NTNU – Trondheim
Norwegian University of
Science and Technology

Kalman Smoothing Techniques in Medical Image Segmentation

Sigurd Storve

Master of Science in Electronics
Submission date: June 2012
Supervisor: Ilangko Balasingham, IET
Co-supervisor: Hans Torp, ISB
Engin Dikici, ISB
Sten Roar Snare, GE Vingmed

Norwegian University of Science and Technology
Department of Electronics and Telecommunications

Problem description

At the department of Circulation and Medical Imaging, Kalman filter based approaches have been utilized for image tracking and segmentation. Until recently, the methods have been based on forward tracking using the Real Time Contour Tracking Library (RCTL). In particular for offline analysis, it would likely be advantageous to utilize image information from multiple images. In the literature Kalman smoothing techniques (such as fixed point, fixed interval and backward-forward smoothers) have been employed for similar purposes. To our knowledge, Kalman smoothers have not previously been applied to medical image processing problems.

During a summer job at the department of Circulation and Medical Imaging and later during his project work at NTNU the autumn 2011, Sigurd Storve was working on extending the current RCTL framework to also handle unscented Kalman filtering.

During his work with the master thesis, Sigurd will work to extend the RCTL framework to handle Kalman smoothing techniques. The work will be divided into three major parts:

- Implement the known methods: Fixed-lag smoothing, fixed point smoothing, and fixed interval smoothing.
- Implement a new novel model-based smoothing approach: smoothing around a learned model.
- Test the methods using datasets with known manual "ground truth" provided by a medical expert.

Oppsummering

Et eksisterende C++ - bibliotek for sanntidssegmentering av ultralyddata vha. Kalmanfiltrering, kjent som real-time contour tracking library (RCTL), brukes som utgangspunkt for implementering og evaluering av forskjellige teknikker for Kalman-smoothing: fixed-point, fixed-lag og fixed-interval Kalman-smoothing. I tillegg har en eksperimentell smoothing-teknikk, gitt navnet statistical smoother og basert på fusjon av segmenterte og lærte gjenomsnittsestimater ved forskjellige posisjoner i hjertesyklusen, blitt foreslått. Evalueringen gjøres med 29 volumetriske ultralydopptak av venstre ventrikkel, med fasit bestående av manuelle segmenteringer utført av en lege.

Den kliniske motivasjonen for dette arbeidet er forbedret nøyaktighet ved automatisk segmentering av venstre ventrikkel, som igjen kan brukes til å øke nøyaktigheten til automatisk måling av klinisk nyttige parametere som f.eks. ejeksjonsfraksjonen.

Evalueringen viser at ingen av de foreslåtte Kalman-smoothingteknikkene gir vesentlig bedre resultater enn vanlig Kalmanfiltrering. For de tre typene Kalman-smoothere blir dette argumentert å følge fra måten kantdeteksjoner gjøres på internt i biblioteket. Manglende ytelsesforbedring fra statistical smoother forklares med for store variasjoner fra person til person; gjennomsnittlig deformasjonsmønster til venstre ventrikkel kan ikke generaliseres til individuelle tilfeller i noen betydelig grad.

Abstract

An existing C++ library for efficient segmentation of ultrasound recordings by means of Kalman filtering, the real-time contour tracking library (RCTL), is used as a building block to implement and assess the performance of different Kalman smoothing techniques: fixed-point, fixed-lag, and fixed-interval smoothing. An experimental smoothing technique based on fusion of tracking results and learned mean state estimates at different positions in the heart-cycle is proposed. A set of 29 recordings with ground-truth left ventricle segmentations provided by a trained medical doctor is used for the performance evaluation.

The clinical motivation for this work is to improve the accuracy of automatic left-ventricle tracking, which in turn can be used to improve the automatic measurement of clinically important parameters such as the ejection fraction.

The evaluation shows that none of the smoothing techniques offer significant improvements over regular Kalman filtering. For the Kalman smoothing algorithms, it is argued to be a consequence of the way edge-detection measurements are performed internally in the library. The statistical smoother's lack of improvement is explained by too large interpersonal variations; the mean left-ventricular deformation pattern does not generalize well to individual cases.

Acknowledgements

I would like to thank my supervisors at ISB, Engin Dikici and Hans Torp for kind support, and my co-supervisors Sten-Roar Snare and Fredrik Orderud at GE Vingmed Ultrasound for good ideas and discussions. I would especially like to thank Sten Roar Snare for very useful feedback on the report manuscripts.

Abbreviations and terminology

- EKF : Extended Kalman filter.
- RCTL : Real-time contour tracking library.
- Covariance Intersection : algorithms for the fusion of two probabilistic estimates.
- LV : Left Ventricle.

- Ground truth : Triangle meshes representing the correct segmentation of the left ventricle.
- Statistical Smoother : The experimental Kalman smoothing algorithm based on learning from ground truths.
- The term *forward-backward smoothers* will be used to refer to both fixed-lag Kalman smoothers and fixed-interval Kalman smoothers implemented according to the forward-backward interpretation of smoothing.
- The words *contour* and *state* are in some places used interchangeably since an RCTL-state vector contains all parameters needed to fully describe the shape of a deformable model. All state vectors are interpreted relative to the left ventricle model used in this project.

Contents

1 Introduction

Between 1995 and 2006 the main cause of death in Norway was cardiovascular diseases [1]. The availability of fast and reliable techniques for measuring the heart function is therefore important both for emergency situations and routine examinations.

Medical ultrasound offers a cheap and safe way for imaging and evaluating the performance of the heart in real-time. The term echocardiography refers to ultrasound used to image the heart, and is usually focused on assessing the performance of the left ventricle, the main pumping chamber of the heart and responsible for maintaining blood flow through the blood vessels.

The current trend in echocardiography is three-dimensional imaging and quantitative analysis as opposed to qualitative, both of which are enabled by rapid technological progress. Much research effort is put into developing algorithms for extracting clinically useful parameters of the heart from ultrasound scans.

Almost all left-ventricular performance parameters are derived from either volume measurements or pressure measurements [2, p. 67]. The ejection fraction quantifies the proportion of the filled left ventricle ejected during a heart-cycle and is an important clinical parameter for predicting a patient's prognosis. It gives a measure of global heart function, i.e. how well the heart function as a single unit.

In two-dimensional ultrasound imaging, estimation of left-ventricular volumes necessarily involves approximations as the calculations are based on one or more two-dimensional slices of the ventricle. Several methods differing in complexity and accuracy are given in [3].

The task of locating objects and boundaries in images is in general termed *image segmentation*. In medical image segmentation, the objects of interest are typically organs or organ boundaries and the images are two- or three-dimensional. Three-dimensional ultrasound offers volumetric images from which volumes can be computed directly with great accuracy, given that the segmentation is accurate.

Hence, real-time estimates for performance measures derived from volume measurements, such as the ejection fraction, directly depends on the ability to perform real-time segmentation in an automated and robust way; an estimated delineation of the left ventricle throughout one full heart-cycle contains all necessary information.

However, automatic segmentation of ultrasound images is a challenging task, mostly because of the poor image quality caused by the inherent problem of

speckle noise. This physical phenomenon is responsible for the characteristic texture-like noise pattern of ultrasound that despite its appearance does not represent tissue structure. Speckle is discussed in more detail later, for now it suffices to point out that it complicates the process of automatic edge-detection, a key step in a large class of automatic segmentation algorithms.

1.1 Motivation

The clinical motivation for this work is to achieve more accurate automatic segmentation of ultrasound data in order to improve the quality of derived automatic measurements of clinical parameters. An example of such a parameter is the ejection fraction.

The technical motivation is to improve the real-time contour tracking library (RCTL) [4, 5, 6], which is an extended Kalman filter-based C++ library for efficient segmentation of arbitrary two-dimensional or three-dimensional recordings or video streams. The principle of operation is to fit a deformable shape template to the image data by encoding the deformation parameters as a state vector, which is subsequently updated with edge-detection results.

According to [6], the real-time contour tracking library is capable of 3d real-time left-ventricular segmentation at 25 frames per second on a 2.16 GHz Intel Core 2 duo processor at approximately 8 % CPU power.

One drawback of the Kalman filter is that it cannot make use of future measurements, even if available. This is the motivation for investigating Kalman smoothing techniques within the RCTL framework. The term *smoothing* refers to such estimates based on future measurements.

Future measurements are never available in a true real-time situation, but they might be if a short lag is permissible or if post-processing is performed. Although Kalman filtering techniques are widely used in the literature for performing medical image segmentation [7], for unknown reasons Kalman smoothing techniques is not as widespread.

The question addressed in this work is how different Kalman smoothing techniques can be applied to medical image segmentation within the RCTL framework and whether they can offer any improvement in tracking accuracy. The intuition is that an estimate based on more information will be more accurate.

1.2 This project

This work consists of implementation and evaluation of three classes of known Kalman smoothing techniques and one experimental smoothing algorithm. The

experimental smoother solution is termed the *statistical smoother* and is inspired by the information fusion performed in the Kalman smoothing algorithms, where forward and backward estimates are combined into a single estimate. The statistical smoother instead merges prior knowledge of shape deformation with forward filter estimates.

The performance evaluation will focus on left ventricle segmentation using deformable Doo-Sabin surfaces [6]. Except for the statistical smoother, all smoothing algorithms are usable with general deformable models as well.

The three classes of Kalman smoothing techniques are *fixed-lag smoothing*, *fixed-interval smoothing*, and *fixed-point smoothing*.

The usage of future measurement data necessarily implies a delay in the stream of output estimates. The fixed-lag Kalman smoother introduces a user-definable delay, while the fixed-interval Kalman smoother is limited to a post-processing tool because all measurement information must be available in order to compute any estimate. The principle of operation of the fixed-point smoother is to combine subsequent estimates at specific positions in the heart-cycle into better and better estimates, which is made possible by the very small changes from one heart-cycle to another. Neither the statistical smoother nor the fixed-point smoother introduce any lag.

The statistical smoother is based on learning from a set of ground-truths provided by a medical expert. Average state vectors with associated certainty estimates in the form of covariance matrices are computed at different points in the heart cycle. These can be viewed as estimates in their own right, and are fused with estimates from the extended Kalman filter, just as in the regular smoother algorithms.

As this work focus on the left ventricle of the heart, the assumption can be made that the data is periodic. All recordings used for evaluation purposes consists of a single heart-beat repeated to achieve a periodic extension, but recordings of multiple heartbeats could have been used as well.

Most of the algorithmic experimentation and implementation were performed with a Matlab (R2010b, The MathWorks, inc.) interface to the RCTL-library, called RctlMatlab. This approach was chosen since it is simpler to translate a well-performing smoothing algorithm from Matlab to C++ than it is to conduct all experiments in C++ directly. The performance assessment was also performed with Matlab.

1.3 Previous work

The starting point was an experimental implementation of a fixed-interval smoother inside the real-time contour tracking library itself. This functionality was not usable directly without moderate changes to the core library code. The difficulties arose from the fact that a fixed-interval smoother is not a filter, since no outputs can be produced before all inputs are known.

RCTL's experimental fixed-interval smoother consists of a forward-backward smoother implementation in addition to the covariance intersection algorithm which is introduced in section 2.7.1.

1.4 The structure of this thesis

In section 2 the necessary theoretical background will be presented, including medical ultrasound, Kalman filtering and smoothing, and the real-time contour tracking library. The implementation details follows in section 3, and the methods used to assess tracking performance are presented in section 4. In section 5 the results are presented and subsequently discussed in section 6. Finally, conclusions are presented in section 7. Additional technical details can be found in the appendices.

2 Theory

This section contains the necessary background material. First, a brief introduction to medical ultrasound is given, followed by a description of the human heart. Then the Kalman filter and an extension to non-linear systems, the extended Kalman filter, is introduced. The next section presents three common Kalman smoothing techniques as well as a numerical example exploring the performance gains introduced by smoothing.

A short introduction to subdivision surfaces, which is the type of surface used to model the left ventricle in this project, is then given. The following section introduces the real-time contour tracking library, which is the extended Kalman filter-based contour tracker that all smoothing algorithms in this project are based on.

The last section discusses the covariance intersection algorithm, which is an integral part of the smoothing algorithms, used to fuse two or more estimates into a single best estimate. It will be shown that it comes in two slightly different formulations, depending on what is known about the correlation between the estimates used as input.

2.1 Medical ultrasound

Medical ultrasound imaging is based on pulse echo measurements. A short pulse emitted by a transducer propagates through tissue and energy is reflected back as the pulse passes through layers with different acoustic impedances. The distance to the origin of an echo is determined using the two-way transit time, while the echo strength carries information about the acoustic impedance mismatch that caused it. The frequencies used in medical ultrasound are typically between 1 and 15 MHz.

By assuming a constant speed of sound c $\left[\frac{\mathrm{m}}{\mathrm{s}}\right]$, the formula for distance as a function of two-way transit time becomes

$$d(t) = \frac{ct}{2}$$

An average speed of sound of $c = 1540$ $\frac{\mathrm{m}}{\mathrm{s}}$ is commonly used in medical ultrasound and is justified by the human body being mostly composed of water.

In order to obtain a sector scan (2d) or a volume scan (3d) of a region of interest, pulses must be emitted in different directions. After transmission of a single pulse, the transducer is used as receiver to detect the returned echoes. This introduces a delay determined by the maximum depth; enough time must

be allocated for the pulse to reach the maximum depth and any echo to return to the probe.

The beam steering can be done by moving the probe mechanically, or electronically using a phased array probe, which is composed of a (one- or two-dimensional) array of small transducer elements. The emitted beam is then the superposition of the output from the individual elements, and can be steered by applying suitable delays to the individual elements. A more detailed treatment can be found in [8, p. 177].

The focal point of the phased array probe can also be changed dynamically on receive by adjusting the individual delays. A real-time scan of the region of interest is obtained by repeating the scanning procedure multiple times per second.

Medical ultrasound is a very safe imaging modality. According to [9], "human injury has never been attributed to clinical practice of diagnostic ultrasound."

Ultrasound is inherently affected by speckle noise, which refers to a texture-like noise pattern that degrades the quality of ultrasound images. It is caused by the destructive and constructive interference pattern resulting from many fixed tissue scatterers that are much smaller than the wavelength of the ultrasound pulses [8, p. 215]. Although superficially resembling it, the speckle noise does not correspond to underlying tissue structure.

The effect of speckle is to reduce contrast and visibility of boundaries of anatomical structures. In [10] the negative effect of speckle noise is demonstrated. It is shown that it leads to a reduction in lesion (tissue abnormality) detectability of approximately a factor of eight. Speckle also complicates the task of correctly identifying edges between anatomical structures.

There is also a positive aspect of speckle noise; it conveys information about the motion of the tissue being imaged. This can be taken advantage of with *speckle tracking*, which has been applied for tracking and strain computations in multiple studies.

2.2 The human heart

The heart is a double pump which is responsible for maintaining the blood flow through the human body. It consists of four chambers with valves to maintain correct flow direction. These four chambers are the left and right atrium, and the left and right ventricle. Figure 1 shows a simplified illustration of the main anatomical structures. Referring to figure 1, venous blood containing carbon dioxide and depleted of oxygen enters the right atrium, which contracts as the right ventricle relaxes. The blood then enters the right ventricle, from which it is

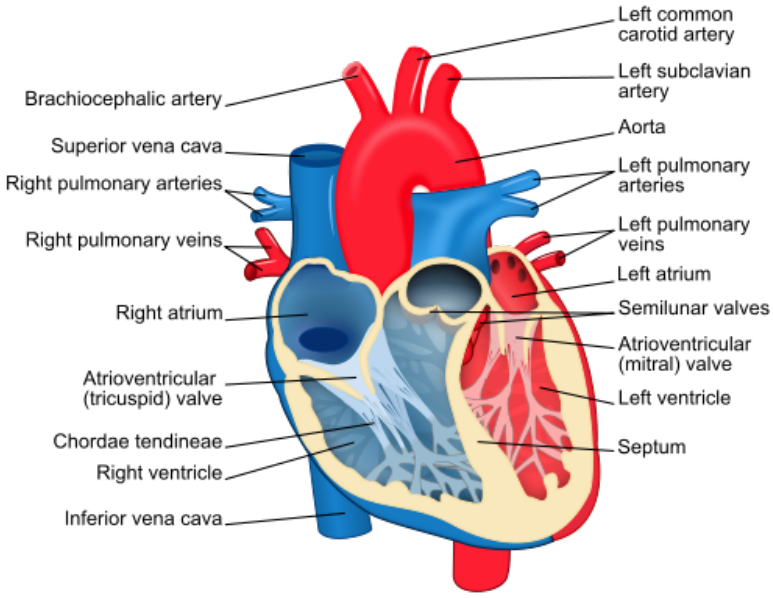


Figure 1: Cross-section of the the human heart. Image courtesy of wikipedia user *ZooFari*, retrieved from http://en.wikipedia.org/wiki/File:Heart_diagram-en.svg

subsequently pumped through the lungs, where gas exchanges occurs. Carbon dioxide is released and oxygen-enriched blood flows to the left atrium, which contracts at the same time as the left ventricle relaxes. This fills the left ventricle with oxygen-rich blood which is finally pumped to the organs when it contracts. This cyclic behavior is illustrated in figure 2.

The term *cardiac cycle* refers to the heart's contraction and relaxation. The relaxation phase when the ventricles fills with blood is termed *diastole*, and the contraction phase where blood is ejected from the ventricles is termed *systole*.

The amount of blood pumped out during one heart cycle can be found as the volume of blood in the left ventricle at the end of diastole (*end-diastolic volume*) minus the volume of blood present in the left ventricle at the end of the contraction phase (*end-systolic volume*). This is called the *stroke volume* and

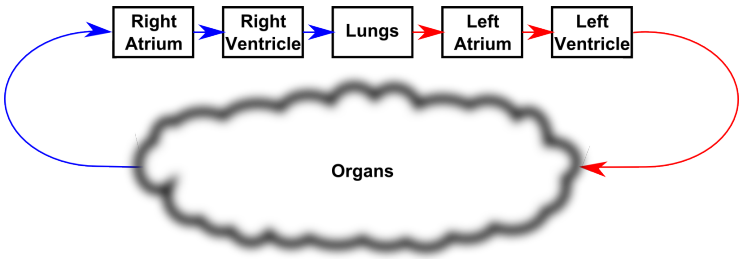


Figure 2: Illustration of the blood flow in the human body. Red arrows symbolize oxygen-rich blood while blue arrows symbolize oxygen-depleted blood.

can thus be computed by the following formula.

$$\mathrm{SV} = \mathrm{EDV} - \mathrm{ESV} \tag{1}$$

A much used measure of heart performance is the *ejection fraction* which is defined as the percentage of EDV that is ejected.

$$\mathrm{E}_f = \frac{\mathrm{SV}}{\mathrm{EDV}} = \frac{\mathrm{EDV} - \mathrm{ESV}}{\mathrm{EDV}} \tag{2}$$

A healthy person typically have an ejection fraction of approximately 60 % [11].

2.3 The Kalman filter

The Kalman filter is a recursive algorithm for computing optimal state estimates given noisy observations and knowledge of the system dynamics and the relation between system states and measurement vectors [12]. It maintains a Gaussian posterior distribution conditioned on the set of observations so far. The Kalman filter requires linear system- and measurement models. The extended Kalman filter and the unscented Kalman filter are two approaches for relieving this constraint. However, these extensions are not optimal estimators in general. Only the extended Kalman filter will be discussed, since this is the approximation technique used in the real-time contour tracking library.

The notation used throughout the report is as follows. The expression $\hat{\mathbf{x}}(k|l)$ denotes an estimate for the value of $\mathbf{x}$ at discrete time k, given observational data up to and including discrete time l. In the same manner, the expression $\mathbf{P}_{k|l}$ denotes the covariance associated with the above estimate.

Because the Kalman filter is usually employed in control-theoretic applications, its equations normally include a control term, i.e. the known control inputs to the system. This is not needed in contour tracking applications and hence dropped from the equations.

The original Kalman filter requires linear models because it involves computing the mean and covariance of a transformed Gaussian distribution, which is simple in the linear case [12, pp. 53]. The non-linear case is highly non-trivial and some sort of approximation is needed since the optimal solution of the non-linear filtering problem involves a potentially unbounded number of parameters [13], making real-time operation difficult. The extended Kalman filter linearizes the non-linearities around an estimate.

The *information filter formulation* is algebraically equivalent to the standard formulation of the Kalman filter and is often used to reduce the computational requirements when the number of measurements exceeds the number of states, which is the case in the real-time contour tracking library.

2.3.1 The dynamic system model

The Kalman filter algorithm requires the following framework consisting of a state-transition function $f(k)$ and a measurement function (or measurement model) $h(k)$. The state-transition function models the evolution of the system with time, i.e. given the state vector $\mathbf{x}(k)$ at discrete time k, it returns $\mathbf{x}(k+1)$, the state vector at time $k+1$.

