Available online at www.sciencedirect.com
SCIENCE DIRECT®
Real-Time Imaging 9 (2003) 433–439
www.elsevier.com/locate/rti

Kalman filter recipes for real-time image processing

Michael Piovoso, Phillip A. Laplante*

Great Valley Graduate Center, Penn State University, 30 Swedesford Road, Malvern, PA 19355-1443, USA

Abstract

Kalman filters are an important technique for building fault-tolerance into a wide range of systems, including real-time imaging. From a software engineering perspective, however, it is not easy to build Kalman filters. Each has to be custom designed and most software engineers are not sufficiently grounded in the necessary systems theory to perform this design.

The contributions of this paper, therefore, are a set of recipes for implementation of the Kalman filter to a variety of real-time imaging settings, the presentation of a set of object-oriented requirements, and a design for a class of Kalman filters suitable for real-time image processing.

First, we describe the Kalman filter and motivate its use as a mechanism for fault-tolerant computing and sensor fusion. Next, the details of using Kalman filters in imaging applications are discussed and several associated algorithms presented. Then, the advantages of using object-oriented specification, design and languages for the implementation of Kalman filters are explored. Finally, we present a specification and design for a class of Kalman filters, which is suitable for coding. This work extends significantly upon that first appearing in 2003 at an SPIE conference (Laplante and Neill, proceedings of the real-time imaging conference, SPIE, Santa Clara, January 2003, pp. 22–29).

1. Introduction

Fault-tolerant control of real-time systems is based on mitigating errors due to missing deadlines and bad decisions based on noisy data. Typical approaches for fault-tolerance in such systems include recovery blocks, redundant hardware and software, mean-square-error filtering, N-version programming, Kalman filtering and model checking [1,2].

Real-time imaging systems typically involve the processing of a stream of digital images captured via a camera, analysis of the images, then control of some system based on the results of the analysis. The typical configuration for a control system featuring imaging devices, sensors and actuators is shown in Fig. 1.

Since control of the system is based on the correct processing of the image within the required time frame, it is important that the images be as fault-free as possible and processed on time.

1.1. The Kalman filter

The Kalman filter was proposed in 1960 for use in optimal control of navigation systems based on non-imaging information [3]. However, the Kalman filter has been used for image processing since the early 1970s [4], and has appeared widely in image processing literature since then. It has also been used in a variety of applications.

For example, in addition to a complete literature review, Biemond et al. also provide a "fast" Kalman filter for recursive restoration of images that take advantage of an FFT to reduce the number of computations from $O(n^4)$ to $O(n^2 \log_2 n)$. Xie et al. used a Kalman filter for the recursive estimation of eye features for the purpose of tracking [5]. Lippiello et al. use a Kalman filter for real-time pose estimation of moving objects using a stereo video camera system [6]. Finally, Turney et al. provide an FPGA implementation for a per-pixel adaptive temporal Kalman filter [7].

A Kalman filter is used to estimate the state variables of a multivariable feedback control system subject to stochastic disturbances caused by noisy measurements of input variables. The Kalman filtering algorithm works by combining the information regarding the system

*Corresponding author. Tel.: +1-610-725-5314; fax: +1-610-889-1334.

E-mail addresses: mjp5@gv.psu.edu (M. Piovoso), plaplante@gv.psu.edu (P.A. Laplante).

1077-2014/$ - see front matter © 2003 Elsevier Ltd. All rights reserved.
doi:10.1016/j.rti.2003.09.005

dynamics with probabilistic information regarding the noise. The filter is very powerful in that it supports estimations of past, present and even future states and, in particular, can do so even when the precise nature of the noise is unknown.

There are two kinds of equations for the Kalman filter—time update equations and measurement update equations. The time update equations project forward in time the current state and error covariance estimates to obtain the a priori estimates for the next time step. The measurement update equations are responsible for the feedback in that they incorporate a new measurement into the a priori estimate to obtain an improved estimate. This will be seen shortly.

