Under consideration for publication in Formal Aspects of Computing

Quantitative Verification of Kalman Filters

Alexandros Evangelidis and David Parker

School of Computer Science, University of Birmingham
Birmingham, United Kingdom

Abstract. Kalman filters are widely used for estimating the state of a system based on noisy or inaccurate sensor readings, for example in the control and navigation of vehicles or robots. However, numerical instability or modelling errors may lead to divergence of the filter, leading to erroneous estimations. Establishing robustness against such issues can be challenging. We propose novel formal verification techniques and software to perform a rigorous quantitative analysis of the effectiveness of Kalman filters. We present a general framework for modelling Kalman filter implementations operating on linear discrete-time stochastic systems, and techniques to systematically construct a Markov model of the filter's operation using truncation and discretisation of the stochastic noise model. Numerical stability and divergence properties are then verified using probabilistic model checking. We evaluate the scalability and accuracy of our approach on two distinct probabilistic kinematic models and four Kalman filter implementations.

1. Introduction

Estimating the state of a continuously changing system based on uncertain information about its dynamics is a crucial task in many application domains ranging from control systems to econometrics. One of the most popular algorithms for tackling this problem is the *Kalman filter* [Kal60], which essentially computes an optimal state estimate of a noisy linear discrete-time system, under certain assumptions, with the optimality criterion being defined as the minimisation of the mean squared estimation error.

However, despite the robust mathematical foundations underpinning the Kalman filter, developing an operational filter in practice is considered a very hard task since it requires a significant amount of engineering expertise [May82a]. This is because the underlying theory makes assumptions which are not necessarily met in practice, such as there being precise knowledge of the system and the noise models, and that infinite precision arithmetic is used [GA14, Sim06]. Avoidance of numerical problems such as round-off errors also remains a prominent issue in filter implementations [Gib11, GA14, Sim06, ZM15]. Finally, tuning a filter in order to compensate for modelling errors provides a further challenge for practical applications. Our goal in this paper is to develop formal verification techniques that allow the detection of possible failures in specific Kalman filter implementations arising from such issues.

The Kalman filter repeatedly performs two steps. The first occurs before the next measurements are available and relies on prior information. This is called the *time update* (or prediction step) and propagates the "current" state estimate forward in time, along with the uncertainty associated with it. These variables are defined as the a priori state estimate $\hat{x}^-$ and estimation-error covariance matrix P^-, respectively. The second

Correspondence and offprint requests to: Alexandros Evangelidis, e-mail: a.evangelidis@cs.bham.ac.uk

step is called the *measurement update* (or correction step) and occurs when the next state measurements are available. The Kalman filter then uses the newly obtained information to update the a priori $\hat{x}^-$ and P^- to their a posteriori counterparts, denoted $\hat{x}^+$ and P^+, which are adjusted using the so-called optimal *Kalman gain* matrix K.

The part of the filter that could hinder its numerical stability, and so cause it to produce erroneous results, is the propagation of the estimation-error covariance matrix P in the time and measurement updates [BSL01, GA14, May82a]. This is because the computation of the Kalman gain depends upon the correct computation of P and round-off or computational errors could accumulate in its computation, causing the filter either to diverge or slow its convergence [GA14]. While, from a mathematical point of view, the estimation-error covariance matrix P should maintain certain properties such as its symmetry and positive semidefiniteness to be considered valid, subtle numerical problems can destroy those properties resulting in a covariance matrix which is theoretically impossible [KBS71]. Out of the two steps in which the filter operates, the covariance update in the measurement update is considered to be the *"most troublesome"* [May82a]. In fact, the covariance update can be implemented using three different but algebraically equivalent forms, and all of them can result in numerical problems [BSL01].

To address the aforementioned challenges, we present a general framework for modelling and verifying different filter implementations operating on linear discrete-time stochastic systems. It consists of a modelling abstraction which maps the system model whose state is to be estimated and a filter implementation to a discrete-time Markov chain (DTMC). This framework is general enough to handle the creation of various different filter variants. The filter implementation to be verified is specified in a mainstream programming language (we use Java) since it needs access to linear algebra data types and operations.

Once the DTMC has been constructed, we verify numerical stability properties of the Kalman filter being modelled using properties expressed in a reward-based extension [FKNP11] of the temporal logic PCTL (probabilistic computation tree logic) [HJ94]. This requires generation of non-trivial reward structures for the DTMC computed using linear algebra computations on the matrices and vectors used in the execution of the Kalman filter implementation. The latter is of more general interest in terms of the applicability of our approach to analyse complex numerical properties via probabilistic model checking.