The measurement function defines the relationship between a state vector and the observation vector, i.e. the observation vector $\mathbf{z}$ obtained if the system is in a given state $\mathbf{x}$. Both functions are allowed to be time-varying.

$$\mathbf{x}(k+1) = f[\mathbf{x}(k), k] + \mathbf{w}(k) \tag{3a}$$

$$\mathbf{z}(k) = h[\mathbf{x}(k), k] + \mathbf{v}(k) \tag{3b}$$

$$\mathbb{E}\left\{\mathbf{w}_k \mathbf{w}_j^T\right\} = \mathbf{Q}_k \delta(k-j) \quad \text{[Process noise cov.]} \tag{3c}$$

$$\mathbb{E}\left\{\mathbf{v}_k \mathbf{v}_j^T\right\} = \mathbf{R}_k \delta(k-j) \quad \text{[Measurement noise cov]} \tag{3d}$$

$$\mathbb{E}\left\{\mathbf{w}_k \mathbf{v}_j^T\right\} = 0 \;\forall\; k, j \quad \text{[Noise independency]} \tag{3e}$$

The vector $\mathbf{w}(k)$ is the process noise, which is assumed to be zero-mean Gaussian with covariance matrix $\mathbf{Q}_k$ and the vector $\mathbf{v}(k)$ is the measurement noise, which is assumed to be zero-mean Gaussian with covariance matrix $\mathbf{R}_k$. These noise terms represent system modeling errors and unknown measurement disturbances, respectively.

The original Kalman filter requires linear state-transition and measurement functions. In this case equations 3 reduces to the following.

$$\mathbf{x}(k+1) = \mathbf{F}_k\mathbf{x}(k) + \mathbf{w}(k) \tag{4a}$$
$$\mathbf{z}(k) = \mathbf{H}_k\mathbf{x}(k) + \mathbf{v}(k) \tag{4b}$$

where $\mathbf{F}_k$ and $\mathbf{H}_k$ are possibly time-varying matrices.

2.3.2 The original Kalman filter

The original Kalman filter was introduced in 1960 by Rudolf Kalman [14]. It is a recursive algorithm for estimating the state of a system in such a way that the the expected minimum-mean squared error is minimized [12].

The algorithm operates in a prediction-correction cycle. First, the state vector and covariance of the next time step are predicted using the state-transition function. Then an observation is made and the updated state and covariance is computed as a linear combination of the information from the measurement vector and the prediction. A complete derivation of these equations can be found in [12].

It is also possible to derive the Kalman filter from a statistical point of view [15].

Prediction

$$\hat{\mathbf{x}}(k+1|k) = \mathbf{F}_k\hat{\mathbf{x}}(k|k) \tag{5a}$$
$$\mathbf{P}_{k+1|k} = \mathbf{F}_k\mathbf{P}_{k|k}{\mathbf{F}_k}^T + \mathbf{Q}_k \tag{5b}$$

Update

$$\mathbf{K}_{k+1} = \mathbf{P}_{k+1|k}{\mathbf{H}_{k+1}}^T \left[\mathbf{H}_{k+1}\mathbf{P}_{k+1|k}{\mathbf{H}_{k+1}}^T + \mathbf{R}_{k+1}\right]^{-1} \tag{6a}$$
$$\hat{\mathbf{x}}(k+1|k+1) = \hat{\mathbf{x}}(k+1|k) + \mathbf{K}_{k+1}\left[\mathbf{z}(k+1) - \mathbf{H}_{k+1}\hat{\mathbf{x}}(k+1|k)\right] \tag{6b}$$
$$\mathbf{P}_{k+1|k+1} = \mathbf{P}_{k+1|k} - \mathbf{K}_{k+1}\mathbf{H}_{k+1}\mathbf{P}_{k+1|k} \tag{6c}$$

2.3.3 Extended Kalman filter

The extended Kalman filter is an extension of the original Kalman filter to systems where the state- and measurement models are allowed to be arbitrary differentiable functions. However, this extension is at the expense of the filter's optimality. The extended Kalman filter may also diverge in certain cases.

The state-transition- and measurement matrices are now the Jacobian matrices of the state-transition function and measurement function given by equations 3a and 3b, respectively.

$$\mathbf{F}_k = \left.\frac{\partial f(k)}{\partial \mathbf{x}}\right|_{\mathbf{x}=\hat{\mathbf{x}}(k|k)} \tag{7}$$

$$\mathbf{H}_{k+1} = \left.\frac{\partial h(k+1)}{\partial \mathbf{x}}\right|_{\mathbf{x}=\hat{\mathbf{x}}(k+1|k)} \tag{8}$$

$$\tag{9}$$

With these modifications, the Kalman filter equations from section 2.3.2 becomes

Prediction

$$\hat{\mathbf{x}}(k+1|k) = f[\hat{\mathbf{x}}(k|k), k] \tag{10a}$$

$$\mathbf{P}_{k+1|k} = \mathbf{F}_k \mathbf{P}_{k|k} {\mathbf{F}_k}^T + \mathbf{Q}_k \tag{10b}$$

Update

$$\mathbf{K}_{k+1} = \mathbf{P}_{k+1|k} {\mathbf{H}_{k+1}}^T \left[\mathbf{R}_{k+1} + \mathbf{H}_{k+1}\mathbf{P}_{k+1|k}{\mathbf{H}_{k+1}}^T\right]^{-1} \tag{11a}$$

$$\hat{\mathbf{x}}(k+1|k+1) = \hat{\mathbf{x}}(k+1|k) + \mathbf{K}_{k+1}\left[\mathbf{z}(k+1) - h[\hat{\mathbf{x}}(k+1|k), k+1]\right] \tag{11b}$$

$$\mathbf{P}_{k+1|k+1} = \mathbf{P}_{k+1|k} - \mathbf{K}_{k+1}\mathbf{H}_{k+1}\mathbf{P}_{k+1|k} \tag{11c}$$

2.3.4 Extended information filter

The extended information filter is algebraically equivalent to the extended Kalman filter, but more efficient when the dimensionality of the measurement vector exceeds that of the state vector. The following description is based on [16].

The prediction stage is the same as in the extended Kalman filter (equations 10), but the update stage is different. The system model is the same as for the extended Kalman filter and is given by equations 3.

$$\mathbf{P}_{k|k}{}^{-1} = \mathbf{P}_{k|k-1}{}^{-1} + \mathbf{H}_k{}^T \mathbf{R}_k{}^{-1} \mathbf{H}_k \tag{12a}$$

$$\hat{\mathbf{y}}(k|k) = \hat{\mathbf{y}}(k|k-1) + \mathbf{H}_k{}^T \mathbf{R}_k{}^{-1} (\mathbf{H}_k \hat{\mathbf{x}}(k|k-1) + \mathbf{z}(k) - h[\hat{\mathbf{x}}(k|k-1), k]) \tag{12b}$$

where $\hat{\mathbf{y}}(k|k) = \mathbf{P}_{k|k}{}^{-1} \hat{\mathbf{x}}(k|k)$ and and $\mathbf{z}(k)$ is the observed measurement vector at time step k. This shows that the extended information filter maintains $\mathbf{P}_{k|k}{}^{-1}$ and $\hat{\mathbf{y}}(k|k)$ in the filter loop, as opposed to $\mathbf{P}_{k|k}$ and $\hat{\mathbf{x}}(k|k)$ in the extended Kalman filter.

Equations 12 shows that except for the matrix $\mathbf{R}_k$, all inversions involves matrices of the same dimensionality as the state vector. This ensures an asymptotic running time independent of the dimensionality of the measurement vector, in contrast to the extended Kalman filter as described in section 2.3.3.

2.4 Kalman smoothers

The standard Kalman filter formulation does not utilize measurement information past current time-step, even if so is available. The term smoothing refers to state estimates of the form $\hat{\mathbf{x}}(k|l)$, where $l > k$, i.e. state estimates based on future measurements. In a real-time situation such future measurements are of course not available, but can be if a small time-lag is tolerable or off-line post-processing is performed.

The different Kalman smoothers can be classified into three types: fixed-point smoothing, fixed-lag smoothing and fixed-interval smoothing.

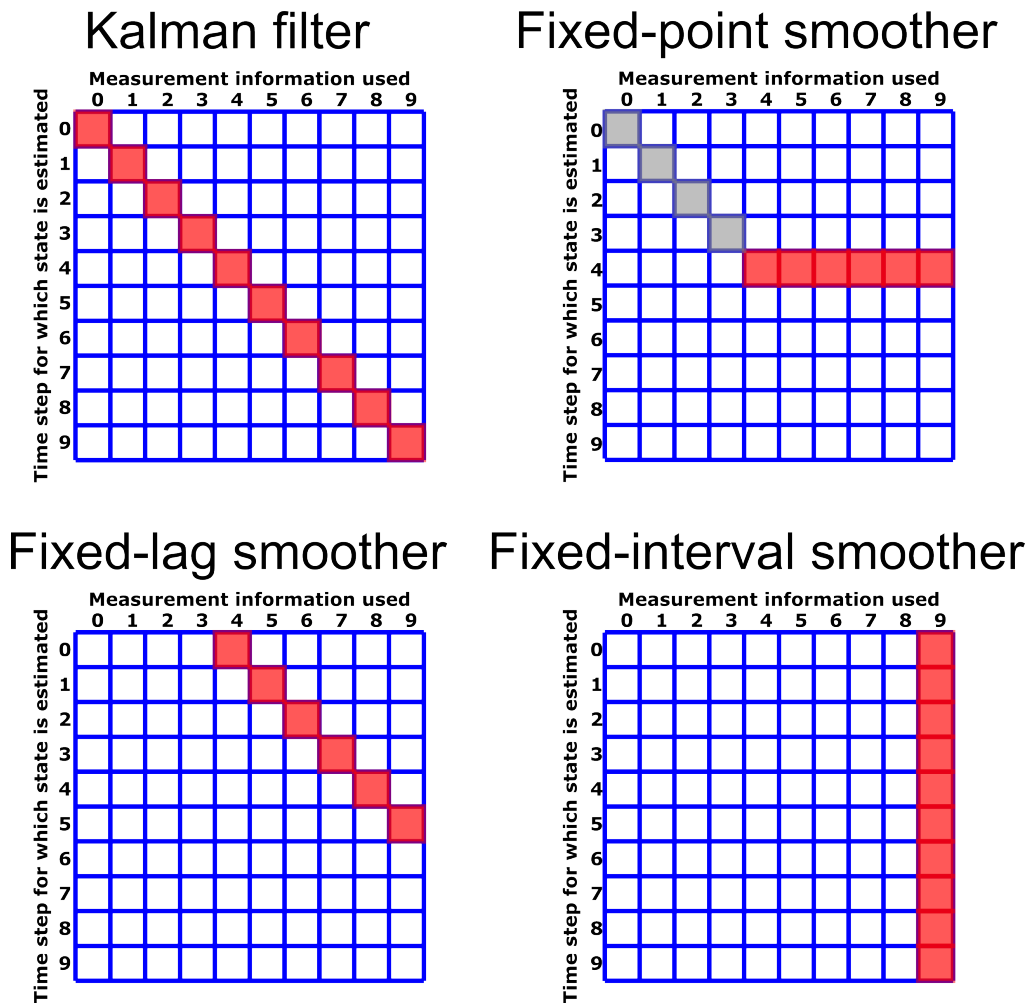


Figure 3: Graphical comparison of the Kalman filter to fixed-point, fixed-lag, and fixed-interval Kalman smoothers. The red squares show computed posterior estimates. The gray squares in the fixed-point smoother are forward posterior estimates needed before the fixed-point of interest is reached. Both the fixed-point and fixed-lag is four. In the fixed-interval smoother it is assumed that the measurement data interval is fixed and finite, while the other algorithms may continue on indefinitely in the same manner.

Figure 3 shows a simple graphical comparison of the Kalman filter and the three smoother types. The horizontal axis denotes the set of observations used as basis when computing state estimates, i.e. measurement information used equal to k means that the set $\{\mathbf{z}(0), \mathbf{z}(1), \ldots, \mathbf{z}(k)\}$ of measurements has been used.

The fixed-point Kalman filter produce a better and better estimate for the state at a single time step as more and more measurements arrive. The fixed-lag smoother, on the other hand, computes an estimate of the state a certain number of time steps before current measurement. Finally, the fixed-interval smoother computes a state estimate conditioned on all of the measurements for each time step. This smoother type assumes a fixed and finite set of measurements and is therefore restricted to post-processing only.

By inspecting figure 3, the fixed-point Kalman smoother appears to be uninteresting for segmentation of dynamic anatomical structures since the results would be estimates of the segmentation at a single fixed instant in time, but increasing in accuracy as more and more measurements arrives. It is the periodic motion of the heart that makes fixed-point smoothing interesting, since the estimate of e.g. the left ventricle at a specific point in the heart-cycle can be based on a data set containing many heart-cycles.

The different smoother types comes in different mathematical formulations. Usually, the regular Kalman filter equations are employed together with a separate set of equations used for a backward pass. This backward pass use the future measurement data to compute correcting terms for the regular forward Kalman filter estimates.

In [17] an important interpretation of the fixed-interval smoother as a combination of two optimal filters running in each direction is presented. This idea is further extended in [12] to give a practical means for implementing the fixed-lag smoother for reasonably short fixed-lags. This interpretation of the smoothers will be referred to as *forward-backward* smoothers, and serve as the theoretical foundation for actual implementations. The forward-backward approach allows reuse of existing Kalman filters.

2.4.1 Forward-backward approach to fixed-interval smoothing

The forward-backward Kalman smoother is now summarized. The forward filter is operated in the usual way, giving a posterior state and covariance estimate of the interior point of interest, $\mathbf{x}(k|k)$ and $\mathbf{P}_{k|k}$.

The backward running Kalman filter starts with the last measurement, ends on the first measurement, and is initialized with a prior estimate of infinite

covariance (which assures that the initial state estimate has no influence). It differs from the forward Kalman filter by performing the prediction stage using the following equations.

$$\hat{\mathbf{x}}(k+1)_{\text{backward}} = \mathbf{F}_k{}^{-1}\hat{\mathbf{x}}(k)_{\text{backward}} \tag{13a}$$

$$\mathbf{P}_{k+1,\text{backward}} = \mathbf{F}_k{}^{-1}\left[\mathbf{P}_{k,\text{backward}} + \mathbf{Q}_k\right]\left(\mathbf{F}_k{}^{-1}\right)^T \tag{13b}$$

Where $\mathbf{F}_k{}^{-1}$ is the inverse of the state-transition matrix used in the forward-running Kalman filter.

The update stage is performed as usual. The backward-running Kalman filter is stopped at the time step before (in backward direction) the point of interest; then a *prediction* step is performed. The two estimates for the time step of interest is then fused with the covariance intersection algorithm for independent estimates given by equation 17. This is done for all time-steps and yields the fixed-interval smoothed estimates.

By comparing equations 13 with the regular Kalman filter equations 5, it is apparent that the backward Kalman filter is equivalent to a regular Kalman filter with two modifications: the state transition matrix $\mathbf{F}$ is replaced by its inverse and the process noise covariance matrix $\mathbf{Q}$ is replaced with $\mathbf{Q}_{\text{backward}} = \mathbf{F}^{-1}\mathbf{Q}\left(\mathbf{F}^{-1}\right)^T$.

2.4.2 Forward-backward approach to fixed-lag smoothing

The fixed-lag smoother is implemented in the same manner as the fixed-interval smoother. A smoothed estimate for the state at time step k is computed with fixed-lag L by first using the forward filter to obtain an estimate $\hat{\mathbf{x}}_\mathbf{f}(k|k)$ of the state at time k. Then a backward filtering pass is performed starting at time $k+L$ with no prior knowledge and ending with a state prediction $\hat{\mathbf{x}}_\mathbf{b}(k|k+1)$ from time step $k+1$ to time step k. Finally the forward and backward estimates $\hat{\mathbf{x}}_\mathbf{f}(k|k)$ and $\hat{\mathbf{x}}_\mathbf{b}(k|k)$ are combined using covariance intersection. This procedure is illustrated in figure 10. The remarks in section 2.4.1 on the backward filter applies in this case as well.

It should be noted that with a fixed-lag of L time-steps, the smoother is not able to compute any estimates before enough forward estimates are obtained. Hence a delay of L time-steps in introduced, and this technical difficulty must be paid attention to in an actual implementation.

2.4.3 Numerical example: applying the forward-backward Kalman smoother to a damped mass-spring system

Figure 4a shows a damped mass-spring system. The governing differential equation and the analytical solution is derived in appendix D. The forward-backward fixed-lag and fixed-interval Kalman smoothing algorithms will be demonstrated on this system, and the results referred to in the discussion of smoothing algorithms within the RCTL framework.

The state-transition matrix is used to generate a vector of mass displacement values, subsequently transformed to an observation vector by addition of Gaussian noise. A Kalman filter runs forward and backward and the estimates are fused at all time steps in the measurement data interval, yielding the fixed-interval smoothed estimates.

The position estimation errors are computed at all time-steps for the forward, backward, and smoothed state estimates, and it is demonstrated that the fixed-interval smoother produces the best estimates. The same procedure is used to evaluate the fixed-lag smoother for two different lag values in order to demonstrate both that the fixed-interval smoother produces better results than the fixed-lag smoother and that the performance gain from smoothing increases with the lag-value.

Two additional experiments are performed: not inverting the state-transition matrix on the backward pass, and utilizing final update steps instead of prediction steps. This represents two deviations from the theory presented in section 2.4.1.

Figure 4 shows state estimates and associated squared position estimation errors for the forward and backward Kalman filters and the fused estimates, while figure 5 shows the estimated diagonal terms of the state covariance matrix.

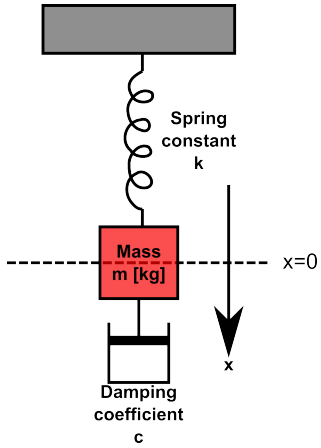
In order to get a system oscillating at a few Hertz and decaying fast enough to be significantly non-stationary in the time interval from zero to ten seconds, the following parameters for the mass-spring system were used.

- Mass $m = 0.200$ [kg]
- Spring constant: $k = 60.0[\frac{\text{kg}}{\text{s}^2}]$
- Damping coefficient: $c = 0.10$

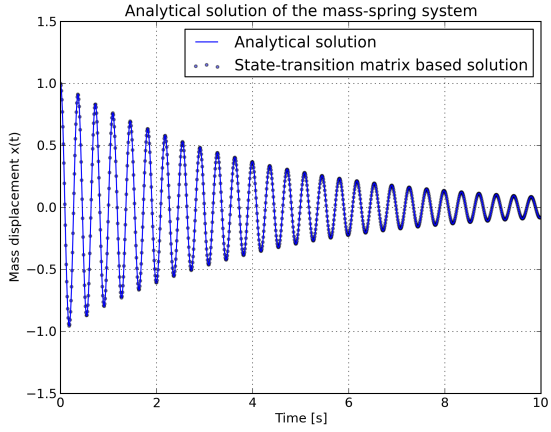
The state of the system was simulated using an initial displacement of 1.0 [m] and a velocity of 0.0 $[\frac{\text{m}}{\text{s}}]$, using the state-transition matrix with a time-step of 0.01 s. With the above parameter values the state-transition can be determined numerically as shown in appendix D.

$$\phi = e^{0.1\begin{pmatrix} 0 & 1 \\ -\frac{k}{m} & -\frac{c}{m} \end{pmatrix}} = e^{\begin{pmatrix} 0 & 0.01 \\ -3 & -0.005 \end{pmatrix}} = \begin{pmatrix} 0.98506236 & 0.00992524 \\ -2.97757236 & 0.98009974 \end{pmatrix}$$

Figure 4b shows the analytical continuous-time simulation as well as discrete time-steps obtained from the state-transition matrix ϕ.



(a) The damped mass-spring system



(b) Analytical solution of the damped mass-spring oscillation shown with discrete displacement values computed using the state-transition matrix.

In order to keep the example simple, only scalar observations of the position (mass displacement) were used as measurement input to the Kalman filters. Hence, the measurement matrix is

$$\mathbf{H} = \begin{pmatrix} 1 & 0 \end{pmatrix}$$

The vector of noisy position measurements was created by adding zero-mean Gaussian noise with variance 12.0 to the simulated position vector.

The Kalman filter was initialized with the correct measurement noise variance $R = 12.0$, and with the process noise covariance matrix

$$\mathbf{Q} = \begin{pmatrix} 1.0 \cdot 10^{-6} & 0 \\ 0 & 1.0 \cdot 10^{-6} \end{pmatrix}$$

This small process noise is justified by the fact that there are no system modeling errors in this case, by design of the example. The initial state estimate used by the Kalman filter was position 10.0 m and velocity 1.0 $\frac{\mathrm{m}}{\mathrm{s}}$ and covariance matrix

$$\mathbf{P}_0 = \begin{pmatrix} 10 & 0 \\ 0 & 10 \end{pmatrix}$$

The regular Kalman filter results are obtained from the forward pass of the fixed-interval smoother. The results are shown in table 1, where the results have been averaged over at least 50000 simulations in order for the statistics to converge. All smoothing algorithms have an identical forward pass, and hence the same forward squared position error.

Algorithm	**Forward squared position error [m^2]**	**Backward squared position error [m^2]**	**Smoothed squared position error [m^2]**	**Improvement by smoothing**
Fixed-interval smoother	0.109	0.265	0.014	7.8
Fixed-interval smoother, final update	-	0.265	0.014	7.8
Fixed-interval smoother, no inverse	-	0.230	0.076	1.4
Fixed-lag 3 smoother	-	-	0.070	1.6
Fixed-lag 16 smoother	-	-	0.044	2.5

Table 1: Numerical simulation results. Squared backward error is only computed for the fixed-interval smoother. The experiments are averaged over at least 50000 repetitions so that at least the first two decimals have converged. Right column is ratio between forward error and smoothed error. The forward error is the same for all smoothing algorithms.