Unfortunately, Kalman filters are hard for software engineers to build, at least for those engineers that do not have the equivalent of a masters level education in electrical engineering. Each filter has to be "hand-designed" and debugged. Moreover, a great deal of expertise in control systems engineering is required to perform such manual designs. Finally, specialized hardware such as FPGAs, ASICs and DSP coprocessors are often used to implement the algorithms, which is not always practical in PC-based and mobile computing environments.

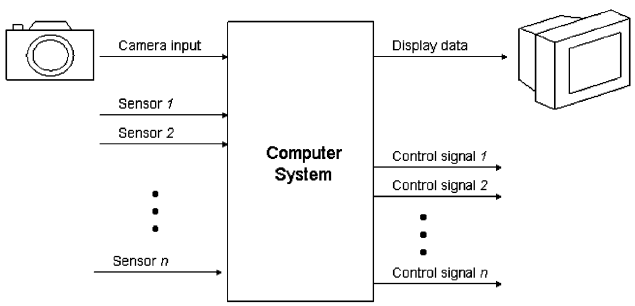

Fig. 1. A typical control system featuring imaging devices.

It is suspected that because of these shortcomings, the benefits of the Kalman filter in real-time imaging systems are not widely enjoyed. Our objective, therefore is to provide simple, easily coded algorithms that apply in a wide range of settings and without special hardware.

2. Kalman filtering for imaging applications

The basic problem to be considered in imaging applications is dealing with the degradation of image sequences due to noise introduced during the image acquisition process (Fig. 2).

Suppose that the image is captured by one (or more) camera(s) and the image is processed for display and/or for further algorithmic analysis and processing. The objective is to construct a filter that will reduce the noise, and in particular, remain insensitive to sudden spike errors that can easily fool other types of filters such as a mean-square-error (MSE) filter. While the median filter also works well in this regard, the Kalman filter can be tuned by adjusting its parameters based on the variations in the noise statistics, which the median filter cannot.

2.1. Image representation

While it is possible to model the camera input as just another sensor or the display device as simply the target for a set of control signals, it is probably the case that this simplification is unacceptable in many cases. Sensor inputs and control signals are often a single wire, differing significantly for the information that must be exchanged with imaging devices.

Representation of an image in a Kalman Filter can either be pixel-wise (as in [7]) or block-wise (as in [8]).

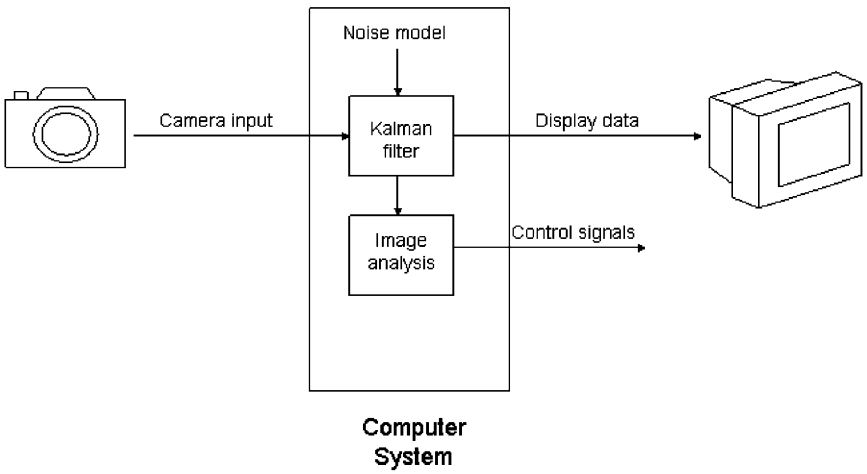

Fig. 2. Kalman Filter for image processing.