We have implemented this framework within a software tool called VerFilter, built on top of the probabilistic model checker PRISM [KNP11]. The tool takes the filter implementation, a description of the system model being estimated and several extra parameters: the maximum time the model will run, the number of intervals the noise distribution will be truncated into, and the numerical precision, in terms of the number of decimal places, to which the floating-point numbers which are used throughout the model will be rounded.

The decision to let the user specify these parameters is particularly important in the modelling and verification of stochastic linear dynamical systems, where the states of the model, which comprise of floating-point numbers, as well as the labelling of the states, are the result of complex numerical linear algebra operations. Lowering the numerical precision usually means faster execution times at the possible cost of affecting the accuracy of the verification result. This decision is further motivated by the fact that many filter implementations run on embedded systems with stringent computational requirements [Sim06], and being able to produce performance guarantees is crucial.

We demonstrate the applicability of our approach by verifying four distinct filter implementations: the conventional Kalman filter, the Carlson-Schmidt square-root filter, the Bierman-Thornton U-D filter and the steady-state Kalman filter. This allows us to evaluate the trade-offs between different variants. For the system models, we use *kinematic state models*, since they are used extensively in the areas of navigation and tracking [BSL01, LJ03]. We evaluate our approach with two distinct models. We demonstrate that our approach can successfully analyse a range of useful properties relating to the numerical stability of Kalman filters, and we evaluate the scalability and accuracy of the techniques.

This paper is an extension of the conference paper [EP19]. Here, we: provide a more thorough treatment of the probabilistic model constructed for verification; expand the scope of our analysis, studying a new class of properties (modelling errors); and evaluate two new implementations (the U-D and steady-state filters).

Related Work. Studies of Kalman filter numerical stability outside of formal verification are discussed above and in more detail in the next section. To the best of our knowledge, there is no prior work applying probabilistic model checking to the verification of Kalman filters. Perhaps the closest is the use of non-probabilistic model checking on a variant of the filter algorithm in the work of [MGB03], which applied model checking to target estimation algorithms in the context of anti-missile interception. In general, applying

formal methods to state estimation programs is an issue which has concerned researchers over the years. For example, [RVWL03, WS04] combined program synthesis with property verification in order to automate the generation of Kalman filter code based on a given specification, along with proofs about specific properties in the code. Other work relevant to the above includes [RG03], which used the language ACL2 to verify the loop invariant of a specific instance of the Kalman filter algorithm.

2. Preliminaries

We start with background material on Kalman filters, including the difficulties that arise when implementing them and some of the variants that have been proposed to address them. We also briefly discuss probabilistic model checking and PRISM.

2.1. The Kalman filter

The Kalman filter tracks the state of a linear stochastic discrete-time system of the following form:

$$x_{k+1} = F_k x_k + w_k \qquad z_k = H_k x_k + v_k \tag{1}$$

where x_k is the $(n \times 1)$ system state vector at discrete time instant k and F_k is a square $(n \times n)$ state transition matrix, which relates the system state vector x_k between successive time steps in the absence of noise. In addition, z_k is the $(m \times 1)$ measurement vector and H_k is the $(m \times n)$ measurement matrix, which relates the measurement with the state vector. Finally, w_k and v_k represent the process and measurement noises, with covariance matrices Q_k and R_k, respectively. Given the above system and under certain assumptions, the Kalman filter is an optimal estimator in terms of minimising the mean squared estimation error.

Below, we explain the working of the filter in more detail. The notation $\hat{x}_k^-$ stands for the *a priori* state estimate at time step k, with the "hat" symbol denoting the estimate, and the minus superscript denoting that the measurements at time k have not been processed yet. Mathematically, it can be written as $\mathbb{E}[x_k \mid z_1, z_2, z_3, \ldots, z_{k-1}]$, which is the conditional expectation of the random variable x_k given the measurements up to and including time step $k-1$. Analogously, $\hat{x}_k^+$ denotes the *a posteriori* state estimate at time k, meaning that in this case the measurements at time step k have been taken into account in the estimation of x_k, and can be written as $\mathbb{E}[x_k \mid z_1, z_2, z_3, \ldots, z_k]$.