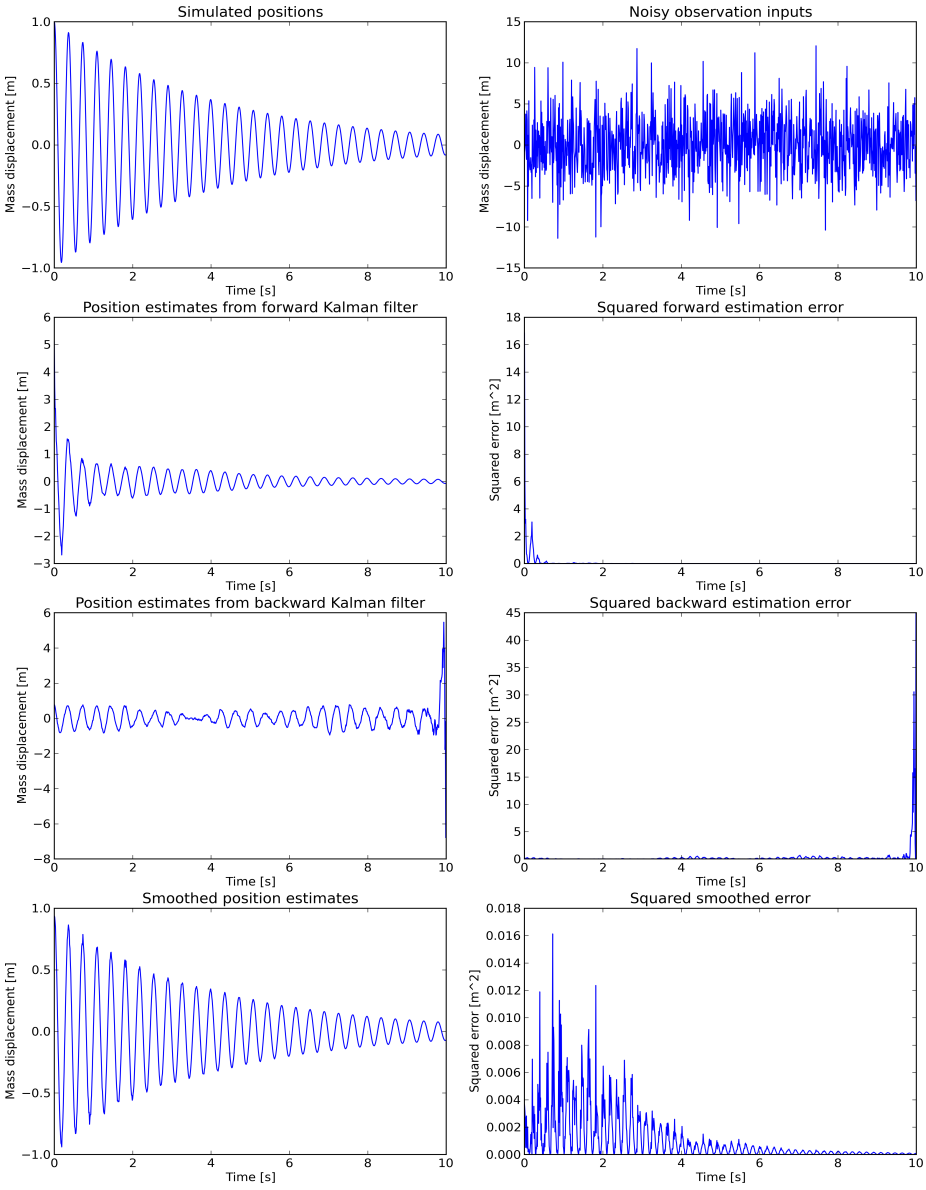


Figure 4: Forward-backward smoothing on the simple mass-spring system.

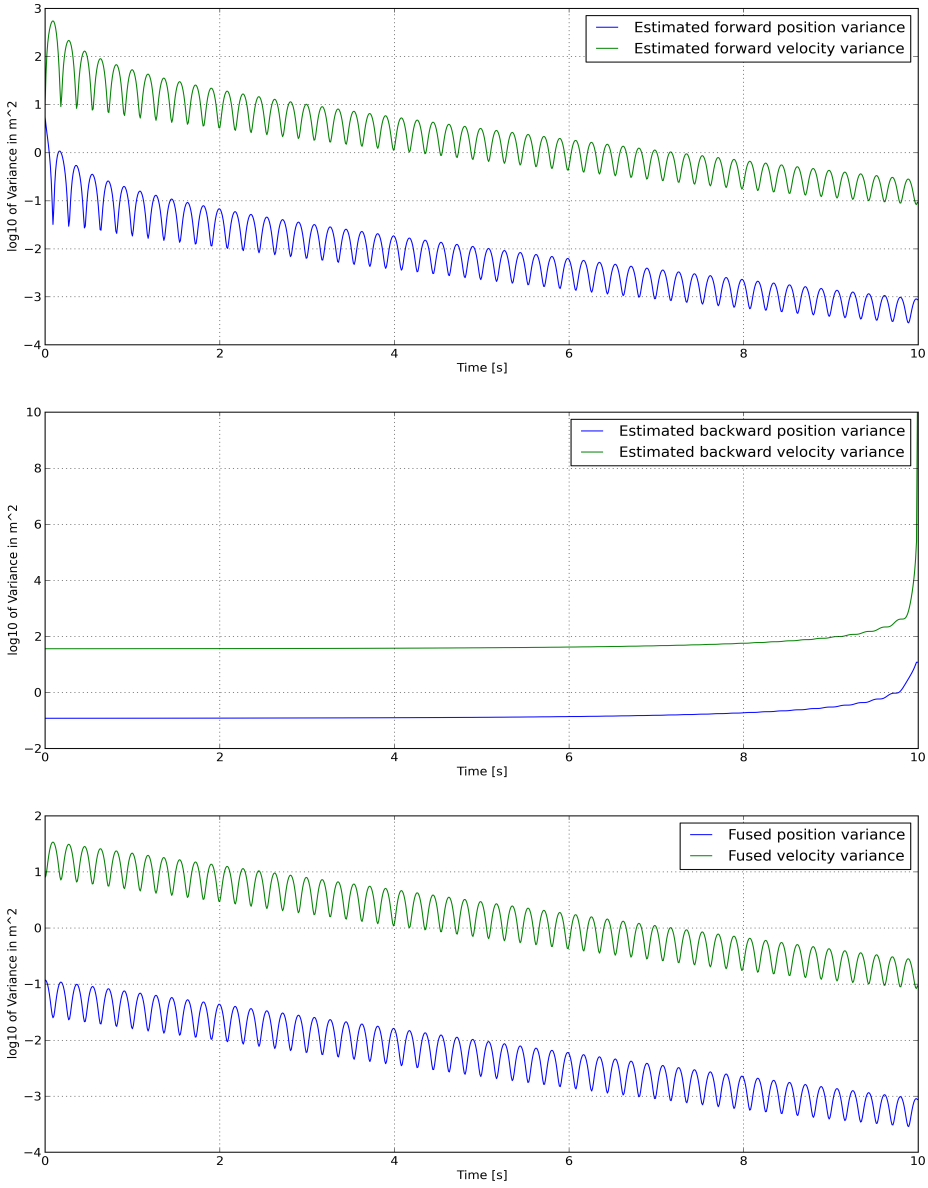


Figure 5: Logarithm of the estimated position and velocity covariances.

2.5 Subdivision surfaces

This section contains a brief discussion of subdivision surfaces. The focus will be on Doo-Sabin surfaces, the specific type implemented in the real-time contour tracking library. References are given to more in-depth treatments.

The theory behind subdivision surfaces is complicated, but the basic principles are easy. Subdivision surfaces offer smooth surfaces requiring only a small number of parameters for their specification. This contrasts e.g. triangular meshes where smoothness require many triangles, and hence many parameters, and depends on the scaling.

A subdivision surface is the limit of a recursive refinement process. Examples of subdivision schemes are Catmull-Clark [18] and Doo-Sabin [19, p. 188]. The basic principle is to start with an initial coarse mesh and repeatedly "cut the corners off" in a deterministic manner. Each iteration increases the surface smoothness, but also the number of vertices, edges and faces. The important point is that the limit surface is fully determined by the initial mesh.

A more mathematical treatment of subdivision surfaces can be found in [20]. The technical details are hidden from users of the real-time contour tracking library, and will not be discussed here.

2.5.1 Terminology

Below is a summary of terminology commonly used in computer graphics.

- Node: A point in space.
- Control point / control node: A node used to control the shape of a curve or surface.
- Edge: The line between two nodes.
- Face: The area enclosed by edges. In a triangular mesh, the faces are triangular and in a quadrilateral mesh, the faces are quadrilaterals.
- Renderer: Software that converts a mesh into the final two-dimensional projection on a screen or other medium.
- Mesh: A collection of vertices, edges and faces together with the topological information.

Subdivision schemes come in two categories: approximating and interpolating. The difference is whether or not the initial control points remain in the

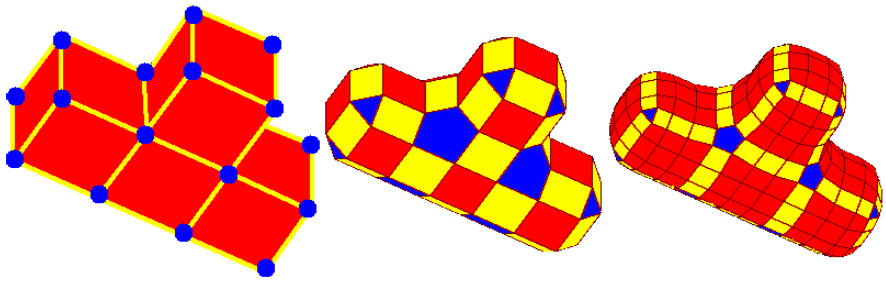


Figure 6: Example showing successive subdivisions of a Doo-Sabin surface. Figure courtesy of Fredrik Orderud. Retrieved from *http://upload.wikimedia.org/wikipedia/commons/5/56/DooSabin_subdivision.png*

same position after a subdivision (interpolating). Different mesh types can be used, and the real-time contour tracking library employs quadrilateral meshes, i.e. meshes that are composed of quadrilaterals.

2.5.2 Doo-Sabin subdivision surfaces

According to [6], Doo-Sabin surfaces have the desirable property that the inherent smoothness and continuity of surface derivatives of spline surfaces is combined with the support of arbitrary topology offered by flat polygonal meshes. Figure 6 shows examples of subdivision steps of a Doo-Sabin surface. Continuing this process yields the smooth limit surface.

In addition to the recursive subdivision scheme, [6] presents a method for exact surface evaluation at arbitrary parametric positions, based on a similar result for Catmull-Clark surfaces [21]. This is used in RCTL to compute the spatial positions of the edge-detectors on the deformed model.

2.6 The real-time contour tracking library

This section contains a brief description of the *real-time contour tracking library*. A more thorough description can be found in [22] and [23].

The real-time contour tracking library is written in C++ and capable of performing real-time segmentation of two- and three-dimensional image data, including but not restricted to ultrasound recordings. It was primarily developed by Fredrik Orderud in cooperation with GE Vingmed Ultrasound during his

PhD. In addition to the library, there also exist a Matlab interface and user interfaces for two- and three-dimensional operation.

The configuration of parameters of the extended Kalman filter in RCTL, including the setup of the model hierarchy, is done through .xml configuration files.

The segmentation is performed using a state-space approach. A state vector is used to encode the shape information and a Kalman filter is used to estimate the deformation and positioning needed to fit the deformable model onto the image data. Edge-detections are performed from a set of points distributed on the deformable contour, along the surface normals (in both directions). The aim is to measure the correct positions of the points in the image data, in order to update the state vector. This is done on a frame-by-frame basis on the input recording. A more detailed description can be found in [23].

The main design goal of this library is speed. Although there are alternative segmentation schemes available, the Kalman filter-based approach in the real-time contour tracking library is probably among the world's fastest. In this project, ultrasound recordings in the DICOM format have been used. [1]

2.6.1 Deformable models

The type of deformable models used in this project is Doo-Sabin surfaces, which was discussed in section 2.5. The model data is stored in a HDF5 file [2] These files contains the parameterized shape template, including information about which control points that are allowed to move, and the distribution of edge-detection points on the surface. Given a HDF5 file and a vector containing the deformation parameters, the contour is fully described.

By allowing control points to move along the initial surface normal intersecting it, the *local deformation model* can be introduced. By letting $x_{l,i}$ denote the i-th normal displacement, we get the following expression for contour control points

$$\mathbf{q}_i = \bar{\mathbf{q}}_i + x_{l,i}\mathbf{n}_i \tag{14}$$

where $\bar{\mathbf{q}}_i$ is the initial position and $\mathbf{n}_i$ is the initial surface normal. If a control point is stationary, the corresponding normal displacement value is fixed at zero.

[1] DICOM is short for Digital Imaging and Communications in Medicine. It is a standard used in medical imaging.

[2] HDF5 is a flexible file format for storing and managing data. For more information, see http://www.hdfgroup.org

Surface points can then be computed as sums of control points weighted by the basis functions

$$\mathbf{p}_l = \sum_i b_i \mathbf{q_i} \tag{15}$$

In addition to the local deformation model which permits local modification of a shape template, the *global deformation model* is introduced. This is an affine transform T_g that acts on the whole model and permits translation, rotation and scaling of the deformed model.

$$\mathbf{p} = T_g(\mathbf{p}_l) \tag{16}$$

where $\mathbf{p}_l$ is given by equation 15. Parametric contour coordinate information is suppressed in these equations.

In summary, the deformable model consists of a parameterized surface with an initial shape and with control point movement, translation, rotation and scaling available as adjustable parameters in order to fit it onto an image.

2.6.2 Edge-detection normal displacement measurements

The edge-detection results is the information enabling updating of the deformable models. There are a number of different algorithms to choose from, but all offer the functionality described by an abstract edge-detector interface. It suffices to know that given a set of starting points, search directions, and a frame, functionality exists for returning the corresponding most likely intensity transitions.

The edge-detector types *step* and *gradient* has been used in this project. The step edge-detector is used for all tracking on ultrasound data. It detects a transition from one brightness level to another, such as the left-ventricular border separating blood and the ventricle wall. The gradient edge-detector detects the maximum rate of change of brightness along the search vector, and was used when tracking rendered ground-truth meshes (described in appendix B).

Edge-detectors are grouped into *criterions*, which makes it possible to use different types and settings in different parts of the model. In this project, only a single criterion has been used in the left-ventricle model.

As the low-level implementation details is out of the scope of this report, it will only be shown by an example how edge-detections are done.

Figure 7 shows an illustration of a single edge-detection performed starting from the point $\mathbf{p}$ and searching in the direction $\mathbf{n}$. The point $\mathbf{p}_{\text{obs}}$ is the point returned as the most likely intensity transition point. The length of the red line

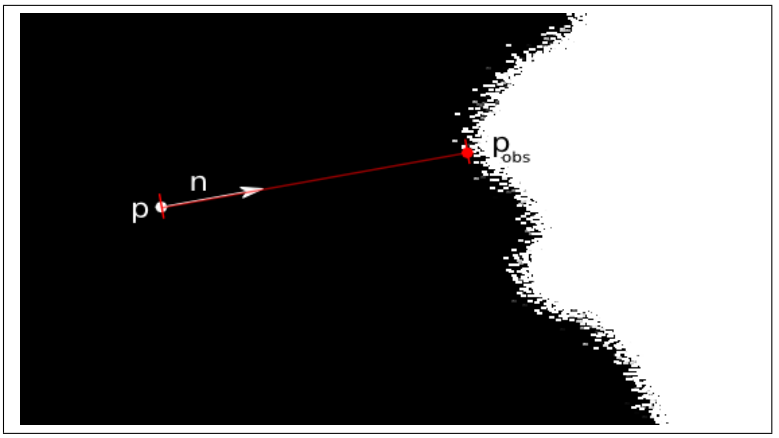


Figure 7: Illustration of a single edge-detection. The white part represent an anatomical structure in an ultrasound image. Here, $\mathbf{p}$ is the edge-detection point, $\mathbf{n}$ is the search direction and $\mathbf{p}_{\text{obs}}$ is the detected edge.

corresponds to the normal displacement as computed according to the following formula.

$$v = \mathbf{n}^T(\mathbf{p}_{\text{obs}} - \mathbf{p})$$

2.6.3 The state-transition model

The Kalman filter framework requires a state-transition function modeling the evolution of the system's state with time, as shown in equation 3. For this project, a stationary model has been used where the state-transition function is a simple regularization towards an initial shape, which means that if no measurement inputs are supplied, the contour will converge to this initial shape.

The stationary model is represented by a matrix $\mathbf{A}_1$, which defines the *regularization*, i.e. the rate at which the model converge to the shape template. More details can be found in [23].

2.6.4 The measurement model

The Kalman filter framework also requires a measurement model, which specifies the connection between measurements and state vectors. This function defines

how the filter interprets the measurement vectors. This is also the non-linear part of the tracking library.

Each deformable model has a number of edge-detection points, which is specified by a set of parametric values. The measurement vector associated with a state vector $\mathbf{x}$ is defined as the result of performing the edge-detections starting on the predicted contour, when the true contour is as defined by the state vector $\mathbf{x}$.

This is illustrated in figure 8. The resulting normal displacement measurement vector is a vector consisting of the length of the lines from edge-detection points to the red contour, in the order defined by edge-detection numbering.

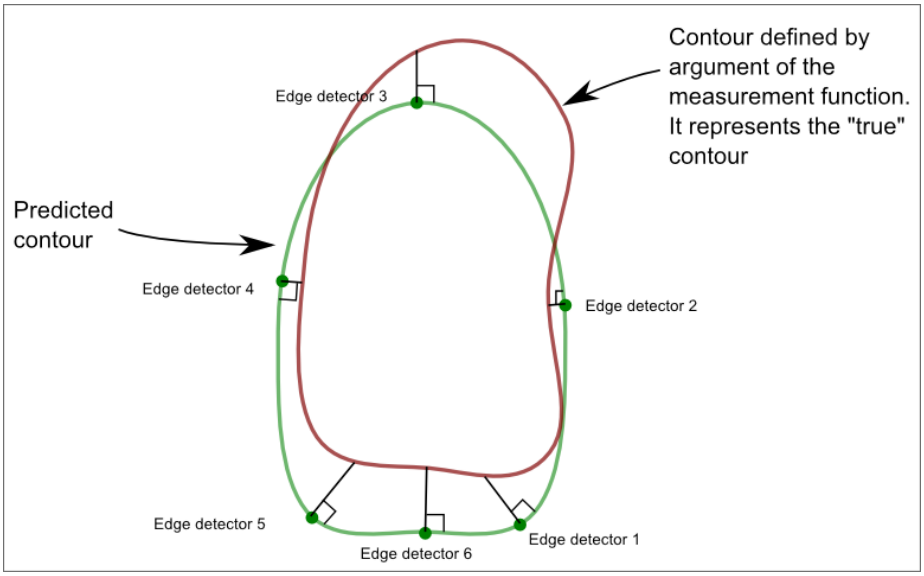


Figure 8: Illustration of the measurement model. The search is performed starting in the green edge-detection points on the predicted contour. The length of the black lines is the elements in the associated measurement vector. In this example, the number of edge-detection points is six. This illustration is for the two-dimensional case, but the same principles applies in 3d.

2.6.5 The Rctl3dApp application

The Rctl3dApp is a visualization front-end for the real-time contour tracking library. It shows arbitrary slices of ultrasound data with the fitted model and can also show volume curves.

2.6.6 RctlMatlab

It is possible to interface RCTL from Matlab. This extension is called RctlMatlab. The ultrasound recording to track on and the .xml configuration file containing details about the deformable model is specified during initialization. Filtering on arbitrary frames is then possible by specifying the frame number. Additional functionality exists for exporting the state vector and covariance estimate, updating their values, and exporting a triangle mesh-representation of a state vector.

2.6.7 RctlSlicer

There also exists a Matlab interface called RctlSlicer that permits slicing of ultrasound data and deformable models for visualization purposes. RctlSlicer was used to create most of the visualizations of slices through ultrasound data and models in this report.

2.6.8 Noise parameters and autoscaling

The noise settings control the allowed rate of change of shape deformation and global transformation as a response to edge-detection results, and are used by RCTL to construct the process noise covariance matrix.

In order to keep the edge-detection noise parameter in the .xml configuration files independent of the number of edge-detectors in use, automatic scaling of the reported edge-detection variances are in use by default. A noise value is then specified for the whole edge-detection criterion. With this approach, after performing all edge-detections belonging to an edge-detection criterion, each noise value is multiplied by a scaling factor in such a way that the sum of all variances in this criterion sums up to the value specified in the configuration file.

The auto-scaling of measurement noise covariance interferes with the smoothing algorithms, and as a convenient way of turning auto-scaling on or off, the measurement handling code was modified such that a physically meaningless noise value of -1 disables it. This auto-scaling will be discussed later.

2.7 Covariance Intersection

This section introduces the covariance intersection algorithms. The two algorithms presented differ in the assumptions they make. While the first assumes independent estimates, the second assumes no knowledge of the correlation between them.

Although only the latter is explicitly referred to as covariance intersection in the literature, for consistency also the first one will be referred to as such; it will be clear from the context which version is used.

The covariance intersection algorithms operate on estimates of the mean vector and covariance matrix of probability distributions. The goal is to combine multiple estimates into a single estimate.

Appendix C contains a simple numerical example which illustrates the differences between the two versions of the algorithm.

