan filter can be explored as it solves for a higher order approximation of the nonlinearities in the model with the same computational complexity. Particle filters can also be implemented as they may improve prediction accuracy.

Immediate future work would be to implement this in a real-time system to determine if, in fact, the inter-frame prediction does improve the user experience. Further, the first

real-time implementation of this algorithm will be to the surgical simulator in [50]. This prediction scheme will allow for a more immersive experience for surgeons while they train for pedicle screw insertion surgery.

References

[1] Analogous electrical and mechanical systems.

[2] Checkerboard background patterns.

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

[4] Jennifer lopez american idol. http://akkidokie.com/blog/wp-content/uploads/2015/03/JLos-epic-Projector-Dress_akkidokie-1.jpg/. Accessed: 2016-08-05.

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

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

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

[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] 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.

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

[12] 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.

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

[14] 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.

[15] 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.

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

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

[18] 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.

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

[20] 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.

[21] 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.

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

[23] 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.

[24] 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.

[25] 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.

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

[27] 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.

[28] 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.

[29] 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.

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

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

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

[33] 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.

[34] 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.

[35] Andrew J. Heunis. *ECE 686 Course Notes*. University of Waterloo, 1 edition.

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

[37] 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.

[38] 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.

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

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

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

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

[43] 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.

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

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

[46] 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.

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

[48] 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.

[49] 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.

[50] 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.

[51] 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.

[52] 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.

[53] 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.

[54] 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.

[55] 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.

[56] 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.

[57] 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.

[58] 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.

[59] 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.

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

[61] 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.

[62] 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.

[63] 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.

[64] 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.

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

APPENDICES

Appendix A

Sparsity Result for Rectangular Mass Spring Model Jacobian

To determine the sparsity of the Jacobian as a function of the total number of nodes $n = n_1 \times n_2$, the total number of non-zero elements must be found. The number of spring and damper connections here is s, where s is solved for by using Equation (5.5). Trivially, the upper left $3n \times 3n$ matrix has no non-zero elements, as it is always $\mathbf{0}_{3n\times 3n}$. The upper right $3n \times 3n$ matrix contains $3n$ non-zero elements as it is always $\mathbf{I}_{3n\times 3n}$. The lower right $3n \times 3n$ matrix $\mathbf{J}_{av}$ can always be written as

$$\mathbf{J}_{av} = \mathbf{U}_{av} + \mathbf{V}_{av}, \tag{A.1}$$

by solving Equations (3.14) and (3.15). Here $\mathbf{U}_{av}$ is a negative definite, diagonal, $3n \times 3n$ matrix, and $\mathbf{V}_{av}$ is a $3n \times 3n$ matrix containing $2s$ (due to symmetry) negative definite, diagonal, 3×3 sub-matrices. Therefore, $\mathbf{J}_{av}$ will always have $3n + 6s$ non-zero elements. The total number of non-zero elements contained lower left $3n \times 3n$ matrix, $\mathbf{J}_{ap}$, cannot be analytically determined. Instead, an upper bounds of the number of non-zero elements will be calculated instead. Similar to $\mathbf{J}_{av}$, $\mathbf{J}_{ap}$ can be written as the sum of two matrices

$$\mathbf{J}_{ap} = \mathbf{U}_{ap} + \mathbf{V}_{ap}. \tag{A.2}$$

Here, $\mathbf{U}_{ap}$ is a $3n\times 3n$ block-diagonal matrix containing only the values found from Equation (3.20). The maximum number of non-zero elements in $\mathbf{U}_{ap}$ is therefore $9n$. Similar to $\mathbf{V}_{av}$, $\mathbf{V}_{ap}$ contains $2s$ 3×3 matrices, and as a result, will have a a maximum of $18s$ non-zero elements. Altogether, $\mathbf{J}_{ap}$ can have a maximum of $9n + 18s$ non-zero elements. Therefore, the entire Jacobian can have a maximum of $15n + 24s$ non-zero elements. As n_1 and n_2

increase in magnitude, the number of spring connections can be approximated by

$$s \approx 6n. \tag{A.3}$$

As a result, the sparsity of the Jacobian, $S(n)$, for large values of n_1 and n_2 is

$$S(n) = \frac{15n + 24s}{36n^2} \approx \frac{53}{12} \cdot \frac{1}{n}. \tag{A.4}$$

For example, a 21×21 node cloth model would have a sparsity of approximately $S(441) \approx 0.01$, or 1% of its elements are non-zero. Due to the sparsity of this system, sparse matrix decomposition techniques, such as LU factorization [19], can be used to solve linear systems containing the Jacobian. For example, the implicit integration scheme in Section 3.3 can be implemented much more efficiently when using LU factorization on a highly sparse system.