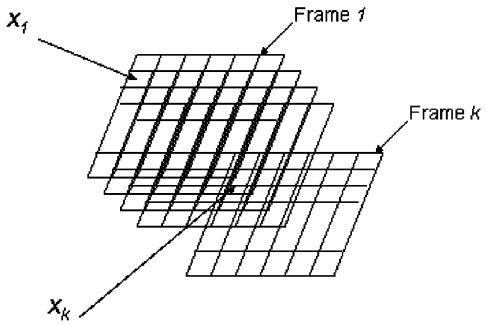

Fig. 3. A series of images captured from a camera.

The advantage of latter approach is that inter-pixel statistical characteristics can be taken into account, however, at significant computational cost. The former approach offers the advantage that it is faster, and we follow that construction.

The image model is as follows. Let x_k represent a particular pixel found in frame k (taken at time k) of the image (see Fig. 3).

Then each pixel represents a time series in frame k.

2.2. Kalman equations

The ideal noise-free pixel value is given by s_k, which is assumed to be a first order AR model. This is a model that is generally used to represent the behavior of pixels in video signals [9]. The process model is

$$s_{k+1} = as_k + n_k, \tag{1}$$

where a is a constant that depends on the signal statistics and n_k is the process noise, which is assumed to be white Gaussian with zero mean and variance σ_n^2. While it might be desirable to assume that the noise is Poisson distributed, making the assumption that the noise is Gaussian simplifies the design. Ultimately, however, it will be desirable to construct the class of filters in such a way as to be extensible to different noise sources.

In any case, the measured signal is given by

$$x_k = s_k + v_k, \tag{2}$$

where v_k is the independent additive zero mean Gaussian white noise with variance of σ_v^2.

If the noise and signal are wide-sense stationary random processes that are fully determined by their second-order statistics, then the Kalman filter is essentially a Weiner Filter. A more flexible model of the Kalman filter would not incorporate this assumption.

The Kalman filter output is represented by y_k, which is the estimate of the signal at time k. The variance of the estimation error variance is defined by

$$\sigma_k^2 = E[(y_k - s_k)^2]. \tag{3}$$

The Kalman filter gain is K_k.

For the single pixel case, the Kalman filtering algorithm is given by the following algorithm. Here $y_{-1} = 0$ and $\sigma_{-1}^2 = \sigma_v^2$.

$$K_k = \frac{a^2\sigma^2(k-1) + \sigma_n^2}{a^2\sigma^2(k-1) + \sigma_n^2 + \sigma_v^2},$$

$$y_k = K_k x_k + a[1 - K_k]y_{k-1},$$

$$\sigma_k^2 = a^2[1 - K_k]\sigma_{k-1}^2 + \sigma_n^2.$$

This algorithm is applied to each pixel, and each time instant. The estimated pixel value, y_k is output for each iteration, k to provide the filtered image output for display or image analysis purposes. The updated Kalman filter gain K_k and updated noise estimation, σ_k^2, are also output at each iteration. As before, a is a constant that depends on the signal statistics.

3. More recipes

The previous section introduced a model for a single camera and filtering one pixel at a time. In some cases, however, it is desirable to have 2 or more cameras with differing noise characteristics capturing the same image. Potential applications include robot microsurgery, vision-based navigation, and weapons targeting.

3.1. Two or more sensors

Assume that there are two or more cameras that observe the same image. In this case the model for the image of each sensor is

$$x_k = \Phi_k x_{k-1} + w_{k-1}, \tag{4}$$

where x_k is the vector of camera outputs for the kth pixel, Φ_k is the state transition matrix at time k that defines not only the structure, but also the interrelations between the pixel values and w_k is the state noise that drives the process. The observations are noise corrupted versions of the image pixels.

$$y_k = x_k + v_k, \tag{5}$$

where v_k is the corrupting noise in the images themselves. We assume that the observation noise is uncorrelated to the state noise. Furthermore, we assume that the mean of the initial pixel and its state error

covariance is known. That is

$$E[x(0)] = \hat{x}_0,$$
$$E[(x(0) - \hat{x}_0)(x(0) - \hat{x}_0)^T] = P_0. \quad (6)$$