2.7.1 Covariance intersection under the independency assumption

Section A contains the derivation for the scalar case, and it is pointed out where the independency assumption is used. This derivation is based on [12, p. 323]. In addition, it is stated in [12] that the natural generalization to the vector case shown below is correct.

The algorithm can be summarized as follows. Given two estimates $(\hat{\mathbf{x}}_A, \mathbf{P}_A)$ and $(\hat{\mathbf{x}}_B, \mathbf{P}_B)$, the fused estimate is computed as

$$\mathbf{P} = \left[\mathbf{P}_A^{-1} + \mathbf{P_B}^{-1}\right]^{-1} \tag{17a}$$

$$\hat{\mathbf{x}} = \mathbf{P}\left[\mathbf{P}_A^{-1}\hat{\mathbf{x}}_A + \mathbf{P}_B^{-1}\hat{\mathbf{x}}_B\right] \tag{17b}$$

2.7.2 Covariance intersection under unknown correlation

The assumption that estimates are independent might not always hold in practice. In [13] it is described an alternative covariance intersection which makes no assumptions on the correlation between the two estimates to be fused.

The algorithm can be summarized as follows. Assume estimates $(\hat{\mathbf{x}}_A, \mathbf{P}_A)$ and $(\hat{\mathbf{x}}_B, \mathbf{P}_B)$ are given. The fused estimate is computed as

$$\mathbf{P} = \left[\omega\mathbf{P}_A^{-1} + (1-\omega)\mathbf{P}_B^{-1}\right]^{-1} \tag{18a}$$

$$\hat{\mathbf{x}} = \mathbf{P}\left[\omega\mathbf{P}_A^{-1}\hat{\mathbf{x}}_A + (1-\omega)\mathbf{P}_B^{-1}\hat{\mathbf{x}}_B\right] \tag{18b}$$

If $\omega \in [0, 1]$ it can be shown that the fused estimate is consistent [13]. A visualization of this fact is shown in figure 33 where the covariance ellipse of the fusion based on this algorithm encompasses the intersection of the input covariance ellipses. It can be seen that this is not the case when fusing with the covariance intersection algorithm assuming independency.

The following suggested ω weights the estimates according to the trace of the covariance matrices in such a way that the estimate with the least covariance matrix has most influence while the resulting value is always in $[0, 1]$.

$$\omega = \frac{\text{trace}(\mathbf{P}_B)}{\text{trace}(\mathbf{P}_A) + \text{trace}(\mathbf{P}_B)} \tag{19}$$

3 Implementation details

This section contains implementation details for the three Kalman smoothers and the experimental statistical smoother. All algorithms are implemented in Matlab, using the RctlMatlab interface to the RCTL library and a convenient algorithm abstraction presented in figure 15.

The real-time contour tracking library is a confidential property of GE Vingmed Ultrasound, Horten, Norway. This limits the detail level in this presentation, and for the same reason, the Matlab source code of the smoothing algorithms is not included in this report.

The forward-backward approach to Kalman smoothing is fundamental to this high-level approach for smoothing. The smoothing algorithms are built on top of Kalman filters; all that is required is black-box access to a contour tracker capable of outputting probabilistic state-vector estimates.

3.1 Fixed-point smoother

The basic principle of operation is that each heart-cycle is similar to the previous, and thus it is reasonable to expect obtaining better and better contour estimates for a specific heart-cycle position p by a proper combination of estimates from successive heart-cycles.

This smoother solution is motivated by the observation that contour estimates for a specific heart-cycle position differs from cycle to cycle when using the extended Kalman filter, even when the input is a single heart-cycle played in loop. This is demonstrated in figure 17, where 1, 10, 100, and 1000 contour estimates for the same cycle position is superimposed on a slice of the recording, with the correct ground-truth mesh showed in red. The variability is large where there is much yellow area.

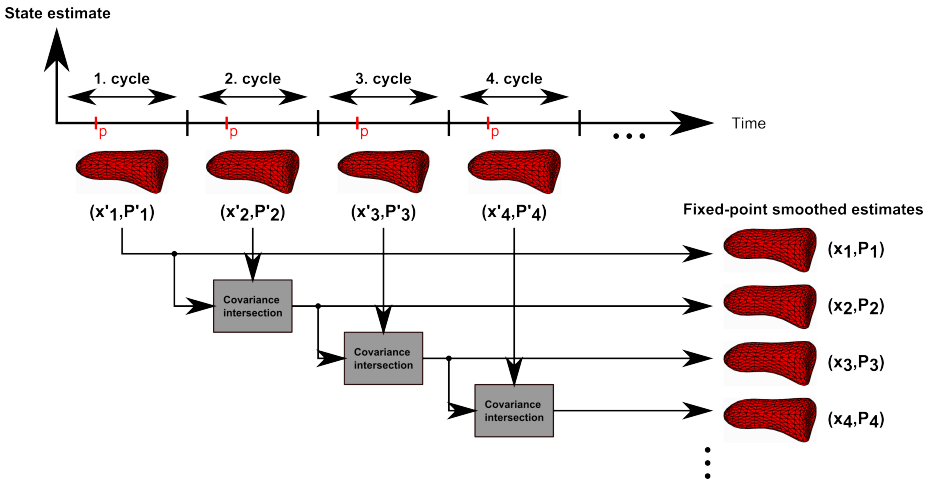


Figure 9: Illustration of the fixed-point Kalman smoother algorithm. The smoothed contour estimate for cycle position p, iteration k is obtained by fusing forward Kalman filter estimate for position p, iteration k with the smoothed estimate for position p, iteration $k-1$. Fusion is performed using the covariance intersection algorithm which assumes unknown correlation.

Figure 9 shows how a fixed-point smoothed estimate for the contour at position p in the heart-cycle is obtained. To obtain fixed-point smoothed estimates for all positions in the heart-cycle, the smoothing algorithm is repeated for each p of interest.

Since each contour estimate is not independent of the others, the unknown-correlation version of the covariance intersection algorithm introduced in section 2.7.2 is used to perform the covariance fusion.

3.2 Fixed-lag smoother

The fixed-lag smoother operates in two distinct phases: filling up frames and smoothing. In the first case not enough frames have arrived to do the backward pass, while the second case is normal operation; one input frame results in one state estimate.

Of the different smoothing algorithms, the fixed-lag smoother is the one most resembling a filter, the main difference being a time-lag between input and output. This represents two challenges: the initialization where no smoothed estimates can be returned and keeping track of the time stamp for which contour estimates applies.

The first problem is only transient and in this project is automatically solved by the two warm-up cycles, which guarantees that the frame buffer is filled for the lag-values used (either 3 or 16). The synchronization is performed internally and the details are invisible from the point of view of figure 15.

3.2.1 No prior knowledge on the backward pass

According to the theory in section 2.4.2 the backward pass must be initialized with an infinite covariance matrix in order to not use any prior information. This is in general a complicating factor since infinity is not a valid floating point number. Because of RCTL's choice of performing the edge-detections relative to the predicted contour (section 2.6.2), the backward pass is *not* initialized with an infinite covariance matrix. Instead, the same initial state and covariance is used as for the forward pass. Other alternatives are also investigated and will be discussed below.

The adverse effect of performing backward passes with no prior knowledge (in practice achieved using an identity matrix multiplied with a large scalar) within the RCTL framework is demonstrated in figure 20. No prior knowledge makes the filter overly confident in the noisy edge-detection results, and severely deformed estimates may in the worst case be the result.

3.2.2 Variations on the algorithm

In addition to using the initial state and covariance on the backward pass, support for not resetting the state on the backward pass was implemented. Referring to figure 10, filtering is then performed in the forward direction up to a certain point followed by filtering frames in the backward direction, without resetting the state when the direction is reversed.

According to the forward-backward approach to Kalman smoothing, a final backward prediction should be used at the time step k where the forward and backward passes meet to avoid using measurement number k twice. But since the state-transition model used in RCTL is just a simple regularization, a final *update* step is used in the implementation.

In summary, the following fixed-lag smoother algorithms were implemented. They will subsequently be referred to by these names.

- **Fixed-lag L smoother**: Adjustable lag-value L, computes backward estimates with a final update step before fusing forward and backward estimates. The backward pass is initialized with the same initial state and covariance as for the forward filter.

- **Fixed-lag L smoother, no backward reset**: Does not reset the state to initial state and covariance prior to backward pass. Instead the forward estimate is used directly. On the backward pass, only updated estimates are computed and subsequently fused with the forward estimates.

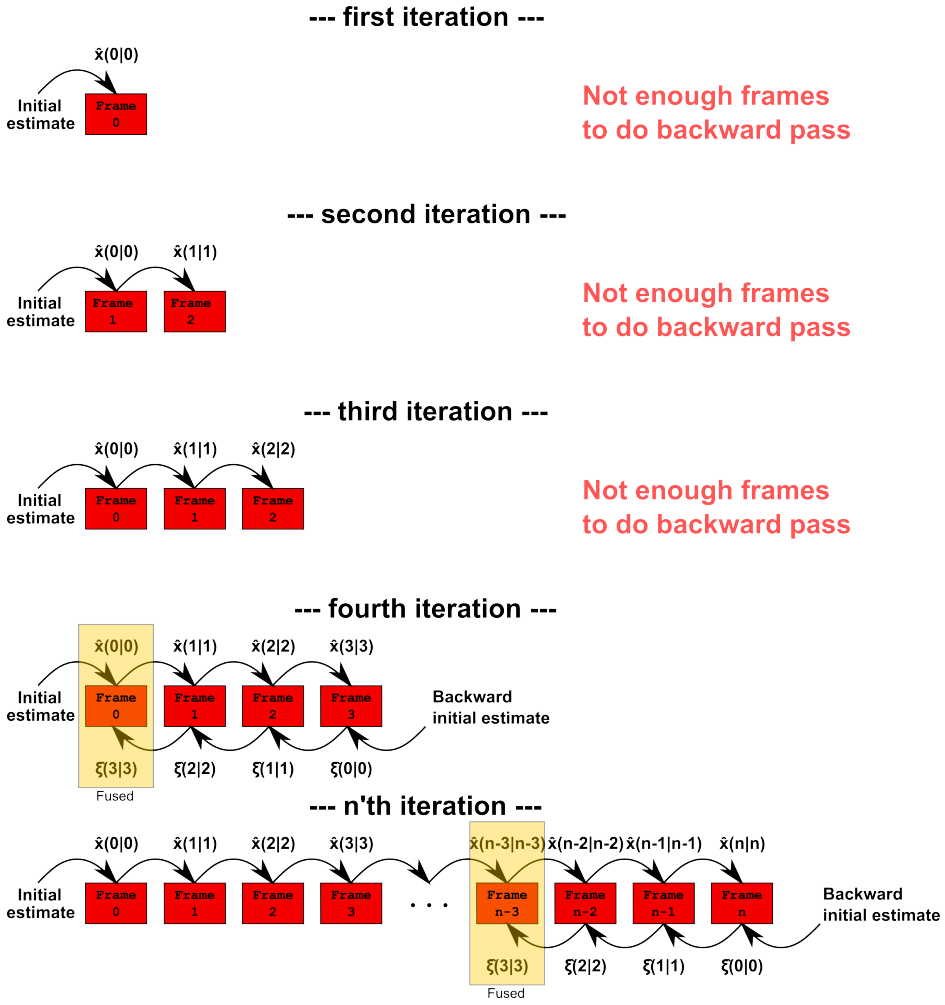


Figure 10: Illustration of the implementation of the fixed-lag Kalman smoother, in this case the fixed lag is three. After a sufficient number of frames (lag + 1) has been received, the smoother can do the backward pass and hence compute output values.

3.3 Fixed-interval smoother

Figure 11 illustrates how the fixed-interval smoother was implemented. The forward and backward estimates are obtained through the RctlMatlab interface.

Support is implemented for fusing the forward posterior with the backward predictions (as stated in the literature) or with the backward updated estimates (better performance observed). The fusion is in either case done with the covariance intersection algorithm assuming independent estimates, which is discussed in more details in section 2.7.1.

In summary, the following fixed-interval algorithms are implemented and will be referred to by these names.

- **Fixed-interval smoother w/final update**
- **Fixed-interval smoother w/final prediction**

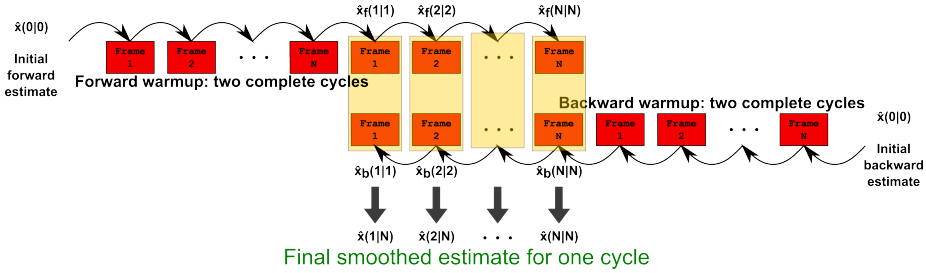


Figure 11: Illustration of the fixed-interval smoother algorithm. Two warm-up cycles are performed in both directions and the forward and backward estimates for a complete cycle in the interior is fused to produce the smoothed estimates.

3.4 The statistical smoother

The statistical smoother uses covariance intersection to fuse learned information about LV deformation (from ground-truth segmentations) with the regular forward Kalman filter estimates.

Although superficially similar, this method is not the same as principal component analysis (PCA), where the most prominent deformation directions are computed and used to define the allowable model deformation space. The statistical smoother approach can instead be thought of as an averaging taking estimation certainty into account. The goal is that the learned states can be used to fill in parts where the filtered estimate has high uncertainty.

The covariance intersection algorithm requires information in the form of state vectors and covariance matrices. Thus, for all ultrasound recordings, the ground-truth triangle meshes were converted to RCTL state vectors by following the procedure in appendix B.

The learned state vectors and covariance matrices are obtained from the state-vector version of the ground-truth meshes in the following way. After obtaining the state-vector description, the frame number corresponding to end-diastole is obtained from the original ground-truth data. Then the knowledge of the total number of frames and the fact that each recording contains exactly one heart-cycle is used to define a mapping from $[0, 1]$ to the frames, where 0 corresponds to the first frame after end-diastole (first frame of systole) and 1 corresponds to end-diastole. This involves wrapping around the frame index.

The normalized cycle position interval $[0, 1]$ is then split up into $N = 100$ equidistant points and linear interpolation of each of the 29 vectors of ground-truth state vectors is utilized, which enables the computation of average state vectors and covariance matrices at each of the N points in $[0, 1]$ using the sample average and sample covariance, respectively.

3.4.1 Algorithmic details

Figure 12 shows the principle of operation of the statistical smoother. The statistical smoother requires knowledge of the heart-cycle position of the current Kalman filter estimate. During testing, this knowledge was obtained from the ground-truth data files, but in a real-life situation e.g. the volume of the estimated contour can be used to obtain this information, by exploiting that the volume is normally largest at end-diastole and smallest at end-systole.

Since the learned covariance matrix for position p is obtained as the sample covariance matrix from all state-vectors describing the state at position p, it is

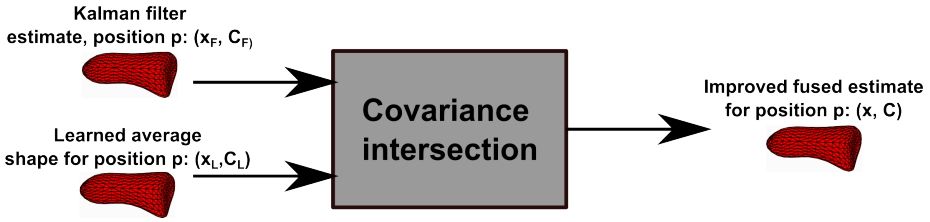


Figure 12: Operation of the statistical smoother. Learned average state vectors computed from ground-truth meshes of a training set is combined with the regular EKF state estimate via the covariance intersection algorithm to produce a possibly improved state estimate.

scaled very differently from the covariance matrix associated with the Kalman filter state vector estimate. Hence a *covariance multiplier* is introduced as a parameter, which can be used to tune how much confidence that is placed on the learned state estimate. The implementation uses a covariance multiplier of 375000 times the sample covariance.

As explained in section 2.6, the deformation model consists of a global part (translation, rotation, and scaling) and a local part (deformation of the shape template). These parameters are concatenated to form the full state vector.

However, only the local deformation part of the ground-truth state vectors are fused with the regular Kalman filter estimate, since this is the part containing information about average left-ventricular deformation. The orientation and location of the left ventricle in the ultrasound volume is a function of the probe placement used when recording, and can only be estimated by the Kalman filter.

Hence, after fusing the interpolated learned state estimate, the subrange of the state vector and covariance matrix describing the global parameters are copied back from the Kalman filter estimate.

4 Method

This section describes the smoother performance evaluation and comparison to the extended Kalman filter based tracker. A set of 29 three-dimensional ultrasound recordings of a single heart-cycle, each with a manual ground-truth segmentation of the left-ventricle, is used as input. Each algorithm (including the reference EKF) produces one triangle mesh estimate for each frame in the recordings. Mean absolute surface error and mean absolute volume error is computed by comparison with the the ground-truth segmentations. The evaluation is done with two different configurations of the underlying EKF-tracker. The results will be presented as tables, averaged over all ultrasound recordings, and as histograms comparing the performance of the different smoother types with EKF at selected positions in the heart-cycle.

The real-time contour tracking library has functionality for exporting triangle mesh-representations of the Doo-Sabin surface used internally when tracking. Existing Matlab scripts are used for the computation of volume and surface agreement.

The set of recordings consists of single heart-cycles from 10 healthy subjects and 19 subjects with recent first time myocardial infarction, recorded using a Vivid 7 (26 recordings) or a Vivid E9 (3 recordings) ultrasound scanner (GE Vingmed Ultrasound, Norway) with a matrix array (3V) transducer. Each recording is in DICOM format and the number of frames in each recording is 27.5 ± 5.7 (mean $\pm$ SD). Recordings are looped in order to present the various estimation algorithms with multiple heart-cycles.

4.1 The deformable LV model

Figure 13 shows the deformable Doo-Sabin that was used in this project to segment the left ventricle. Also shown is the control point grid and edge-detectors with search normals. The model is in its initial state with no control point normal displacements.

The model has 20 control points, of which all but two are allowed to move freely along surface normals of the initial model. The base (center point at the bottom) and the apex (the tip) is required to be stationary. The number of edge-detection points spread out evenly over the surface is 500. The edge-detections are performed using edge-detectors of type ”step” and 30 samples along the search line, with a spacing of 1 mm.

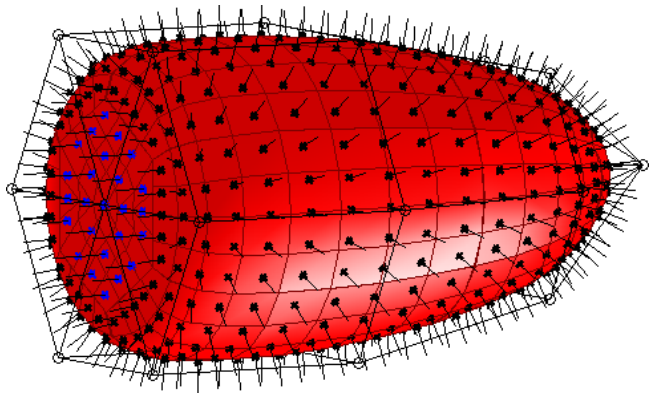


Figure 13: The deformable Doo-Sabin model used when tracking the left vectricle. It is shown in its initial state, i.e. the state where all control node normal displacements are zero. The flat part to the left is the base and the tip at the right is the apex.

4.2 The manual ground-truths

The endocardial left-ventricle segmentation of the recordings was performed by a trained medical doctor using a semi-automatic tool (4D AutoLvq, GE Vingmed Ultrasound, Norway). Each ground-truth is in the form of a .mat file (Matlab data file) containing

- 'edFrame' : Frame number of end diastole.
- 'esFrame' : Frame number of end systole.
- 'filename' : Name of corresponding DICOM recording.
- 'mesh' : Data structure containing the correct segmentation in the form of one triangle mesh for each frame.

Figure 14 shows an example of a manual segmentation of the left ventricle.

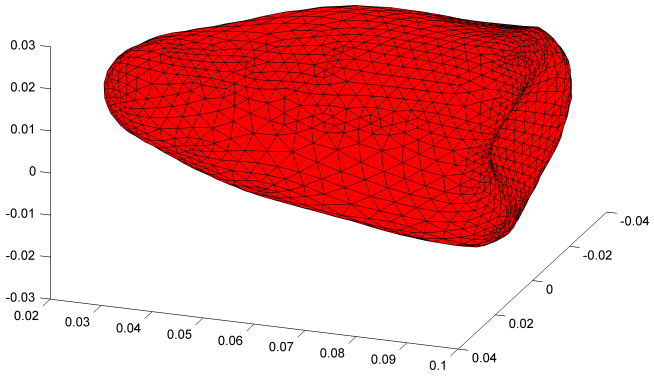


Figure 14: Rendered triangle mesh description of the correct left ventricle segmentation of the first frame in recording **A38**.

4.3 Evaluation and comparison of the estimation algorithms

The evaluation consists of three separate parts: Computing the mean estimation errors for all algorithms, computing the mean agreement between the various smoothers and EKF on a recording-by-recording basis, and visualizing the estimation errors for each algorithm at four points in the heart-cycle, in the form

of histograms. The flexible algorithm evaluation framework is illustrated in figure 15.