An important variable in the Kalman filter is the so-called *Kalman gain matrix* K_k. In the scalar case, this can be thought of as a weighting factor whose entries take values in the interval $[0, 1]$ and which adjusts the a priori state estimate according to how much "trust" or "belief" is placed on the newly obtained measurements. For example, if the Kalman gain is zero, this would indicate that there is no uncertainty associated with the a priori estimate $\hat{x}_k^-$ and so the a posteriori state estimate $\hat{x}_k^+$ would be identical.

The task of the Kalman filter is to find the optimal Kalman gain matrix in terms of minimising the sum of estimation error variances or the mean squared estimation error. These can be obtained by summing the elements of the main diagonal (trace) of the a posteriori estimation-error covariance matrix P_k^+. Now, in order to solve for the optimal Kalman gain at time k, we differentiate the trace of P_k^+ with respect to K_k and then set its derivative equal to zero, to obtain the following Kalman gain equation: $K_k = P_k^- H_k^T (H_k P_k^- H_k^T + R_k)^{-1}$. In this equation, the computation inside the parenthesis is the covariance of the *innovation* (or measurement residual), which represents the difference between the actual and the predicted system measurement. This innovation covariance matrix is usually referred to in the literature as S, and is typically calculated as a separate step before the Kalman gain computation.

The conventional Kalman filter estimation process begins by initialising:

$$\hat{x}_0^+ = \mathbb{E}[x_0] \qquad P_0^+ = \mathbb{E}[(x_0 - \hat{x}_0^+)(x_0 - \hat{x}_0^+)^T] \tag{2}$$

It then proceeds by iterating between two steps. The time update is given as:

$$\hat{x}_{k+1}^- = F_k \hat{x}_k^+ \qquad P_{k+1}^- = F_k P_k^+ F_k^T + Q_k \tag{3}$$

The measurement update is given as:

$$y_{k+1} = z_{k+1} - H_{k+1}\hat{x}^-_{k+1} \qquad S_{k+1} = H_{k+1}P^-_{k+1}H^T_{k+1} + R_{k+1} \tag{4}$$

$$K_{k+1} = P^-_{k+1}H^T_{k+1}S^{-1}_{k+1} \tag{5}$$

$$\hat{x}^+_{k+1} = \hat{x}^-_{k+1} + K_{k+1}y_{k+1} \qquad P^+_{k+1} = (I - K_{k+1}H_{k+1})P^-_{k+1} \tag{6}$$

2.2. Divergence of Kalman Filters

Even though, from a theoretical point of view, the Kalman filter may give the impression that it will produce an optimal estimate, this might not be the case when applied to realistic situations. The reason is that the theory behind the Kalman filter often makes assumptions which could not be met in practice, such as the assumption that there is precise knowledge of the system and noise matrices, or that the numbers can be represented with infinite precision [Sim06]. Specifically, it is assumed that the representation of the model used in the Kalman filter algorithm corresponds exactly to the system being modelled [Gib11]. As a result, the approximation of the actual process which is expressed by the state space equations might lead to a phenomenon known as *divergence* [May82a] where the calculated state estimate from the Kalman filter does not represent the actual state of the process accurately.

Consequently, there is a high potential for the estimated error covariance to be inconsistent with the actual error covariance [BN76], and as a result the accumulated estimation errors will hinder the effectiveness and accuracy of the estimation procedure. As an aside, it should be pointed out that model approximations of the actual process are not necessarily the result of an erroneous analysis, but they can also arise in a deliberate manner as a way to restrict the computational requirements of the Kalman filter [Jaz70] (e.g., reducing the order of the model), and as a result it becomes quite significant to quantitatively analyse this "degree of suboptimality"[BH12]. The *precise* identification of the conditions which could result in divergence is not a straightforward task, mainly because the whole concept of divergence is considered somewhat nebulous, encompassing a whole spectrum of errors and is regarded more or less as a qualitative concept [AM12].

The causes of divergence are mostly attributed to either modelling or numerical errors [BSL01, BH12]. This is because the process of developing the Kalman filter in a computer implicitly means to "transfer" the rich mathematical theory which surrounds the Kalman filter to the computer's digital representation. As a result, when the outcome of this process is erroneous, it is either because the constraints imposed on us by the computer (e.g., finite-precision arithmetic) and the "digital" behaviour of the Kalman filter does not match the "theoretical" behaviour or because errors have been introduced during the modelling phase [CG74]. In particular, divergence because of numerical instability of the numerical algorithms used and because of modelling errors, started to become apparent once the Kalman filter was gaining popularity in a growing number of engineering applications [VD86]. In the next sections, we describe what is actually meant by numerical instability and modelling errors in the context of the Kalman filter.