There are two steps to the estimation. First, one needs to estimate the future state given the present information, and then to update that estimate as the next information becomes available. The estimate of the future pixel state given the present information is formulated as

$$\hat{x}_k(-) = \underset{x_k}{\operatorname{argmax}}\, P(x_k/y_{k-1}, y_{k-2}, \ldots, y_1),$$
$$\hat{x}_k(-) = \Phi_{k-1}\hat{x}_{k-1}(+), \quad (7)$$

where $\hat{x}_k(-)$ is the estimate of the state x_k given the information up to time $k-1$, and $\hat{x}_{k-1}(+)$ is the update of state estimate, $\hat{x}_{k-1}(-)$, again given all the information up to time $k-1$. The covariance of the error in $\hat{x}_k(-)$ is

$$P_k(-) = \Phi_{k-1}P_{k-1}(+)\Phi_{k-1}^T + Q_{k-1}, \quad (8)$$

where Q_{k-1} is the covariance of the state noise w_{k-1}, and $P_{k-1}(+)$ is updated state error covariance estimate, $P_{k-1}(-)$ with the sample y_{k-1}.

When the new data, y_k, becomes available the previous estimates of the state and state error covariance must be updated to reflect the new information,

$$\hat{x}_k(+) = \hat{x}_k(-) + K_k[y_k - \hat{x}_k(-)],$$
$$P_k(+) = [I - K_k]P_k(-),$$
$$K_k = P_k(-)[P_k(-) + R_k]^{-1}. \quad (9)$$

The term K_k is the Kalman gain and R_k is the covariance of the measurement noise, v_k. Fig. 4 illustrates the flow chart for this process.

The Kalman filter will output the updated, filtered estimates of the pixels from each of the cameras. In addition, the covariance of the errors in the state-estimates at each time k provides information as to the reliability of the estimates. If the state covariance matrix is not diagonal, then there exists a transformation of the states so that the covariance of the new states is diagonal. That transformation is given by the matrix whose columns are the eigenvectors of $P_k(+)$. Let t_i be the ith eigenvector of $P_k(+)$. Then,

$$P_k(+)t_{k,i} = \lambda_i t_{k,i} \quad (10)$$

and

$$\hat{\tilde{x}}_k(+) = T_k\hat{x}_k(+), \quad (11)$$

where T_k is the matrix whose columns are the orthonormal eigenvectors.

Since new states are statistically independent of each other with known variances given the eigenvalue associated with the corresponding eigenvector, a weighted average pixel can computed. The weights represent the fraction of the total variance associated with each element of $\hat{\tilde{x}}_k(+)$ and are given by