The average *mean absolute surface error* and *mean absolute volume error* over all recordings is computed for the EKF and all smoother algorithms, i.e. for a recording with N frames a total of N mean absolute surface errors and N mean absolute volume errors are computed for a given algorithm. These are averaged, the process repeated for the 29 recordings and finally averaged over all recordings. This results in one single mean absolute surface error value and mean absolute volume error value for each estimation algorithm. The results are summarized in section 5.1.1, with one table for each of the two configuration files.

The performance comparison between EKF and the smoothers are performed by computing the average mean absolute surface error and mean absolute volume errors over all frames for each algorithm. This results in two vectors of length 29 for each algorithm. The vector corresponding to EKF is then subtracted from each of the others (which corresponds to smoothers). The average error difference from EKF is obtained by computing the sample mean and standard deviations of the difference vectors. These results are summarized in section 5.1.2, with one table for each of the two configuration files.

The smoothing algorithms in the following list were compared to the regular extended Kalman filter-based tracker:

- Fixed-interval smoother using final update steps (experimental)
- Fixed-interval smoother using final prediction steps (textbook solution)
- Fixed-lag smoother with lag 3
- Fixed-lag smoother with lag 16
- Fixed-lag smoother with lag 16 and no reset on backward pass (experimental)
- Fixed-point smoother operating on 10 consecutive heart-cycles.
- Fixed-point smoother operating on 100 consecutive heart-cycles.
- The statistical smoother

In addition to these, the closest Doo-Sabin state vector representation of ground-truth meshes (explained in appendix B) was also implemented in the

estimation algorithm framework of figure 15 to obtain bounds on the lowest possible estimation errors.

The evaluation script measures average absolute distance between the estimated surface and the ground truth surface and the absolute volume estimation error in percent relative to the ground-truth volume. Existing functionality for computation of average point to surface distance and volume of triangle meshes is reused.

Triangle meshes are exported from RCTL in order to use the evaluation scripts which requires input in the form of triangle meshes. This triangulation of the smooth Doo-Sabin surface used internally by RCTL is not believed to introduce significant errors.

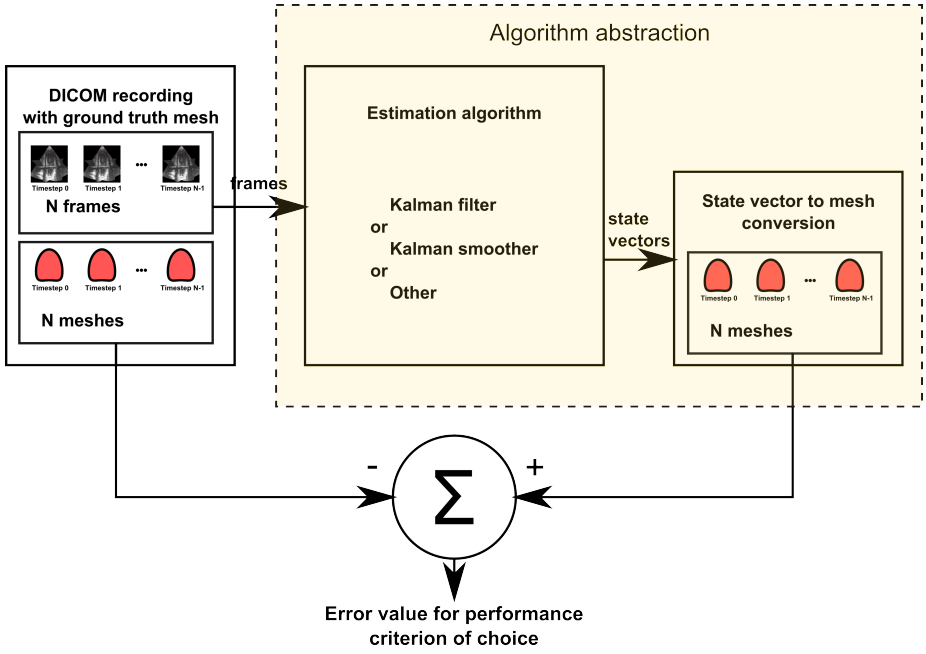


Figure 15: Tracking algorithm abstraction employed for comparison of Kalman filtering and Kalman smoothing algorithms.

4.4 Warmup cycles

The extended Kalman filter must track a certain number frames in order to be properly initialized. This is referred to as warm-up, and can be characterized as the period where the deformable contour is stretching out or contracting from the initial shape towards the correct contour position for the first time. Once this is done, the filter responds correctly to changes in the image volume and the tracker returns meaningful estimates. When evaluating, the filter is operated for two complete cycles before any contour estimates are used, also for the backward tracking pass in the fixed-interval smoother.

4.5 Visualizing the performance at specific positions in the heart cycle

Average error measures for all algorithms were computed for the heart-cycle positions 0.0, 0.5, 1.0, and 1.5, where 0.0 corresponds to end-diastole (ED) and 1.0 corresponds to end-systole (ES). The results for each smoother algorithm is compared with the EKF in a histogram. Figure 16 illustrates the heart cycle position defined as a real number with respect to ED and ES. Note that it is possible to extend beyond 1.0 (ES).

4.6 Two different configuration files

Two different .xml configuration files for the extended Kalman filter have been used when evaluating algorithm performance. Since the smoother algorithms are implemented on top of the EKF-tracker, its configuration applies for the smoothers as well.

The file **STEP_test.xml** is tuned for good filtering performance in normal filtering operation. The difference between the two configuration files is that **STEP_test.xml** use the default auto-scaling of the reported measurement noises and **NoiseScalingDisabled.xml** does not. The noise parameters in **NoiseScalingDisabled.xml** was manually re-tuned to match the tracking behavior of **STEP_test.xml**.

4.7 Leave-one out cross-validation

The performance of the statistical smoother was evaluated using leave-one out cross-validation. The principle is to evaluate on one recording and use the rest as training data. This is repeated for each of the 29 recordings, and the results

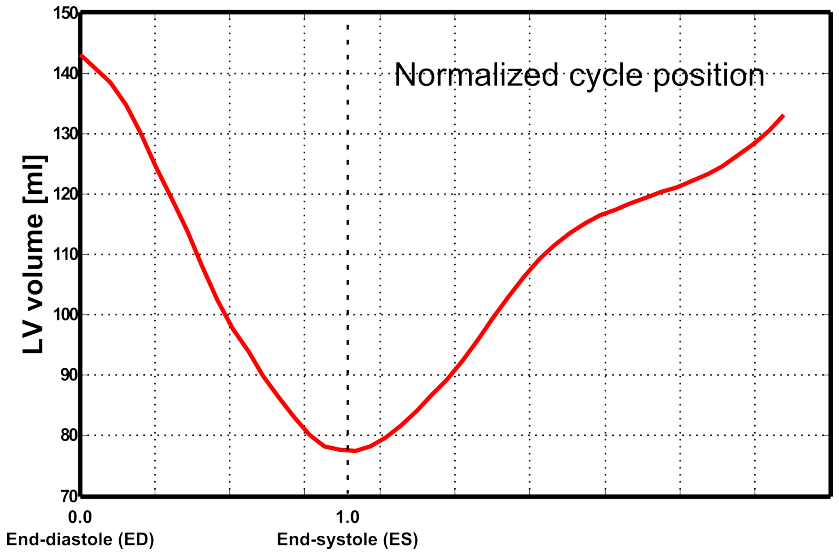


Figure 16: The cycle position definition as a real number, where 0.0 corresponds to end-diastole and 1.0 corresponds to end-systole.

are averaged, in order to measure how well the learned results generalize to independent cases.

5 Results

In order to facilitate direct comparison of algorithm performances, each criterion (mean absolute surface error and mean absolute volume error) use the same scale on the y-axis in all histograms.

5.1 Summary of performance comparisons

5.1.1 Performance of all algorithms

These tables show average mean absolute surface error and average mean absolute volume for each estimation algorithm (including EKF), averaged over all 29 recordings. Units are millimeters for surface errors and percentages of the ground truth volume for volume errors.

	Mean absolute surface error [mm] (mean ± SD)	Mean absolute volume error [% of ground-truth] (mean ± SD)
Extended Kalman filter	2.05 ± 0.68	11.28 ± 8.77
Fixed-interval smoother w/final update	2.05 ± 0.70	11.18 ± 8.42
Fixed-interval smoother w/final prediction	2.09 ± 0.73	11.73 ± 9.11
Fixed-lag 3 smoother	2.27 ± 0.87	13.67 ± 9.27
Fixed-lag 16 smoother	2.10 ± 0.70	11.48 ± 7.94
Fixed-lag 16 smoother, no reset on backward	2.05 ± 0.70	11.19 ± 8.53
Fixed-point smoother 10 iterations	2.07 ± 0.71	11.60 ± 9.64
Fixed-point smoother 100 iterations	2.07 ± 0.71	11.60 ± 9.49
Statistical smoother	2.04 ± 0.70	11.42 ± 8.74
Closest Doo-Sabin representation of ground-truth mesh	1.19 ± 0.13	11.22 ± 1.08

Table 2: STEP_test.xml

	Mean absolute surface error [mm] (mean ± SD)	Mean absolute volume errors [% of ground-truth](mean ± SD)
Extended Kalman filter	2.12 ± 0.60	12.17 ± 9.75
Fixed-interval smoother w/final update	2.17 ± 0.69	12.73 ± 9.45
Fixed-interval smoother w/final prediction	2.13 ± 0.62	12.80 ± 9.50
Fixed-lag 3 smoother	2.25 ± 0.72	13.51 ± 9.18
Fixed-lag 16 smoother	2.18 ± 0.64	12.32 ± 8.43
Fixed-lag 16 smoother, no reset on backward	2.13 ± 0.66	12.10 ± 9.44
Fixed-point smoother 10 iterations	2.11 ± 0.60	12.16 ± 9.66
Fixed-point smoother 100 iterations	2.11 ± 0.59	12.18 ± 9.71
Statistical smoother	2.12 ± 0.60	12.26 ± 9.82
Closest Doo-Sabin representation of ground-truth mesh	1.19 ± 0.13	11.22 ± 1.08

Table 3: NoiseScalingDisabled.xml

5.1.2 Estimation error relative to EKF

These tables show average and standard deviation of the difference between EKF error and smoother error on a recording-by-recording basis, for both mean absolute surface error and mean absolute volume error. Units are millimeters for surface errors and percentages of the ground truth volume for volume errors.

	Mean difference from EKF in mean absolute surface error [mm] (mean ± SD)	Mean difference from EKF in mean absolute volume error [% of ground-truth] (mean ± SD)
Fixed-interval smoother w/final update	0.00 ± 0.04	-0.01 ± 0.65
Fixed-interval smoother w/final prediction	-0.03 ± 0.10	1.79 ± 0.97
Fixed-lag 3 smoother	-0.22 ± 0.44	5.20 ± 5.35
Fixed-lag 16 smoother	-0.05 ± 0.14	0.65 ± 2.39
Fixed-lag 16 smoother, no reset on backward	-0.00 ± 0.04	-0.09 ± 0.58
Fixed-point smoother 10 iterations	-0.02 ± 0.12	-0.24 ± 1.82
Fixed-point smoother 100 iterations	-0.02 ± 0.12	-0.25 ± 1.77
Statistical smoother	0.01 ± 0.06	-0.03 ± 0.69

Table 4: STEP_test.xml

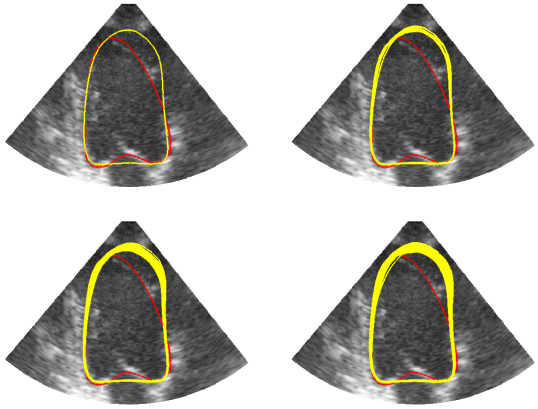
	Mean difference from EKF in mean absolute surface error [mm] (mean ± SD)	Mean difference from EKF in mean absolute volume errors [% of ground-truth](mean ± SD)
Fixed-interval smoother w/final update	-0.05 ± 0.28	-1.15 ± 4.18
Fixed-interval smoother w/final prediction	-0.01 ± 0.09	1.23 ± 1.88
Fixed-lag 3 smoother	-0.13 ± 0.45	3.34 ± 6.22
Fixed-lag 16 smoother	-0.05 ± 0.21	-0.06 ± 3.41
Fixed-lag 16 smoother, no reset on backward	-0.00 ± 0.09	-0.29 ± 1.12
Fixed-point smoother 10 iterations	0.01 ± 0.04	0.03 ± 0.63
Fixed-point smoother 100 iterations	0.01 ± 0.03	0.04 ± 0.57
Statistical smoother	0.00 ± 0.02	0.10 ± 0.24

Table 5: NoiseScalingDisabled.xml

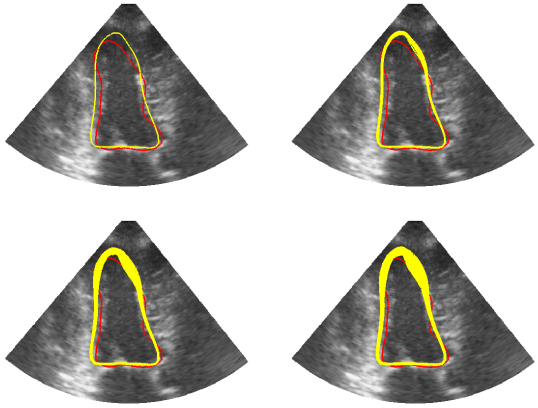
5.2 The fixed-point smoother

5.2.1 Visualization of contour estimate spread

This section contains slices showing the tracked contour (with regular EKF) at the same heart-cycle position, but at several consecutive cycles. They are plotted on top of each other, so that thick regions corresponds to areas of great variability.



(a) Ground-truth (red) with 1, 10, 100, and 1000 estimates (yellow) for ED frame (recording **A38.dcm**)

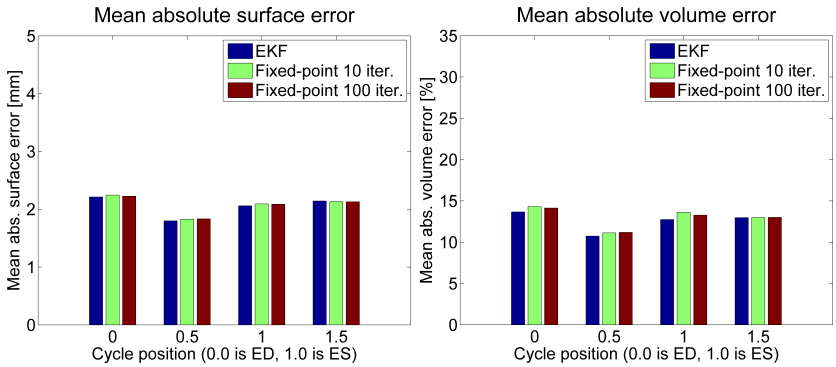


(b) Ground-truth (red) with 1, 10, 100, and 1000 estimates (yellow) for ES frame (recording **A38.dcm**)

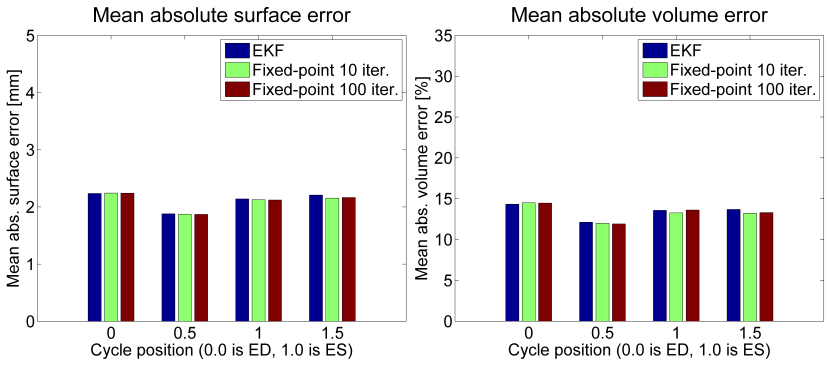
Figure 17: Multiple superpositioned slices of estimated contours for ED frame and ES frame, shown with the correct segmentation (red)

5.2.2 Evaluation at selected heart-cycle positions

This section contains histograms comparing the fixed-point smoother with EKF at four positions in the heart-cycle, using two different configuration files.



(a) Using the **STEP_test.xml** configuration file.



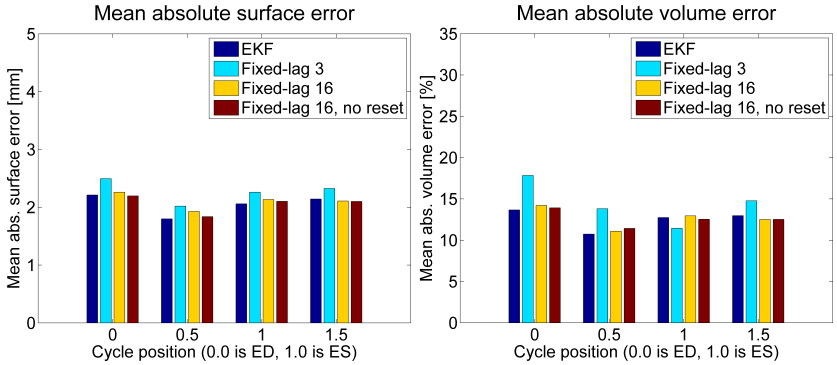
(b) Using the **NoiseScalingDisabled.xml** configuration file.

Figure 18: Performance evaluation of fixed-point smoothing using 10 or 100 iterations compared to the regular extended Kalman filter based tracker. The experiment is run using two different configurations of the underlying Kalman filter. For each row: left is *mean absolute surface error* and right is *absolute volume error* when comparing to the ground-truth segmentation.

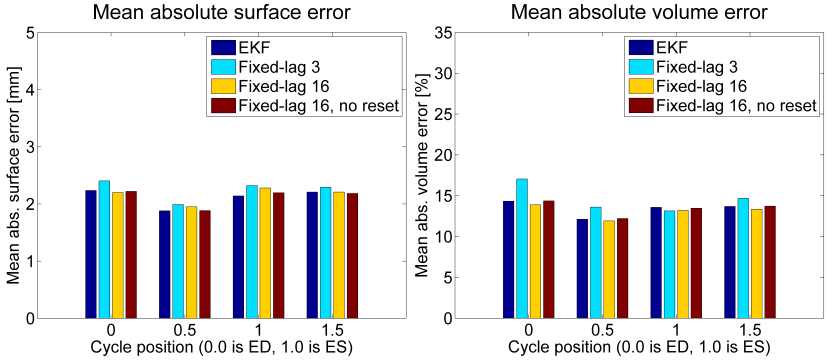
5.3 The fixed-lag smoother

5.3.1 Evaluation at selected heart-cycle positions

This section contains histograms comparing the fixed-lag smoother with EKF at four positions in the heart-cycle, using two different configuration files.



(a) Using the **STEP_test.xml** configuration file.

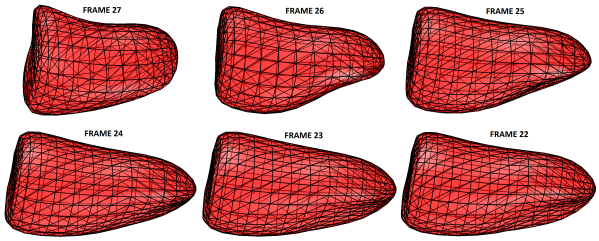


(b) Using the **NoiseScalingDisabled.xml** configuration file.

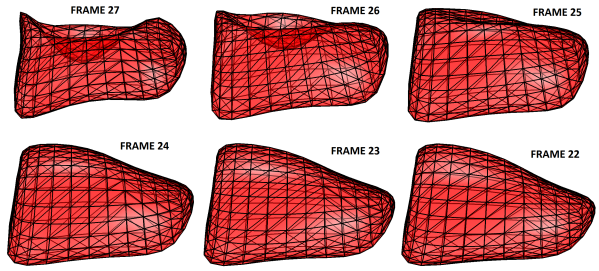
Figure 19: Performance evaluation of different fixed-lag smoother solutions compared to the regular extended Kalman filter based tracker. Experiment is run using two different configurations of the underlying Kalman filter. For each row: left is *mean absolute surface error* and right is *absolute volume error* when comparing to the ground-truth segmentation.

5.3.2 Deformed shapes from the backward pass

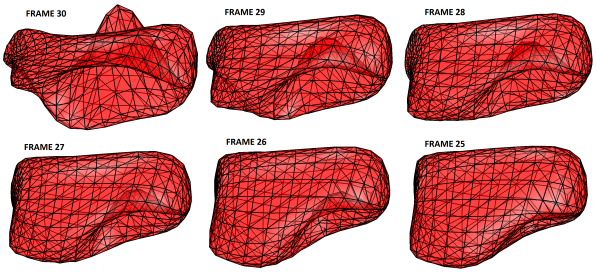
This section contains deformed contour estimates from the first few steps in the backward pass when the backward filter is initialized with a large covariance matrix $P = 1.0 \cdot 10^{10}\mathbf{I}$ where $\mathbf{I}$ is the identity matrix. This represents a practical implementation of the ”no prior knowledge” discussed in section 2.4.1.