2.3. Numerical Instability of the Kalman filter

In order for P to be statistically valid it must be (symmetric) positive definite. Briefly, this means that all of its eigenvalues are positive real numbers. This is for two reasons. First, from a modelling perspective, if its eigenvalues were zero, this would translate to a filter which completely trusts its estimates and consequently would avoid taking into account the subsequent measurements, placing all of its "belief" in the system model [BSL01]. Second, from a numerical stability perspective, it does not suffice for the eigenvalues of P to be greater than zero, because if they are in close proximity to zero, then round-off errors could cause them to become negative, rendering it totally invalid [AM12, GA14, Kai80].

In fact, the three equivalent forms to express the covariance measurement update are all susceptible to numerical errors [BSL01] and cannot guarantee the numerical stability of P. For example, the covariance update $P^+_k = (I - K_kH_k)P^-_k$ is generally not preferred because it is too sensitive to round-off errors [BSL01], which means neither the symmetry nor the positive definiteness of P_k can be guaranteed. That is because this update takes the product of nonsymmetric and symmetric matrices, a form which has been characterised as undesirable [May82a].

Alternatively, changing the covariance measurement update equation to $P^+_k = P^-_k - K_kS_kK^T_k$ could

potentially pose a *"serious numerical problem"* [May82a], such as P_k losing positive definiteness. Finally, while *Joseph's stabilised form* [BJ68], given by $P_k^+ = (I - K_k H_k)P_k^-(I - K_k H_k)^T + K_k R_k K_k^T$, is considered to preserve the numerical robustness of P^+, it is also not totally insensitive to numerical errors [BSL01]. An additional disadvantage is the high computational complexity, which is $O(n^3)$ [GA14, May82a], since the number of arithmetic operations such as additions and multiplications is considerably higher compared to the simpler form.

To ameliorate these numerical problems, an alternative form of expressing the covariance time and measurement updates is using so-called *square-root filters*. These are generally considered superior to conventional filter implementations mainly because of their ability to increase the numerical stability of the propagation of the estimation-error covariance matrix P, and have often been described as outstanding [KBS71, May82a]. It should be noted that the term square-root filter is mostly used to refer to the measurement update of the Kalman filter algorithm, since it is this part that can cause numerical problems [Gib11]. They were motivated by the need for increased numerical precision because of word lengths of limited size in the 1960s [Sim06] and by the concern with respect to the numerical accuracy of P in the measurement update of the Kalman filter equations [Gib11]. Potter [Bat64] proposed the idea of square-root filters and this idea has evolved ever since. The idea, which was limited to noiseless systems, is that P is factored into its square root C, such that $P = CC^T$, and as a result C is propagated through the measurement update equations, instead of P. Replacing P with its square-root factor C has the effect of doubling the numerical precision of the filter, thus making it particularly suitable for matrices which are not well-conditioned or when increased precision cannot be obtained from the hardware [Gib11, GA14, May82a, Sim06].

2.4. Divergence Due to Modelling Errors

Divergence detection falls under the wider scope of estimation theory, and more specifically to what is often referred to as *consistency evaluation* of estimators. An estimator is said to be *consistent* when its estimates will converge to the true value as the sample size increases [BSL01], or informally when its estimates become increasingly accurate over time. Also, it is important to consider the context under which the Kalman filter operates. For example, the *consistency tests* that can be applied to a Kalman filter operating in a simulation-based environment cannot be applied to the situation where the Kalman filter operates in real-time, and processes real-time data [BSL01]. This is because devising statistical tests based on the *actual* estimation errors to assess the consistency of a Kalman filter is only applicable in simulations, where the true state of the system is known.

A popular way to deal with divergence problems, once they are detected, is called *filter tuning*. This works by adjusting the process noise covariance matrix Q, which the Kalman filter uses to represent the unknown effects affecting the system whose state is to be estimated. This can be thought of as injecting "pseudo-noise" and is considered one of the hardest and most challenging tasks in developing an operational filter [ACCK93, GSDW16, CD16], because it encompasses all the unknown effects that perturb the process model of interest.