$$\omega_i = \frac{1/\lambda_i}{\sum_i 1/\lambda_i}. \quad (12)$$

The optimal estimate of the pixel value is

$$\bar{\hat{x}}_k(+) = \sum_i \hat{\tilde{x}}_{k,i}(+)\omega_i. \quad (13)$$

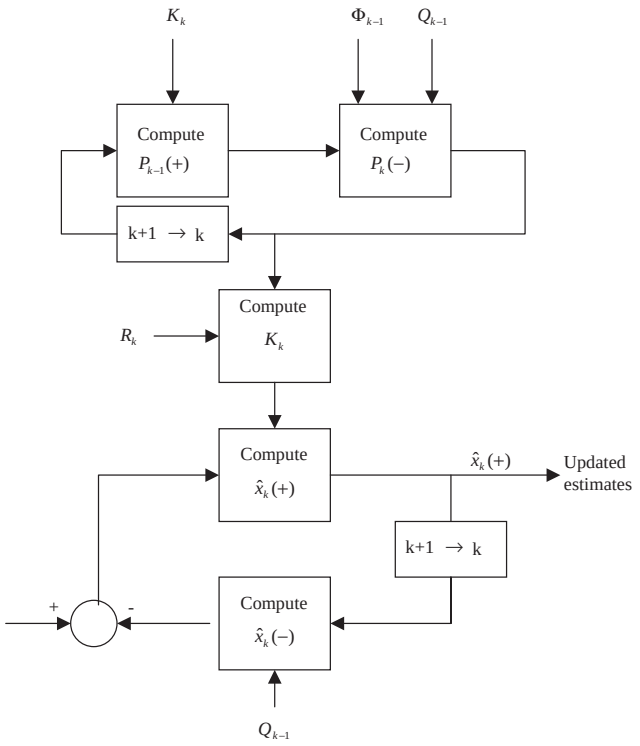

Fig. 4. Flow chart for multi-camera Kalman filtering process.

4. Designing a Kalman filter class

In general how do we "objectify" something that is plainly functional, but for which the number and nature of inputs and outputs differ, and hence the filter has to be handcrafted for each system. In essence, we wish to avoid the obvious solution of simply taking a procedural version of the Kalman filter (e.g. written in C) and "wrapping" it in such a way so that it is compiliable by an object-oriented language compiler (e.g. simplistically converting the C code to C++ by providing trivial class definitions). Such a wrapping approach will not take advantage of the benefits of object-oriented design and

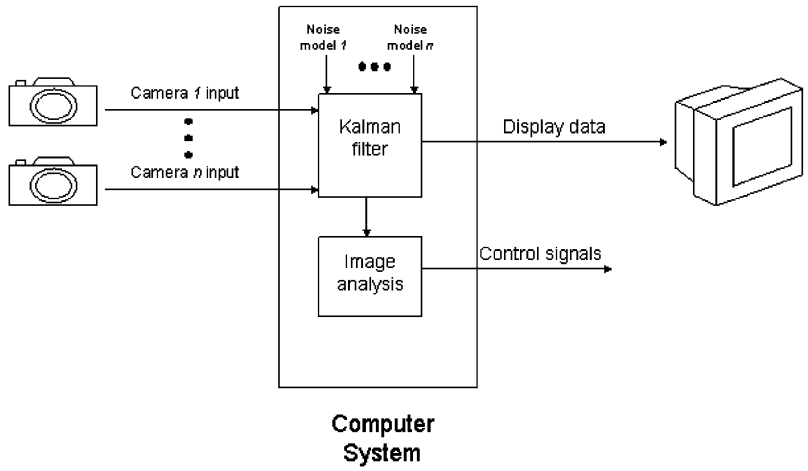

Fig. 5. A multi-camera image processing system.