(a) Illustration of a recording for which the backward pass deformation severity is low



(b) Illustration of a recording for which the backward pass deformation severity is medium

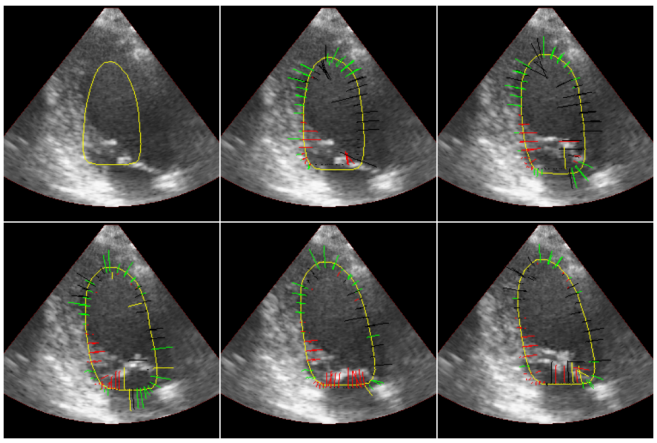


(c) Illustration of a recording for which the backward pass deformation severity is high

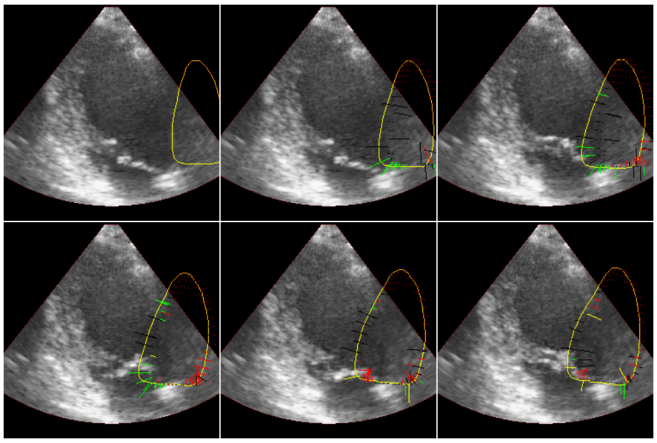
Figure 20: Renderings of the first six LV contour estimates on the backward pass with varying degree of deformation severity. Initial state estimate is the mean shape from .xml config and covariance matrix is $1.0 \cdot 10^{10}\mathbf{I}$, where $\mathbf{I}$ is the identity matrix.

5.3.3 Slices from the backward pass

This section visualizes the effect of the initial contour estimate when performing the backward tracking. For both the initial position from the configuration file and a translated version of it, slices of the first five tracking results are shown. The edge-detection result types are also shown encoded by colors.



(a) The initial contour is as specified in the configuration file **STEP_test.xml** and close to the true location of the contour.



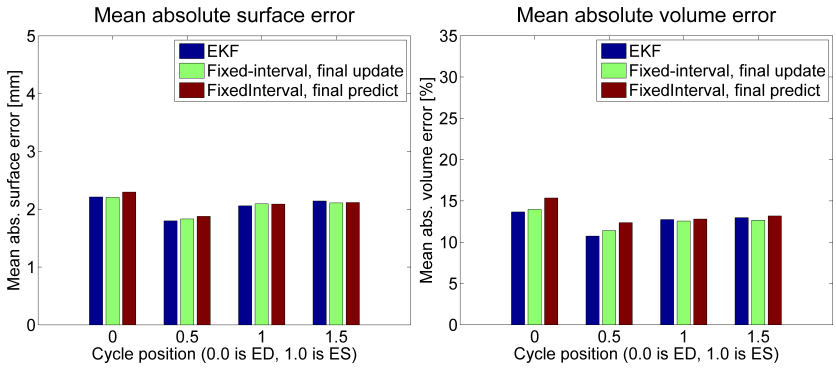
(b) The initial contour is a translated version of that specified in the configuration file **STEP_test.xml** which is far away from the true location of the contour.

Figure 21: The initial contour estimate and the five next backward tracking results with visualization of the edge-detection results, when using different initial contour estimates. Green and red lines are outward and inward detected normal displacements, respectively. Black lines represents no results and yellow discarded results.

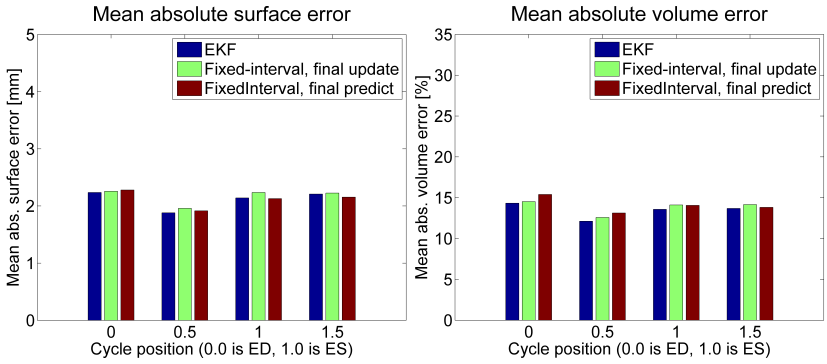
5.4 The fixed-interval smoother

5.4.1 Evaluation at selected heart-cycle positions

This section contains histograms comparing the fixed-interval smoother with EKF at four positions in the heart-cycle, using two different configuration files.



(a) Using the **STEP_test.xml** configuration file.



(b) Using the **NoiseScalingDisabled.xml** configuration file.

Figure 22: Performance evaluation of different fixed-interval smoother solutions compared to the regular extended Kalman filter based tracker. Experiment is run using two different configurations of the underlying Kalman filter. For each row: left is *mean absolute surface error* and right is *absolute volume error* when comparing to the ground-truth segmentation.

5.4.2 Visualization of operation by orthogonal slices

This section contains selected results from a visualization of all tracking results for the fixed-interval smoother. The figures shows orthogonal plane slices through ultrasound data, forward tracked contour (yellow), backward tracked contour (red), fused contour (green), and the ground-truth contour (blue). Configuration files used for forward and backward tracking is **STEP_test.xml**.

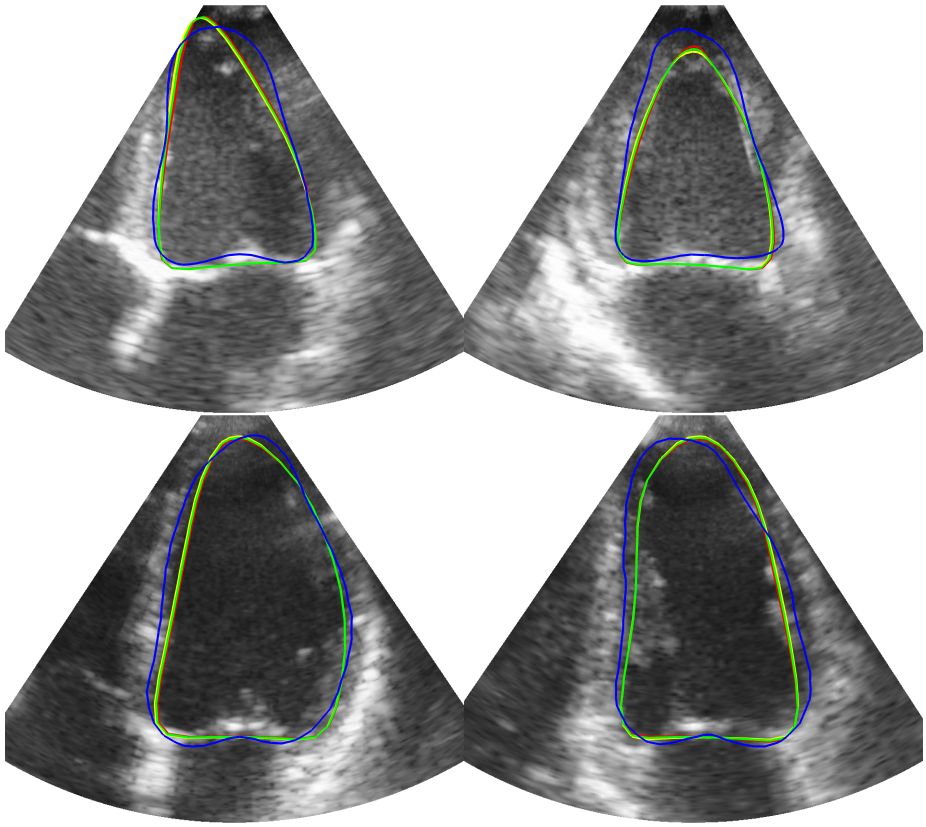


Figure 23: Detailed visualization of the fixed-interval algorithm in operation. Two orthogonal slices through two different ultrasound recordings showing *very close* agreement between the forward (*yellow*), backward (*red*), and fused (*green*) estimates are shown. The ground truth segmentation is also shown (*blue*). This close agreement is representative for the majority of all frames in the 29 ultrasound recordings.

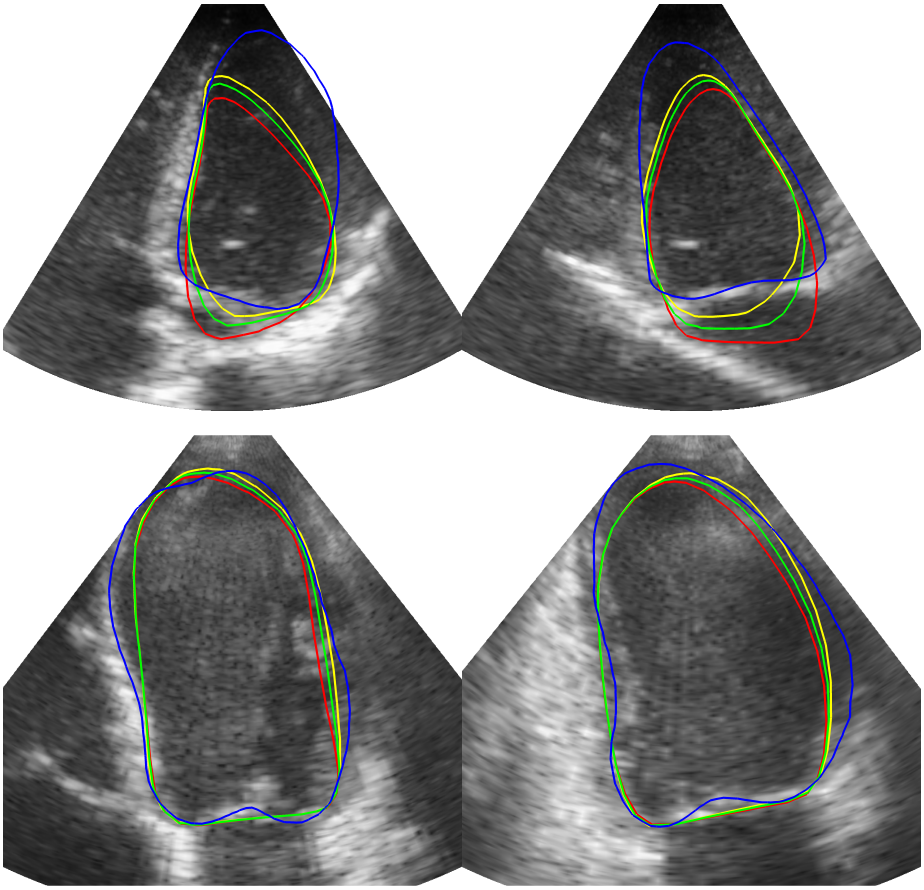
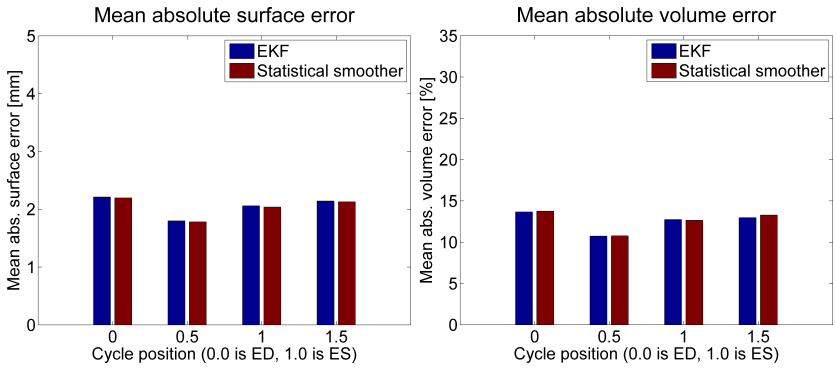


Figure 24: Detailed visualization of the fixed-interval algorithm in operation. Two orthogonal slices through two different ultrasound recordings showing representative examples of the *largest* differences between the forward (*yellow*), backward (*red*), and fused (*green*) estimates. The ground truth segmentation is shown in *blue*.

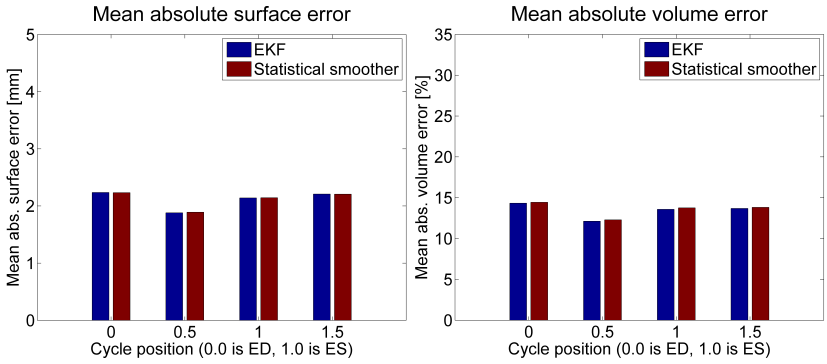
5.5 Statistical smoother

5.5.1 Evaluation at selected heart-cycle positions

This section contains histograms comparing the statistical smoother with EKF at four positions in the heart-cycle, using two different configuration files. Leave-one-out cross-validation is used.



(a) Using the **STEP_test.xml** configuration file.



(b) Using the **NoiseScalingDisabled.xml** configuration file.

Figure 25: Performance evaluation of the statistical smoother compared to the regular extended Kalman filter based tracker. Experiment is run using two different configurations of the underlying Kalman filter. For each row: left is *mean absolute surface error* and right is *absolute volume error* when comparing to the ground-truth segmentation.

5.5.2 Investigation of ground-truth volume curves

Figure 26 shows a plot of all 29 volume curves computed from the ground-truth data files together with the volume curve associated with the mean states (red). They are aligned and normalized as described in section 3.4.

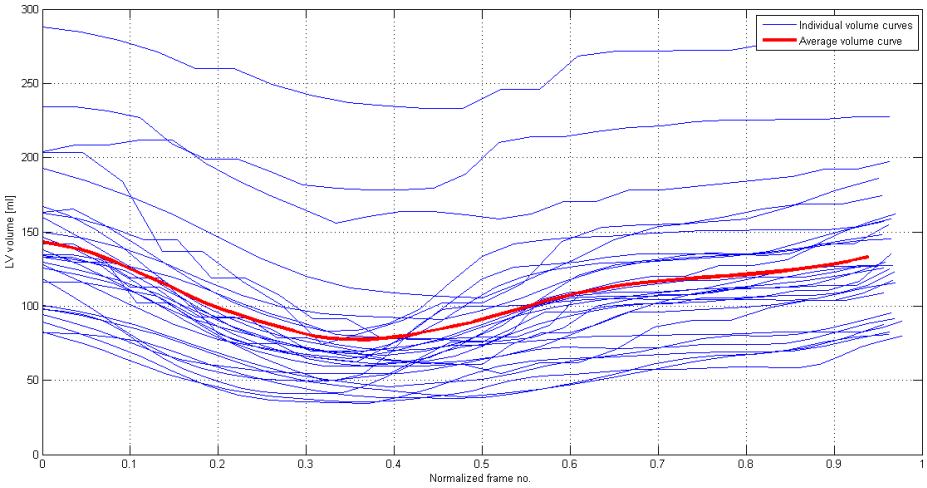


Figure 26: Normalized and aligned volume curves from each individual ground-truth data file (blue) and the average volume curve (red).

5.6 Best representation of the ground-truths as Doo-Sabin deformable LV models

This section contains the results of treating the Doo-Sabin representation of the ground truth meshes as an evaluation algorithm in its own right. The purpose is to investigate the theoretical lower error limits for the specific deformable Doo-Sabin LV model used in this project.

5.6.1 Evaluation at selected heart-cycle positions

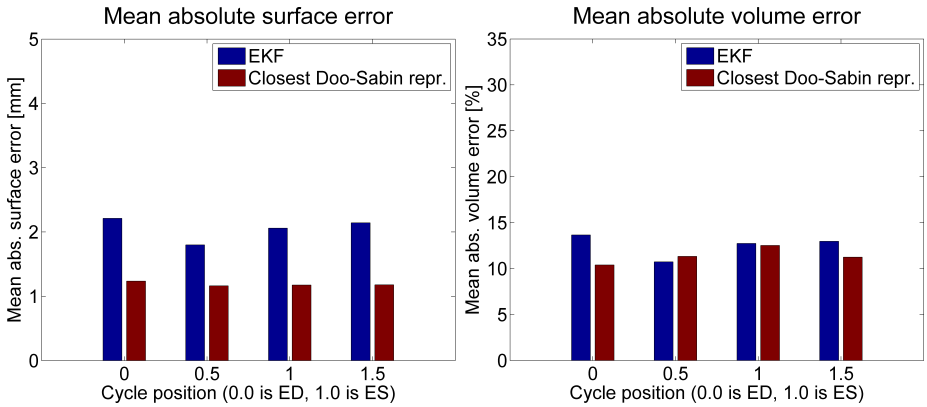
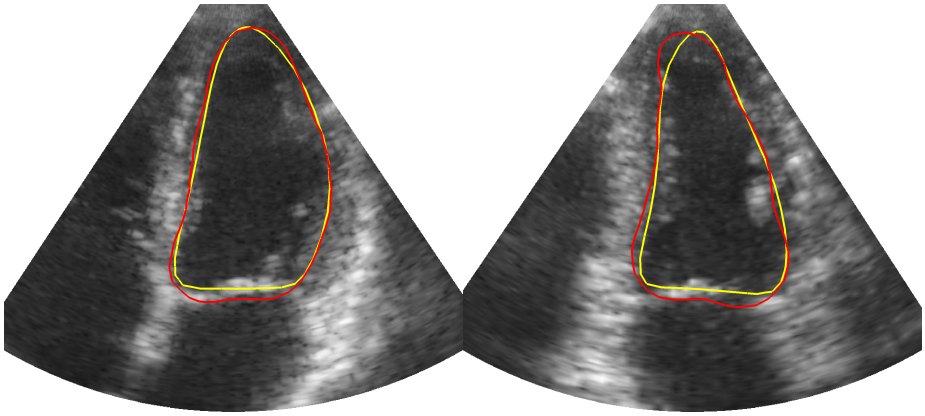


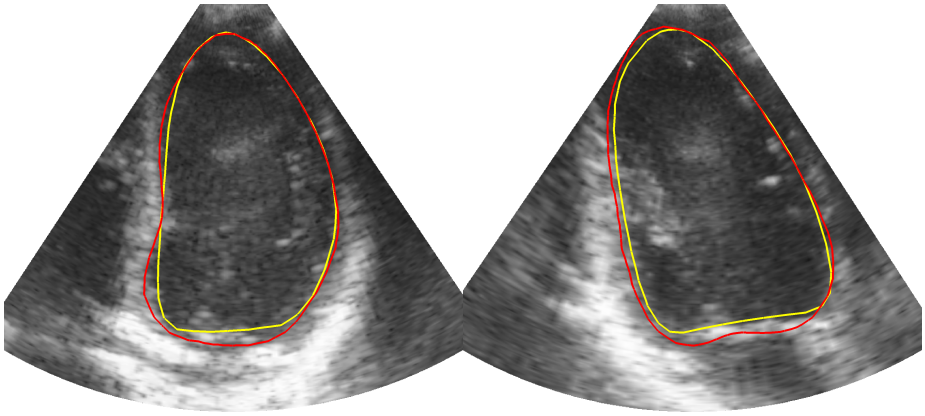
Figure 27: Comparison of the EKF estimation errors and the errors introduced when representing the ground-truths as Doo-Sabin left ventricle models. The configuration file used is **STEP_test.xml**.

5.6.2 Slice illustrations

This section contains orthogonal slices through ultrasound data, ground-truth mesh and closest Doo-Sabin model for two frames from different recordings, illustrating that the ground-truth meshes often have details not representable by the Doo-Sabin model.



(a) **p10 - frame 14**



(b) **p8 - frame 4**

Figure 28: Two orthogonal slices of ultrasound recordings with ground-truth segmentation (red) and the surface defined by the closest Doo-Sabin state vector (yellow).

6 Discussion

General discussion are presented first; later sections address technical details.

The tables in section 5.1 show that differences in the mean absolute surface error between EKF and the smoothers are in the order of fractions of a millimeter, with the exception of the fixed-lag 3 smoother. The smoothing algorithms do not offer any significant reduction in estimation errors. The fixed-lag 3 smoother even degrades the performance. This will be discussed in more details below.

Most emphasis is on the absolute surface error, since it is shown in figure 27 that even the closest representation of the ground-truth meshes as left ventricle Doo-Sabin models introduce volume errors close to what is introduced by the extended Kalman filter. The absolute surface error is less affected.