It can also be thought of as a modelling error compensation technique [May82b, BSL01, BH12, Sim10, Jaz70, AM12], and can be applied to a self-adaptive context, where it is sometimes referred to as *self-tuning* [May82a]. This context is often referred to as adaptive filtering which means that the Kalman filter adjusts its noise levels dynamically according to some performance criterion. Furthermore, filter tuning can also happen in the design phase, e.g., using a simulation-based environment and an iterative search to select parameter values that will achieve the best possible estimation result [May82a].

In this work, we instead focus on a quantitative analysis of the *innovation* (or *measurement residual*), which represents the difference between the actual and the predicted system measurement. This is considered a "*prime indicator*" of filter divergence [AM12]. The *consistency criterion* of the innovation sequence we consider is that its computed covariance from the Kalman filter (e.g., actual statistics) is consistent with the design statistics [AM12]. In other words, we want to verify that the actual statistical properties are consistent with the theoretical statistical properties.

2.5. Kalman Filter Variants

In this paper, we analyse the conventional Kalman filter and three other variants: the *Carlson-Schmidt filter*, the *Bierman-Thornton U-D Filter* and the *steady-state Kalman filter*.

The Carlson-Schmidt filter. This is a form of a square-root filter which relies on the decomposition of P into its Cholesky factors in the time and measurement update equations. The Carlson part of the filtering algorithm, originally given by Carlson [Car73], corresponds to the measurement update, while the Schmidt part corresponds to the time update of the Kalman filter equations. Carlson's algorithm is capable of handling noise and, like Potter's algorithm, processes measurements as scalars. It factors P into the product of an upper-triangular Cholesky factor and its transpose such that $P = CC^T$.

Unlike Potter's initial square-root filter where the factor C is not required to be triangular, in Carlson's square-root implementation, the Cholesky factor C is an upper-triangular matrix. Maintaining C in upper-triangular form has been shown to provide several advantages in terms of storage and computational speed compared to Potter's algorithm [Car73, May82a]. While the choice between a lower and upper-triangular Cholesky factor C is arbitrary [May82a], Carlson motivated the preference to choose an upper-triangular Cholesky factor by the fact that in the time update part of the algorithm, fewer retriangularisation operations are required especially when someone designs a filter to be applied in a tracking or in a navigation problem [Car73].

The Bierman-Thornton U-D Filter. Known as the U-D filter for short, this is one of the most widely used Kalman filter variants [DZ18], which despite its appearance in the early 1970s, due to its numerical accuracy, stability and computational efficiency is *"still the dominant type of factored filter algorithm"* [Gib11]. It is worth noting that in the literature there seems to be some ambiguity whether the U-D filter is considered a square-root filter or not, since there are authors who classify it under the broader category of square-root filters and others who do not [GA14, Sim06]. Specifically, the "Bierman" part of the filtering algorithm, originally given by Bierman [Bie75] corresponds to the observational update, while the "Thornton" part given by Thornton [Tho76] corresponds to the time update of the Kalman filter equations.

Bierman's covariance update, the "actual" U-D filter relies on the decomposition of P into the following matrix product: $P = UDU^T$, where U is a unit upper-triangular and D is a diagonal matrix, respectively [Bie77], a procedure which is often referred to as a modified Cholesky decomposition and the U, D factors as modified Cholesky factors [GA14]. Unlike Carlson's method it does not require computing scalar square roots for every incorporated measurement [May82a, Bie77, Tho76], thus making it rather suitable for problems where the number of variables comprising the state space is large [GA14]. Furthermore, Bierman's algorithm in a manner similar to Carlson's method promotes the use of upper-triangular matrices for the same reasons of computational efficiency. Thornton's algorithm provides an alternative for the conventional Kalman filter's time-update equations as it propagates the U and D factors, instead of P, forward in time, using the numerically stable Modified Weighted Gram-Schmidt (MWGS) orthogonalisation algorithm [Tho76].

The steady-state Kalman filter. In the steady-state Kalman filter, the time-varying variables of the conventional filter, such as the a priori estimation-error covariance and the Kalman gain, are replaced with their steady-state counterparts. As a clarification, the term "steady-state" refers to the Kalman gain that is in steady state [Sim06]. It turns out that, in order to compute the steady-state Kalman gain, the steady-state solution of the a priori error covariance must be found, since the Kalman gain depends on it. This limiting solution can be expressed as the discrete-time algebraic Riccati equation, and the first step before any attempt is made to solve it, is to verify that the necessary conditions hold.