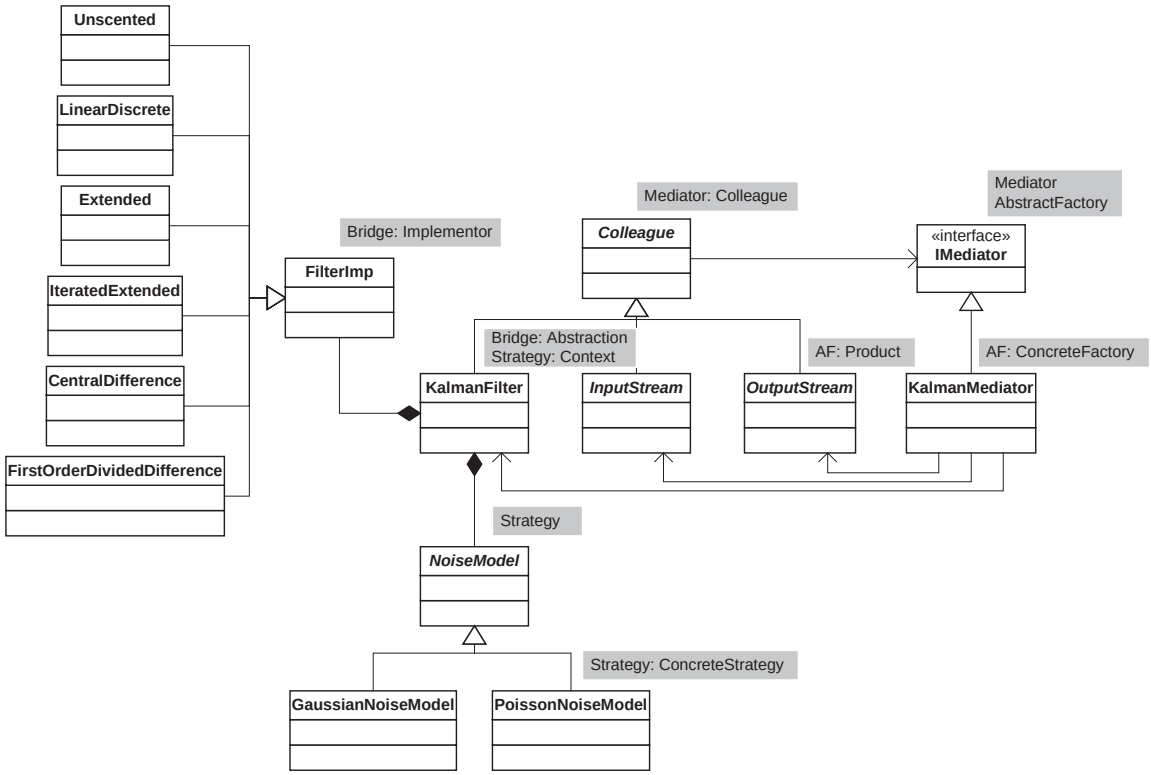

Fig. 6. A class design for a Kalman filter for image processing.

will certainly require handcrafting a new filter for every new system and for every change in the hardware.

In designing a class of Kalman filters for real-time image processing, it is desirable to abstract away all implementation details in such a way as to the benefits of object-orientation. These object-oriented aspects include the fact that the number of input sources may be greater than one, in which case we wish to combine (fuse) the inputs in such a way as to optimize overall noise reduction (see Fig. 5).

In addition, the sensor noise characteristics are to be abstracted as are the camera digitization scheme. The underlying image processing model should be variable from pixel-wise to block-wise processing. Finally, the underlying implementation of the Kalman filter should be transparent.

An object-oriented design of a Kalman filter is presented in Fig. 6. The design is annotated to indicate the GoF patterns used [10].

The motivation for the design was to produce a filter that could be extended for multiple applications, supporting a number of variant algorithms and noise models. The filter must accept n-inputs and the configuration chosen was based upon the "pipes and filters" architectural style supporting both pull and push modes.

The central class in the model is the `KalmanMediator`, this is both a concrete factory responsible for the creation of instances of the `KalmanFilter`, Input Stream and `OutputStream` classes and a mediator class that coordinates communication between those classes so that they can remain un-coupled, and therefore immune to changes in one another. It is this mediation role that allows for both push- and pull-mode operation; without a mediator every instance of the `KalmanFilter`, `InputStream` and `OutputStream` classes would have to know the mode the other instances were using.

The design also makes use of interface and implementation independence in the form of the Bridge and Strategy GoF patterns. The `KalmanFilter` contains a `NoiseModel` and a `FilterImp` instance each of which are abstract superclasses defined concretely for variant types of noise and filter algorithm, respectively. These instances are therefore interchangeable allowing for a number of different applications without modification to the existing software system (the Open Closed Principle). In addition, other subclasses of `NoiseModel` and `FilterImp` can be defined for future uses.