Figure 21 explains why the fixed-lag 3 smoother introduces greater surface and volume errors than EKF. As noted in section 3.2.1, the initial state on the backward pass is as loaded from the configuration. This typically places the contour in the middle of the left ventricle in the recording, as shown in figure 21a. The same figure shows that in the first three frames, the backward filter is still stretching out towards the left ventricle, and hence does not produce meaningful estimates of the location of the left ventricle. Therefore the tracking accuracy is decreased when those backward estimates are fused with forward estimates. Figure 19 shows that in the case of a fixed-lag of 16, this problem is reduced, most likely because the backward filter has had enough frames to lock on to the left ventricle.

This performance degradation problem is avoided with the no-reset fixed-lag smoother, as shown in figure 19. This is because the last forward result is used as initial estimate for the backward pass, i.e. the backward filter starts in a locked-on state.

The lack of improvement with smoothing algorithms can be explained by the way edge-detections are performed in RCTL: along a finite search line in the noisy image and starting from points on the predicted contour. The first part implies that even in the absence of any anatomical edge in the image, the edge-detectors may decide that some part of the noise is the most likely candidate for an edge and return this point together with a measure of uncertainty, calculated from the edge strength. This uncertainty value is unlikely to reflect the true uncertainty, and may in some cases lead to the filter being overly confident that an edge exists.

Performing the edge-detections relative to the predicted contour necessitates an initial contour estimate close to the true contour location, otherwise the

search will be performed in the wrong part of the image volume, as illustrated in figure 21b. This again implies that the forward and backward filter will detect nearly the same edges, leading to very little performance improvement by smoothing. This is happening in figure 23, and is representative for the behavior of the fixed-interval smoother on the majority of the 29 recordings. The only situation in which an improvement is expected is if the backward filtered model attaches to a more correct position. The fused estimate will then be pulled in the right direction.

According to [5], the tracking quality is primarily limited by the the difficulty of edge-detection in ultrasound recordings because of the poor image quality, which fits well with the observed nonexistent performance gains by smoothing techniques.

6.1 Kalman smoothers

6.1.1 The fixed-point Kalman smoother

The fixed-point smoother has the drawback that it requires knowledge of the current position in the heart-cycle, since it combines multiple state vector estimates for the same position. In a real-time situation it might be necessary to estimate this position from e.g. the volume curve of the tracked contour. When the performance of the fixed-point smoother was evaluated in this project, it was possible to obtain the correct cycle-position from the ground-truth data.

Based on figure 17 it was anticipated that the fixed-point smoother would produce better tracking results since estimates of the same cycle position (frame number in the case of a single looped recording) at successive iterations varies in parts of the model, as is evident by the thick yellow parts.

These variations when operating on a periodic input recording can be explained by the measurement function being defined relative to the tracked contour. It is unlikely that the contour estimate is identical to the starting estimate after a complete heart-cycle is tracked. Therefore, the edge-detections will be performed starting at slightly different positions at the first image in the next cycle. This can introduce complicated non-periodicities.

6.1.2 Closest Doo-Sabin representation of the ground truth meshes

An important question is how well the left-ventricle model introduced in section 4.1 is able to represent the ground-truth meshes. Figure 27 shows the average estimation errors when the Doo-Sabin representation was considered an estimation algorithm in its own right.

The figure shows that the typical errors introduced, and therefore the lower limit on estimation errors in this project, are

- Absolute surface error: 1.25 mm
- Absolute volume error: between 10 % and 13 %

An important observation is that the extended Kalman filter tracker almost attains this lowest volume error limit.

Figure 28 shows representative examples of situations where the Doo-Sabin model does not have enough degrees of freedom to fit all the details of the manually segmented ground-truth mesh.

Based on manual inspection of such slices, it appears likely that the volume errors come from sharp bends in the ground-truth segmentation to which the Doo-Sabin model cannot adapt. A significant part of the volume errors also appears to come from the region around the mitral valve. A possible explanation is that this is caused by the deformable model having a non-movable control node there. In addition, this is the place where the model's circumference is largest, which amplifies the effect of deformations on the volume error.

6.1.3 Final update vs. final prediction in the forward-backward smoothers

This discussion is relevant for the smoother solutions based on the forward-backward formulation, i.e. the fixed-lag smoother and fixed-interval smoother.

Figure 22 shows the evaluation results when the EKF is compared to two versions of the fixed-interval Kalman smoother, one in which the final step on the backward pass is a prediction, and the other one where the final step is an update.

Even though a final prediction step should be used, according to the theory in section 2.4.1, the final-update solution was explored based on the intuition that since the dynamical model is a simple regularization relaxing the estimated contour towards an initial shape, the backward estimate would be made less correct by the regularization in frames where the left ventricle is in fact expanding.

The evaluation results in figure 22 indicates that the performance of the final-prediction fixed-interval smoother is most of the time slightly inferior to that of the final-update version, but the performance of both are very close to that of the EKF.

6.1.4 The effect of measurement noise auto-scaling

Disabling measurement noise auto-scaling does not make the smoothers perform better than the EKF, in contrast to what was anticipated.

As explained in section 2.6.8, auto-scaling of measurement noise values are by default enabled in RCTL. This has the effect that measurement variances are reported in such a way that they sum up to a user-definable value.

Auto-scaling introduces artificial certainty. Even if the measurements are done in a wrong part of the image, containing nothing but noise, the auto-scaling results in the filter interpreting the measurements to be roughly as reliable as measurements obtained from the correct anatomical position.

Given two estimates E_1 and E_2 that because of the measurement noise auto-scaling have almost equal covariance matrices, the covariance intersection will return approximately the average of the two estimates. In the case where E_1 and E_2 are independent, this would improve the accuracy. But, as mentioned earlier, in order to obtain estimates of the correct structure, the edge-detections must be performed close to the true location. This implies that the estimates E_1 and E_2 will not be independent, and that the averaging will have smaller effect.

The results when disabling noise-scaling is never as good as the results when using a Kalman filter tuned for good performance by itself, such as the **STEP_test.xml** configuration file.

6.1.5 Not using inverse dynamics

In [12] it is stated that the state-transition matrix used on the backward pass must be the inverse of the state-transition matrix used on the forward pass. This is not done in the smoothers implemented in this project. The reason is twofold: the implementation details are hidden to users of RCTL, and since the state-transition function is a simple regularization towards a mean state, it doesn't make sense to invert it.

Another way of formulating this argument is that RCTL is able to track the left ventricle regardless of which direction the ultrasound recording is played back.

In contrast, referring to table 1 in the numerical fixed-interval smoother example, the squared estimation error increases with a factor greater than five when not inverting the state-transition matrix.

6.1.6 Comparative discussion of the numerical smoother example

Several interesting comparisons can be made with the smoother example in section 2.4.3.

As shown in table 1, the fixed-interval smoother produces position estimates with approximately an order of magnitude less squared error than regular Kalman filtering. Furthermore, even a small lag is able to reduce the squared estimation error significantly. In figure 4, two less-than-perfect forward and backward estimates are fused into a smoothed estimate that is better than any one alone.

The same behavior is not observed when using smoothing techniques with RCTL. When applied to a properly configured extended Kalman filter tracker, the forward and backward pass is combined into something with almost the same accuracy, when comparing to the ground-truths.

It is also evident from the table that when not inverting the state-transition matrix on the backward pass, the reduction in squared estimation error by smoothing is much smaller, while a final update step instead of prediction step does not seem to have any great impact.

The estimated covariances are shown in similar fashion in figure 5, where their logarithms are plotted due to the large dynamic range. The backward filter has much higher estimated variances than the forward filter since it is initialized with infinite uncertainty (a diagonal matrix of $1.0 \cdot 10^{10}$ is used in the simulation). Furthermore, the estimated uncertainty gets lower with time for both the forward and backward filter (note that the time is reversed for the latter), and the fused error estimate is lower than both.

The main explanation for the difference between the example and the results obtained from the RCTL framework is that in the example, the backward estimates are never interpreted as more informative than they are. This is evident by the large backward covariance in the figure. In RCTL, the edge-detectors may report information about edges that do not exist, with a lower associated covariance than what can be justified. Additionally, the measurements in the numerical example are always a noisy version of the true measurements, which is not the case in RCTL if the edge-detections for some reason is started in the wrong part of the image volume.

6.2 The statistical smoother

The statistical smoother was not able to significantly improve the performance, and this section discusses the most likely cause. The algorithm is still included

in this project since the underlying idea of using the covariance intersection algorithm to combine regular Kalman filter estimates with learned information from manual ground-truths is interesting.

The clever separation between global and local parameters in RCTL makes the statistical smoother possible to implement. Using the regular Kalman tracker to obtain the correct alignment of the left ventricle in addition to a deformation estimate and subsequently fusing the deformation part with the learned mean deformation is intuitively reasonable.

Without this clear separation, e.g. only control node displacements from an initial shape as parameters, the recording probe location would be reflected in the deformation parameters, thus rendering the statistical smoother technique useless.

The covariance multiplier (375000) was chosen by trial and error with a script in order to maximize the performance. The learned information will not be given much weight with this large value. Figure 25 shows nearly identical performance when compared to the extended Kalman filter. It was observed that decreasing the covariance multiplier below 15000 increased the mean absolute surface error to over 3 mm.

The large spread of the ground-truth volume curves shown in figure 26 explains why average information is not of much use in single cases. The reasoning is that the widely differing volume curves reflect widely differing deformation patterns in the Doo-Sabin representation of the ground-truths. As a result, few of the recordings are well explained by the learned average heart motion.

This large spread was observed for the first time when investigating the low performance gain offered by the statistical smoother.

6.3 Limitations and future work

6.3.1 The extended Kalman filter is not optimal

Using the forward-backward formulation of optimal Kalman smoothing assumes that the forward- and backward-running filters are optimal. The real-time contour tracking library uses extended Kalman filters, which are not in general optimal filters and hence the smoothed results are not guaranteed to be optimal.

However, the only source of non-linearities in the RCTL framework is the global rotation. According to [6, pp. 7] it is believed that the linearization error is small since there is very little change in global rotation of the heart between successive frames.

6.3.2 A different measurement model

This study indicates that smoothing algorithms offers little change in tracking accuracy. It is argued that this is a consequence of the measurement model being defined relative to the predicted contour location. An interesting topic for future work is to modify the way edge-detections are performed in order to overcome this problem.

A measurement model not defined relative to the predicted contour might also solve other problems. A related project [24] attempted to implement an unscented Kalman filter within the RCTL framework, and concluded with the current measurement not being suited for a practical implementation.

6.3.3 Deformable model with more degrees of freedom

As shown in figure 27, the deformable left ventricle model's degrees of freedom is too low to give a reliable representation of the ground-truth segmentations. This effect is most noticeable in the absolute volume errors. A topic for future work is therefore to investigate the performance of the smoother algorithms compared to the regular EKF-tracker, when using a deformable model of higher resolution.

An informal experiment performed with Rctl3dApp showed better agreement with a subdivided (refined) version of this project's Doo-Sabin model when tracking on a rendered ground-truth. There is however a risk that more degrees of freedom in the model may reduce the smoothness because of increased modeling of edge-detection noise.

6.3.4 Inertia of the fitted models

According to [22, p. 42], "... [The Kalman smoother] can also be used to counteract the inherent inertia of the estimates from Kalman filters due to temporal regularization that only pulls the estimates back in time, but not forward". A topic for future work is to investigate whether the smoothing algorithms implemented in this project offers any improvement with regards to this time-lag in the contour estimates.

7 Conclusion

Four classes of Kalman smoothing algorithms have been presented. The first three are standard textbook solutions: fixed-point, fixed-lag, and fixed-interval smoothing. The fourth is the experimental statistical smoother. Additionally, a numerical example of an unrelated physical system where Kalman smoothing techniques do offer a large improvement in estimator accuracy has been presented.

This work propose how the different smoother types can be used in the context of medical image segmentation. The answer to the question if Kalman smoothing techniques offer any improvement within the RCTL framework is negative. No smoothing algorithm offered consistently better performance than the existing extended Kalman filter. In the special case of fixed-lag smoothing with lag three, a performance degradation was observed.

The lack of performance improvement from smoothing techniques can be traced back to the definition of the measurement model in RCTL. The smoothing process reduces to approximately the average of two similar estimates, where the averaging process is because of RCTL's default auto-scaling of measurement noise values.

Disabling the default noise auto-scaling has also been investigated with the purpose of reducing estimate overconfidence, but no great differences were observed.

The experimental statistical smoother performs nearly identically to the Kalman filter tracker alone. The left-ventricle deformation through a heart-cycle varies too much from person to person that knowledge of average shape deformation is of any use. This limits the maximum amount of average deformation knowledge that can be fused with a regular Kalman filter estimate, without degrading the estimator performance.

Finally, the deformable Doo-Sabin surface model of the left ventricle used in this project is shown to not have enough degrees of freedom to closely represent the ground-truth meshes. In other words, even if the smoothers produced exactly the same estimates as the ground-truth segmentation, the representation as a Doo-Sabin surface introduces non-negligible errors. This has largest impact on the lowest possible absolute volume error, which the extended Kalman filter tracker nearly attains. Therefore, most of the focus has been on the mean absolute surface error when discussing and comparing performance.

References

[1] S. Norway, *Causes of death 1995-2006.* Official statistics of Norway, Statistics Norway, 2009.

[2] J. S. Suri, S. K. Setarehdan, and S. Singh, *Advanced algorithmic approaches to medical image segmentation: state-of-the-art applications in cardiology, neurology, mammography and pathology.* Berlin: Springer, c2002.

[3] N. Schiller, "Two-dimensional echocardiographic determination of left ventricular volume, systolic function, and mass. summary and discussion of the 1989 recommendations of the american society of echocardiography.," *Circulation*, vol. 84, pp. I280–7–, Sept. 1991.

[4] F. Orderud, "A framework for real-time left ventricular tracking in 3d+t echocardiography, using nonlinear deformable contours and kalman filter based tracking," in *Computers in Cardiology, 2006*, pp. 125 –128, sept. 2006.

[5] F. Orderud, J. Hansgård, and S. Rabben, "Real-time tracking of the left ventricle in 3d echocardiography using a state estimation approach," in *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2007* (N. Ayache, S. Ourselin, and A. Maeder, eds.), vol. 4791 of *Lecture Notes in Computer Science*, pp. 858–865, Springer Berlin / Heidelberg, 2007.

[6] F. Orderud and S. Rabben, "Real-time 3d segmentation of the left ventricle using deformable subdivision surfaces," in *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, pp. 1 –8, june 2008.

[7] J. Noble and D. Boukerroui, "Ultrasound image segmentation: a survey," *Medical Imaging, IEEE Transactions on*, vol. 25, pp. 987 –1010, aug. 2006.

[8] T. L. Szabo, *Diagnostic ultrasound imaging: inside out.* Amsterdam: Elsevier Academic Press, 2004.

[9] W. D. O'Brien, Jr., "Assessing the risks for modern diagnostic ultrasound imaging," *Japanese journal of applied physics*, pp. 2781–2788, 1998.

[10] J. Bamber and C. Daft, "Adaptive filtering for reduction of speckle in ultrasonic pulse-echo images," *Ultrasonics*, vol. 24, no. 1, pp. 41 – 44, 1986.

[11] S. I. Fox, *Human physiology.* No. 12th ed., New York: McGraw-Hill, c2011.

[12] R. G. Brown and P. Y. Hwang, *Introduction to random signals and applied Kalman filtering: with MATLAB exercises and solutions.* No. 3rd ed., New York: Wiley, 1997.

[13] S. Julier and J. Uhlmann, "A non-divergent estimation algorithm in the presence of unknown correlations," in *American Control Conference, 1997. Proceedings of the 1997*, vol. 4, pp. 2369 –2373 vol.4, jun 1997.

[14] R. E. Kalman, "A new approach to linear filtering and prediction problems," *Transactions of the ASME–Journal of Basic Engineering*, vol. 82, no. Series D, pp. 35–45, 1960.

[15] R. J. Meinhold and N. D. Singpurwalla, "Understanding the kalman filter," *The American Statistician*, vol. 37, no. 2, pp. pp. 123–127, 1983.

[16] D.-J. Lee, "Nonlinear estimation and multiple sensor fusion using unscented information filtering," *Signal Processing Letters, IEEE*, vol. 15, pp. 861 – 864, 2008.

[17] D. Fraser and J. Potter, "The optimum linear smoother as a combination of two optimum linear filters," *Automatic Control, IEEE Transactions on*, vol. 14, pp. 387 – 390, aug 1969.

[18] E. Catmull and J. Clark, "Recursively generated b-spline surfaces on arbitrary topological meshes," *Computer-Aided Design*, vol. 10, no. 6, pp. 350 – 355, 1978.

[19] G. Farin and D. Hansford, *The essentials of CAGD.* Natick, Mass.: A.K. Peters, c2000.

[20] L.-E. Andersson and N. F. Stewart, *Introduction to the mathematics of subdivision surfaces.* Philadelphia, Pa.: SIAM, cop. 2010.

[21] J. Stam, "Exact evaluation of catmull-clark subdivision surfaces at arbitrary parameter values," in *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '98, (New York, NY, USA), pp. 395–404, ACM, 1998.

[22] F. Orderud, *Real-time segmentation of 3D echocardiograms using a state estimation approach with deformable models*, vol. 2010:55. Trondheim: Norges teknisk-naturvitenskapelige universitet, 2010.

[23] S. R. Snare, *Quantitative cardiac analysis algorithms for pocket-sizedultrasound devices*, vol. 2011:107. Trondheim: Norges teknisk-naturvitenskapelige universitet, 2011.

[24] S. Storve, "Comparison of kalman filters for ultrasound heart-valve tracking, with an emphasis on the unscented filter and the mitral valve, ntnu, ttt4520 signal processing in medical applications, specialization project," tech. rep., 2011.

A Derivation of covariance intersection under the independency assumption

Assume that θ_1 and θ_2 are two *unbiased* and *independent* estimates of some parameter θ. Their covariances are σ_1^2 and σ_2^2, respectively. The following derivation of how to linearly fuse the two estimates into a single estimate with the lowest possible resulting covariance is based on [12, pp 323].

The linear blend of the two estimates can be written in general as

$$\hat{\theta} = k_1\hat{\theta_1} + k_2\hat{\theta_2}$$

We can eliminate one constant by requiring the fused estimate to be unbiased, and using that $\mathbb{E}\left\{\hat{\theta_1}\right\} = \theta$ and $\mathbb{E}\left\{\hat{\theta_2}\right\} = \theta$ since they are assumed to be unbiased.

$$\mathbb{E}\left\{\hat{\theta}\right\} = \mathbb{E}\left\{k_1\hat{\theta_1} + k_2\hat{\theta_2}\right\} = k_1\theta + k_2\theta = (k_1 + k_2)\theta = \theta \tag{20}$$

From equation 20, we can infer that $k_1 + k_2 = 1$, since it must hold for all values of the parameter θ in order for the estimator to be unbiased. Hence, we rename the one free parameter to k, and get the following.

$$\hat{\theta} = k\hat{\theta_1} + (1-k)\hat{\theta_2} \tag{21}$$

The fused covariance can thus be written as

$$\mathbb{E}\left\{(\theta - \hat{\theta})^2\right\} = \mathbb{E}\left\{(\theta - k\hat{\theta_1} - (1-k)\hat{\theta_2})^2\right\} \tag{22}$$

In order to use our knowledge about the estimate covariances σ_1^2 and σ_2^2, we express both sides in terms of the following estimation errors

$$e_1 = \theta - \hat{\theta} \tag{23a}$$

$$e_2 = \theta_1 - \hat{\theta_1} \tag{23b}$$

$$e_3 = \theta_2 - \hat{\theta_2} \tag{23c}$$

To achieve this, we assume the right-hand side of equation 22 can be expressed as $Ae_1 + Be_2$ and solve for A and B.

$$\theta - k\hat{\theta_1} - (1-k)\hat{\theta_2} = Ae_1 + Be_2$$

$$\theta - k\hat{\theta_1} - (1-k)\hat{\theta_2} = A\theta - A\hat{\theta_1} + B\theta - B\hat{\theta_2}$$

By comparing coefficients of $\hat{\theta_1}$ and $\hat{\theta_2}$ we must have $-k = -A \Leftrightarrow A = k$ and $(1-k) = B$, which also is seen to agree with the coefficients of θ. Hence, we have the following expression for the fused covariance

$$\mathbb{E}\left\{e^2\right\} = \mathbb{E}\left\{(ke_1 + (1-k)e_2)^2\right\} = k^2\mathbb{E}\left\{e_1^2\right\} + 2k(1-k)\mathbb{E}\left\{e_1e_2\right\} + (1-k)^2\mathbb{E}\left\{e_2^2\right\} \tag{24}$$

We want to minimize this expression with respect to the blending factor k, thus we compute the two first partial derivatives.

$$\frac{\partial \mathbb{E}\left\{e^2\right\}}{\partial k} = 2k\mathbb{E}\left\{e_1^2\right\} + (2-4k)\mathbb{E}\left\{e_1e_2\right\} - 2(1-k)\mathbb{E}\left\{e_2^2\right\} \tag{25}$$

$$\frac{\partial^2 \mathbb{E}\left\{e^2\right\}}{\partial k^2} = 2\mathbb{E}\left\{e_1^2\right\} - 4\mathbb{E}\left\{e_1e_2\right\} + \mathbb{E}\left\{e_2^2\right\} = 2\mathbb{E}\left\{(e_1 - e_2)^2\right\} \geq 0 \tag{26}$$

We see from equation 26 that any values of k for which the expression 25 is zero must be a minimum value for the covariance of the fused estimate. Hence, we proceed by solving

$$\frac{\partial \mathbb{E}\left\{e^2\right\}}{\partial k} = 0$$

$$2k\mathbb{E}\left\{e_1^2\right\} + 2\mathbb{E}\left\{e_1e_2\right\} - 4k\mathbb{E}\left\{e_1e_2\right\} - 2\mathbb{E}\left\{e_2^2\right\} + 2k\mathbb{E}\left\{e_2^2\right\} = 0$$

$$k\mathbb{E}\left\{e_1^2\right\} + \mathbb{E}\left\{e_1e_2\right\} - 2k\mathbb{E}\left\{e_1e_2\right\} - \mathbb{E}\left\{e_2^2\right\} + k\mathbb{E}\left\{e_2^2\right\} = 0$$

$$k = \frac{\mathbb{E}\left\{e_2^2\right\} - \mathbb{E}\left\{e_1e_2\right\}}{\mathbb{E}\left\{e_1^2\right\} - 2\mathbb{E}\left\{e_1e_2\right\} + \mathbb{E}\left\{e_2^2\right\}} \tag{27}$$

By the assumption that $\hat{\theta_1}$ and $\hat{\theta_2}$ are independent, we have $\mathbb{E}\left\{\hat{\theta_1}\hat{\theta_2}\right\} = \theta^2$ and thus it follows that $\mathbb{E}\left\{e_1 e_2\right\} = \mathbb{E}\left\{(\theta - \hat{\theta_1})(\theta - \hat{\theta_2})\right\} = \theta^2 - \theta\mathbb{E}\left\{\hat{\theta_1}\right\} - \theta\mathbb{E}\left\{\hat{\theta_2}\right\} + \mathbb{E}\left\{\hat{\theta_1}\hat{\theta_2}\right\} = \theta^2 - \theta^2 - \theta^2 + \theta^2 = 0$.