In particular, one can employ the so-called observability and controllability tests, which rely on certain theorems, and provide a way of identifying whether a solution exists or not. The first test states that the matrix pair F, H must be completely observable, which means that the observability matrix M must be of full rank (i.e., $\rho(M) = n$). Once the observability test passes, the controllability test takes place, which seeks to evaluate whether the matrix pair F, C is completely controllable. The procedure followed is relatively similar to the previous case, with the only difference being the extra computation needed for the C matrix, which is a Cholesky factor of the process noise covariance matrix Q.

Pre-computing the steady-state gains results in faster computations. On the other hand, using fixed Kalman gains implies transforming the Kalman filter from a time-varying filter to a time-invariant one, which could hinder the accuracy of the estimates produced.

2.6. Probabilistic Model Checking and PRISM

Our Kalman filter analysis technique is based on *probabilistic model checking*, which is an automated *quantitative verification* technique that seeks to establish *quantitative* properties which relate to the specification of a probabilistic system, with some degree of mathematical certainty [Kwi07]. In order to perform probabilistic model checking, two inputs are required: i) a probabilistic model, which is a representation of a stochastic system; and ii) a specification, usually expressed in probabilistic temporal logic [KNP07]. Therefore, *quantitative verification*, and probabilistic model checking in particular, can be thought of as a generalisation of conventional model checking techniques.

PRISM [KNP11] is a probabilistic model checking tool which supports the construction and formal quantitative analysis of various probabilistic models, including discrete-time Markov chains, continuous-time Markov chains and Markov decision processes. In our work, for the verification of Kalman filters, we use discrete-time Markov chains, which are well suited to modelling systems whose states evolve probabilistically, but without any nondeterminism or external control. They are therefore appropriate here, where we want to verify Kalman filter executions, whose outcomes are probabilistic.

Formally, a discrete-time Markov chain is defined as follows.

Definition 2.1. A *discrete-time Markov chain* is a tuple $M = \langle S,P,AP,L \rangle$ where:

- S is a finite set of states;
- $P : S \times S \to [0,1]$ is a transition probability matrix;
- AP is a finite set of atomic propositions;
- L: $S \to 2^{AP}$ is a labelling function.

Each element $P(s,s')$ of the transition probability matrix gives the probability of moving from state s to s'. We require that $\sum_{s' \in S} P(s,s') = 1$ for all states $s \in S$. If we denote the state of the Markov chain at a particular time step k by X_k, then the transition probabilities can be defined as $Pr(X_{k+1} = s' \mid X_k = s) = P(s,s')$ for any $s, s' \in S$. The set of atomic propositions AP describes properties of interest which can be either true or false in the Markov chain's states, and the labelling function L maps states to the atomic propositions in the set AP.

Properties of discrete-time Markov chains are specified in PRISM using an extension [FKNP11] of the temporal logic PCTL (probabilistic computation tree logic) [HJ94]. A typical property is $\mathtt{P}_{\bowtie p}[\psi]$, where $\bowtie \in \{\leq, <, >, \geq\}$ and $p \in [0,1]$, which asserts that the probability of event ψ occurring meets the bound $\bowtie p$. Events are specified using temporal operators, e.g., $\mathtt{F}\,\Phi$ means that along a path state formula Φ *eventually* holds and $\mathtt{G}\,\Phi$ means that formula Φ *always* holds in every state of the path.

The states of probabilistic models can also be annotated with *rewards* (or costs). A *reward structure* is a labelling function which assigns a non-negative rational value to a state, and model checking a reward-based property usually refers to the computation of its expected value [KNP07]. Such properties are specified using the reward operator $\mathtt{R}$. For example, a state s satisfies a reward-based property of the form $\mathtt{R}^{r}_{\bowtie q}[\mathtt{C}^{\leq \mathtt{k}}]$ if from state s, the expected reward r *cumulated* after k time steps satisfies the bound $\bowtie q$. Moreover, a state s satisfies a property of the form $\mathtt{R}^{r}_{\bowtie q}[\mathtt{I}^{=\mathtt{k}}]$ if, from state s, the expected *instantaneous* reward r at time step k meets the bound $\bowtie q$. Finally, the property $\mathtt{R}^{r}_{\bowtie q}[\mathtt{F}\,\Phi]$ is true in a state s if the expected cumulated reward r before reaching a state in which the formula Φ holds meets the bound $\bowtie q$ [KNP07].