5. Real-time aspects

The key to real-time processing is predictable performance, not necessarily fast processing. For real-time motion applications, 30 to 40 frames per second is the conventional standard. However, for control applications, for example in visual inspection where a reject mechanism is actuated if a product exhibits some defect, the processing rate can be lower. In any case, one of the features of a generalized Kalman filter for imaging is the ability to select response time and adjust accuracy to meet deadline.

One of the challenges in implementing real-time image processing involves the high computational intensity of the algorithm. In the case of Kalman filtering each pixel separately, this cost is quite high. Fortunately, commercial pixel processors, where one processor is assigned per pixel, are available. Furthermore, inexpensive, scalable structures such as the field-programmable gate array are becoming increasingly the platform of choice for imaging applications. Such platforms can easily support the object-oriented approach because of the high-level language support that they provide [11].

But regardless of the underlying hardware platform, unfortunately, there is no way that any programming language (at least not any "real" programming language, used conventionally), object-oriented or not, can guarantee response time in an implemented system [12]. As with any real application the system must be built and the timings verified through measurements. However, the use of the object-oriented approach and the proposed class library allows for easier extension of any working solution, with wider applicability.

6. Conclusions

In this application note we presented a brief review of the Kalman filter, its use in imaging applications, a set of recipes for constructing real-time imaging filters, and a performance analysis. We also described the use of object-oriented design techniques to exploit the benefits of such design in any application domain, and discussed various real-time aspects.

Acknowledgements

We are indebted to Dr. Colin Neill of Penn State University for discussions on object-oriented design and for providing the object-oriented Kalman filter design class.

References

[1] Laplante PA, Neill C. A class of Kalman filters for real-time image processing. Proceedings of the real-time imaging conference. Santa Clara: SPIE; January 2003, p. 22–9.

[2] Schneider F, Easterbrook SM, Callahan JR, Holzmann GJ. Validating requirements for fault tolerant systems using model checking. Third IEEE conference on requirements engineering. CO: Colorado Springs; April 6–10, 1998.

[3] Kalman RE. A new approach to linear filtering, prediction problems. Transactions of the ASME Journal of Basic Engineering 1960:35–45.

[4] Biemond J, Riesek J, Gerbrands JJ. A fast Kalman filter for images degraded by both blur and noise. IEEE Transactions on Acoustics, Speech, and Signal Processing, 1983;31(5;): 1248–56.

[5] Xie X, Sudhakar R, Zhuang H. Real-time eye feature tracking from a video image sequence using Kalman Filter. IEEE Transactions on Systems, Man, and Cybernetics 1995;25(12): 1568–77.

[6] Lippiello V, Siciliano B, Villani L. A new method of image features pre-selection for real-time pose estimation based on Kalman filter. Proceedings of the 2002 IEEE/RSJ international

conference on intelligent robots and systems, EPFL, Lausanne, Switzerland; October 2002, p. 372–7.
[7] Turney RD, Resa AM, Delva JGR. FPGA implementation of adaptive temporal Kalman filter for real time video filtering, March 15, 1999, white paper, available at www.xilinx.com/products/logiccore/dsp/temporal_kalman_fltr.pdf, last accessed November 29, 2002.
[8] Kuo C-M, Chao C-P, Hsieh C-H. An efficient motion estimation algorithm for video coding using Kalman filter. Real-Time Imaging 2002;8:253–64.
[9] Brailean JC, Kleihorst RP, Efstratiadis S, Katsaggelos AK, Lagendijk RL. Noise reduction filters for dynamic image sequences: a review. Proceedings of the IEEE 1995;83(9):1272–92.
[10] Gamma E, Helm R, Johnson R, Vlissides J. Design patterns: elements of reusable object-oriented software. Reading: Addison-Wesley; 1994.
[11] Xilinx Inc. Synopsys (XSI) synthesis and simulation design guide. San Jose, CA: Xilinx, Inc.; 1997.
[12] Laplante PA. Real-time systems design and analysis: an engineer's handbook, 3rd ed. New York: Wiley; 2003.