Hence, equation 27 can be written

$$k = \frac{\mathbb{E}\left\{e_2^2\right\}}{\mathbb{E}\left\{e_1^2\right\} + \mathbb{E}\left\{e_2^2\right\}} = \frac{\sigma_2^2}{\sigma_1^2 + \sigma_2^2} \tag{28}$$

From equation 21 we have

$$\hat{\theta} = k\hat{\theta_1} + (1-k)\hat{\theta_2} = \frac{\hat{\theta_1}\sigma_2^2}{\sigma_1^2 + \sigma_2^2} + \frac{\hat{\theta_2}\sigma_1^2}{\sigma_1^2 + \sigma_2^2}$$

From equation 24 we have

$$\mathbb{E}\left\{e^2\right\} = k^2\sigma_1^2 + 0 + (1-k)^2\sigma_2^2 = \frac{\sigma_2^4\sigma_1^2 + \sigma_1^4\sigma_2^2}{(\sigma_1^2 + \sigma_2^2)^2}$$

$$\mathbb{E}\left\{e^2\right\} = \frac{\sigma_1^2\sigma_2^4 + \sigma_1^4\sigma_2^2}{(\sigma_1^2 + \sigma_2^2)^2} = \frac{\sigma_1^2\sigma_2^2(\sigma_1^2 + \sigma_2^2)}{(\sigma_1^2 + \sigma_2^2)^2} = \frac{\sigma_1^2\sigma_2^2}{\sigma_1^2 + \sigma_2^2}$$

In summary, the expression for the optimal fusion and its associated covariance has been shown to be

$$\hat{\theta} = \frac{\hat{\theta_1}\sigma_2^2 + \hat{\theta_2}\sigma_1^2}{\sigma_1^2 + \sigma_2^2} \tag{29}$$

and

$$\sigma^2 = \frac{\sigma_1^2\sigma_2^2}{\sigma_1^2 + \sigma_2^2} \tag{30}$$

B Conversion of triangle meshes to RCTL state vectors

Given a triangle mesh, how can this be converted into a state vector for a suitable RCTL model so that the mesh is represented as closely as possible? Below it will be shown that the problem can be reduced to rendering triangles as voxels (volume "pixels").

This problem arises in the statistical smoother case, because the manual ground truth segmentations produced by AutoLVQ with manual editing, are only known in the form of a triangle mesh, while the statistical smoother requires a RCTL-state vector description.

The solution chosen was to let RCTL itself do this conversion by creating a voxelized copy of the ground truth meshes. That is, for each recording, every frame's triangle mesh is rendered as voxels into a recording in the RCTL-specific format .h5c, which is basically a four-dimensional array containing image data for all frames. By rendering all triangles in a mesh as continous bright white pixels, a sharp and clearly-defined voxelized version of the mesh results. By finally using RCTL to track on this, the desired state vectors are obtained.

The 4D array consists of unsigned 8-bit chars. One dimension is for frame number, and the remaining three dimensions are x, y, and z.

This custom format .h5c is simply a .hdf file (the letter 'c' denotes cartesian image volumes), which is easily created by first creating the data structure in Matlab and then storing it as a .hdf file by using the Matlab script HdfExport written by Fredrik Ordrud.

The units used in the ground truth meshes is meters which allows an easy physical interpretation. In order to get the recording orientation correct and uniform over all generated recordings, the physical space covered by each recording must be the same for all recording. A global bounding box for all recordings can be computed by finding maximum and minimum x, y, and z coordinate over all triangles. This gives a global bounding box, such that if the voxel space is at least as big, all rendered ground truth segmentations are guaranteed to fit into it.

The only remaining free parameters are spacing in x, y, and z direction, i.e. how many voxels will be used to represent the physical space between maximum and minimum limits in each of the three directions. Approximately 200 in each dimension were used in this project.

B.1 How to parameterize a triangle?

This section briefly presents a bijection between the unit square $\mathbb{S} = [0,1] \times [0,1]$ into an arbitrary triangle defined by $\mathbf{p_1}, \mathbf{p_2}, \mathbf{p_3} \in \mathbb{R}^N$, which will be used below to sample points on triangles. The line segment between $\mathbf{p}_2$ and $\mathbf{p}_3$ as $\mathbf{l}_1(u)$ is parameterized in such a way that $\mathbf{l}_1(0) = \mathbf{p}_2$ and $\mathbf{l}_1(1) = \mathbf{p}_3$. This is accomplished by

$$\mathbf{l}_1(u) = \mathbf{p}_2 + u(\mathbf{p}_3 - \mathbf{p}_2) \quad (u \in [0,1]) \tag{31}$$

Then, for any $u \in [0,1]$, parameterize the line segment between $\mathbf{p}_1$ and $\mathbf{l}_1(u)$ as $\mathbf{l}_2(v)$ in such a way that $\mathbf{l}_2(0) = \mathbf{p}_1$ and $\mathbf{l}_2(1) = \mathbf{l}_1(u)$. This is accomplished by

$$\mathbf{l}_2(v) = \mathbf{p}_1 + v(\mathbf{l}_1(u) - \mathbf{p}_1) \tag{32}$$

By combining equations 31 and 32 the following explicit expression for the bijection is obtained.

$$\mathbf{p}(u,v) = \mathbf{p}_1 + v\mathbf{p}_2 + uv(\mathbf{p}_3 - \mathbf{p}_2) - v\mathbf{p}_1 \quad (u,v \in [0,1]) \tag{33}$$

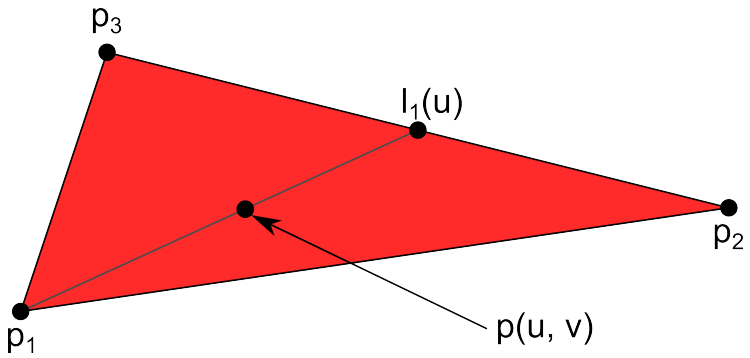


Figure 29: Illustration of bijection between unit square and an arbitrary triangle.

B.2 Rendering a triangle as voxels

The interval is split into N equal parts, which gives N subintervals of length $h = \frac{x_{\max} - x_{\min}}{N}$. Given $x \in [x_{\min}, x_{\max}]$, how to get subinterval index? Solving

$$x_{\min} + hr = x$$

for r gives

$$r = \frac{x - x_{\min}}{h}$$

This gives the number of interval widths h as a real number. In order to obtain an integer index, the following ceiling value is computed.

$$k = \text{ceil}\left(\frac{x - x_{\min}}{h}\right) \tag{34}$$

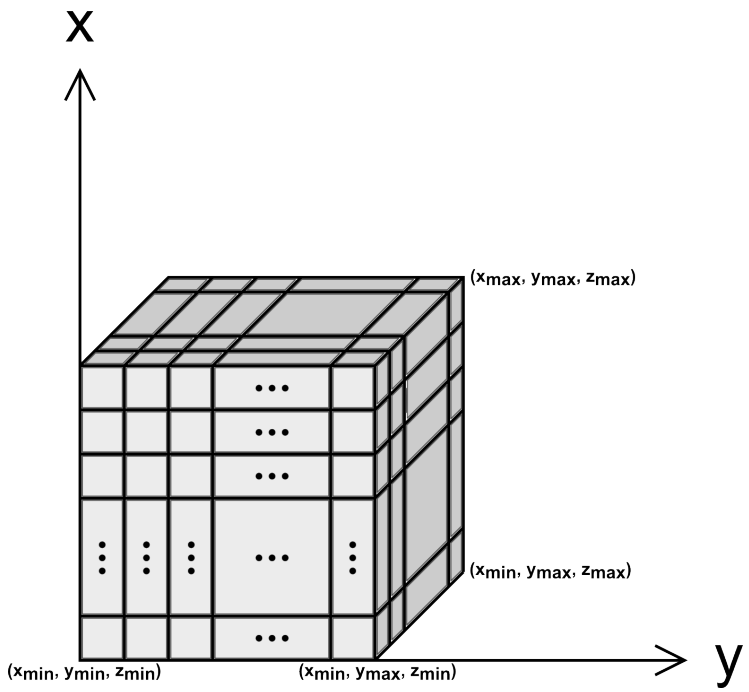


Figure 30: Image voxel volume. The physical x, y, and z limits are partitioned into Nx, Ny, and Nz cells, respectively.

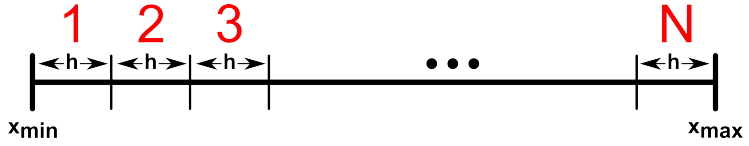


Figure 31: Illustration of mapping between physical coordinate and voxel index.

with the additional boundary-case handling rule that if k is 0, let $k = 1$ instead. Thus, as shown in figure 31, a number in the interval is mapped into an index in the voxel data array.

By making sure that the maximum sample spacing is less than the resolution of the .h5c image volume each rendered triangle will appear as completely filled. This can be validated by playing back the rendered mesh in the Rctl3dApp; if

the contour has visible black spots, the triangle sampling density is increased until the problem disappears.

With this method for rendering a single triangle, rendering a full triangle mesh is simple; just iterate through all triangles and render them one by one. The final recording is made by rendering each ground truth mesh in a separate frame.

B.3 State-vectors from rendered ground truths

After obtaining a .h5c recording consisting of one frame for each ground-truth triangle mesh, the desired state-vectors can be obtained by using RCTL to track the recording and save the estimates. This is illustrated in figure 32 where the Rclt3dApp is used to visualize the tracking.

The state-vector version of the ground-truth needed for the statistical smoother and Doo-Sabin minimum error computation were computed by a Matlab script using RctlMatlab to perform the tracking and return the desired state vectors. A few warmup-cycles is performed before storing any state vector estimates in order to give the tracker enough time to stabilize.

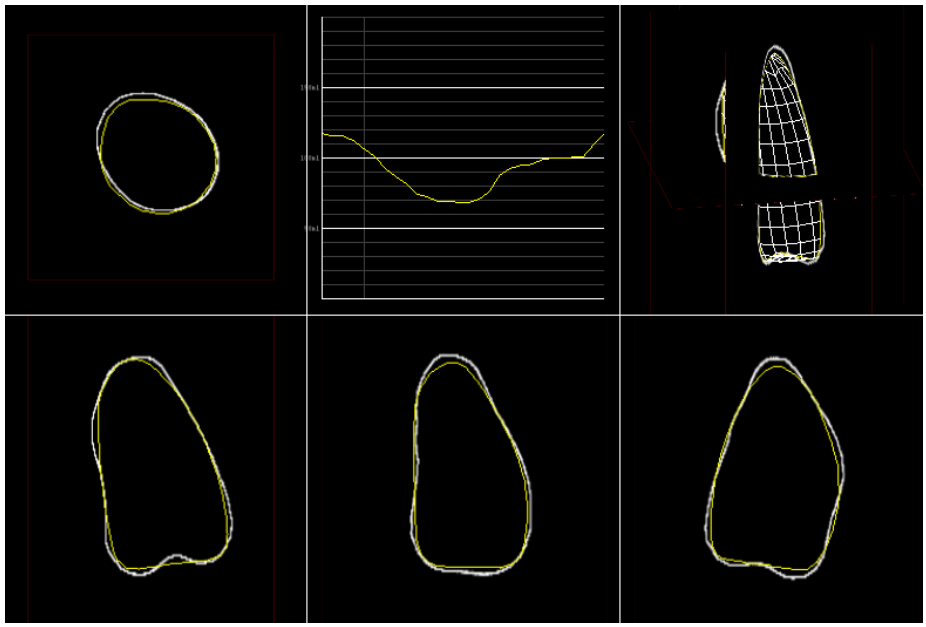


Figure 32: Screenshot of Rctl3dApp tracking on a rendered ground truth mesh. Note the thin bright white contour floating in black space and the tightly tracked contour (yellow). Edge-detector type ”gradient” was used.

C Numerical example comparing the two covariance intersection algorithms

Assume that two independent position estimates are obtained as Gaussian distributions, perhaps from two separate GPS units each containing a Kalman filter. Figure 33 shows the situation. The ellipses shown are contours of equal probability, one standard deviation from the mean. The first position estimate is

$$\mathbf{x}_1 = \begin{pmatrix} 1.5 \\ 2.0 \end{pmatrix}$$
$$\mathbf{P}_1 = \begin{pmatrix} 2.0 & 1.0 \\ 1.0 & 1.0 \end{pmatrix}$$

and the second position estimate is

$$\mathbf{x}_2 = \begin{pmatrix} 0.9 \\ 2.3 \end{pmatrix}$$
$$\mathbf{P}_2 = \begin{pmatrix} 0.1 & 0.0 \\ 0.0 & 5.0 \end{pmatrix}$$

By applying the covariance intersection algorithm assuming independent estimates, we get the following fused estimate, which is shown in black in the figure.

$$\mathbf{x}_{\text{independent}} = \begin{pmatrix} 0.934 \\ 1.770 \end{pmatrix}$$
$$\mathbf{P}_{\text{independent}} = \begin{pmatrix} 0.095 & 0.043 \\ 0.043 & 0.474 \end{pmatrix}$$

If one instead use the covariance intersection algorithm assuming an unknown correlation between the two estimates, the following estimate is obtained, which is shown in green in the figure.

$$\mathbf{x}_{\text{unknowcorr}} = \begin{pmatrix} 0.952 \\ 1.758 \end{pmatrix}$$
$$\mathbf{P}_{\text{unknowncorr}} = \begin{pmatrix} 0.248 & 0.117 \\ 0.117 & 0.805 \end{pmatrix}.$$

Comparing the black and green covariance ellipses to the covariance ellipses of both estimates, it is clear that they are smaller, and hence there is less uncertainty in the fused estimates.

The unknown correlation assumption manifests itself in that the corresponding covariance ellipse is bigger than the fused covariance ellipse assuming independence of estimates. This is also pointed out in [13]; estimation certainty is traded for a provably consistent fused estimate without any assumptions on the correlation between the two input estimates.

It can also be observed that in this specific situation, the fused mean estimates are almost equal.

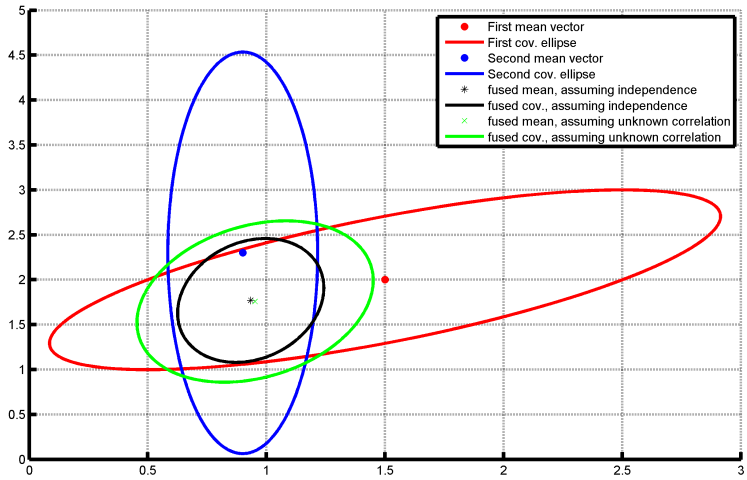


Figure 33: Two uncertain position estimates fused into a more certain estimate. The ellipses are covariance ellipses, i.e. constant-probability curves of one standard deviation.

D Derivation of the state-space formulation of the mass-spring system

A damped mass-spring system is chosen as a simple non-trivial dynamical system to demonstrate Kalman smoothing on. The governing second-order differential equation is converted to two coupled first-order differential equations.

D.1 The damped mass-spring system

Figure 4a shows the damped mass-spring system in question. Newton's second law of motion will be used to derive a second-order linear differential equation for the displacement from rest, denoted by $x(t)$. The spring force is given by $F_s = -kx$ and the friction is modeled as being proportional to the velocity, i.e. $F_f = -c\frac{dx(t)}{dt}$. The differential equation thus becomes

$$\sum F = -kx(t) - c\frac{dx(t)}{dt} = ma = m\frac{d^2x(t)}{dt^2} \tag{39}$$

which can be written

$$\frac{d^2x(t)}{dt^2} + \frac{c}{m}\frac{dx(t)}{dt} + \frac{k}{m}x(t) = 0 \tag{40}$$

D.2 The analytical solution by Laplace transforms

By taking the Laplace transform of both sides, equation 40 translates to

$$X(s) = \frac{sx(0) + x'(0) + \frac{c}{m}x(0)}{s^2 + \frac{c}{m}s + \frac{k}{m}} \tag{41}$$

By taking the inverse Laplace transform of both sides, we get

$$\begin{aligned} x(t) = \mathcal{L}^{-1}\{X(s)\} = \frac{1}{2\sqrt{c^2 - 4km}}\Bigg(& cx(0)\Big[\exp(K_1t)\exp(K_2t) - \exp(K_1t)\exp(-K_2t)\Big] \\ & + x(0)\sqrt{c^2 - 4km}\Big[\exp(K_1t)\exp(-K_2t) + \exp(K_1t)\exp(K_2t)\Big] \\ & + 2x'(0)m\Big[\exp(K_1t)\exp(K_2t) - \exp(K_1t)\exp(-K_2t)\Big]\Bigg) \end{aligned} \tag{42}$$

where

$$K_1 = -\frac{c}{2m} \tag{43}$$

$$K_2 = \frac{\sqrt{c^2 - 4km}}{2m} \tag{44}$$

D.3 Converting the second-order differential equation to two coupled first-order differential equations

State-space methods require that equation 40 be written as a set of coupled first-order differential equations. This is done by the standard procedure of introducing one new variable for each of the higher order derivatives.

$$\begin{aligned} x_1(t) &= x'(t) \\ x_2(t) &= x_1'(t) = x''(t) = kx_1(t) - cx_2(t) \end{aligned}$$

This can be written on matrix form as

$$\begin{pmatrix} x_1'(t) \\ x_2'(t) \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ -\frac{k}{m} & -\frac{c}{m} \end{pmatrix} \begin{pmatrix} x_1(t) \\ x_2(t) \end{pmatrix} \tag{46}$$

or

$$\mathbf{x}'(t) = \mathbf{F}\mathbf{x}(t) \tag{47}$$

By a standard result in the theory of linear systems, the solution of this vector differential equation is

$$\mathbf{x}(t) = e^{\mathbf{F}(t-t_0)}\mathbf{x}(t_0) \tag{48}$$

We define the state-transition matrix as $\phi(\Delta t) = e^{\mathbf{F}\Delta t}$. This matrix moves the system from the state at time t_0 to the state at time $t_1 = t_0 + \Delta t$. We thus have

$$\mathbf{x}(t) = \phi(t - t_0)\mathbf{x}(t_0) \tag{49}$$

Given actual values for Δt, m, k, and c, it is easy to compute this matrix exponential numerically, e.g. by using the SciPy[3] **expm** function .

[3]SciPy is an open source library of algorithms and mathematical tools for the Python programming language.

E Files included with this report

Additional details about this work can be found in the accompanying files:

- The Python source code used to generate all figures in the numerical smoother example in section 2.4.3.
- Algorithm evaluation details for all recordings and all estimation algorithms, the raw data used to generate the tables in section 5.1.1.