In addition, PRISM supports numerical properties such as $\mathtt{P}_{=?}[\mathtt{F}\ \mathit{fail}]$, which means "what is the probability for the system to eventually reach a state labelled with *fail*?". PRISM also allows a wide range of other such properties to be specified and analysed. See, e.g., [FKNP11] for full details of the syntax and semantics of the PRISM property specification language.

3. Quantitative Verification of Kalman Filters

In this section, we describe our approach to modelling and verifying Kalman filter implementations. This is based on the construction and analysis of a probabilistic model, a discrete-time Markov chain (DTMC), representing the behaviour of a particular Kalman filter executing in the context of estimating the state of a linear stochastic discrete-time system. The probabilistic model is automatically constructed based on a specification of the filter and the system whose state it is trying to estimate. System properties required to

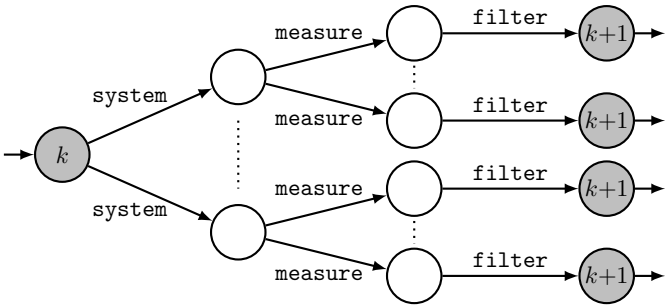


Fig. 1. Illustration of the evolution of the DTMC from time step k to k+1.

analyse numerical stability or modelling errors are then verified using probabilistic model checking queries. We describe each of these phases in the following sections.

3.1. Constructing Probabilistic Models of Kalman Filter Execution

We define a high-level modelling abstraction which can be instantiated to construct models of various different Kalman filter implementations. The modelling abstraction comprises three components: the first and second correspond to the system and measurement models along with their associated noise distributions; the third is the Kalman filter implementation itself used to estimate the state of the system model in the presence of uncertainty. The first two of these are defined mathematically along the lines described in Section 2.1. The third is specified in detail using a mainstream programming language, since it requires linear algebra data types and operations. Our implementation (see Section 4) uses Java and associated numerical libraries.

DTMC states and transitions. The DTMC represents the *dynamic* evolution of these components and the interactions that occur between them, imitating a real-time tracking scenario. The variables which define the DTMC's states correspond to the system, measurement and filter models. More precisely, each state of the Markov chain takes the form $(k, x, z, \hat{x}, f)$. The first element k is the time step. The next two elements are: x, the true state of the system model (i.e., the system whose state is being estimated); and z, the measured state of the system. The remaining two elements represent the state of the Kalman filter: $\hat{x}$ is the current a posteriori estimate of the system state; and f comprises any others variables used by the filter to keep track of its estimate (for example, for the conventional Kalman filter, this is the estimation-error covariance matrix P; for the U-D filter, these are the two matrices U and D). Typically, the Kalman filters make use of several other auxiliary variables to compute each successive state estimate, but these do not need to be stored in the DTMC.

Each transition of the DTMC represents a single discrete-time step comprising the evolution of the system model and its measurement, perturbed by different noise values and the corresponding updates to the filter. Moving from time step k to time step $k+1$, we have three intermediate updates:

- $x_{k+1} := \texttt{system}(x_k)$
- $z_{k+1} := \texttt{measure}(x_{k+1})$
- $(\hat{x}_{k+1}, f_{k+1}) := \texttt{filter}(\hat{x}_k, f_k, z_{k+1})$

The first two functions, `system` and `measure`, correspond to Equation (1) in Section 2.1. Each are subject to noise and so this represents a probabilistic state update. The third function, `filter`, captures the behaviour of the Kalman filter and will differ for each implementation considered. Notice that `filter` uses the previous state $(\hat{x}_k, f_k)$ of the filter and the newly measured system state z_{k+1}, but it does not have access to the "true" state of the system model x_{k+1}. Unlike the first two functions, the update of the filter state is deterministic.

For the conventional Kalman filter, f comprises the estimation-error covariance matrix P. The function `filter` corresponds to the equations in Section 2.1 used to define $\hat{x}^+_{k+1}$ and P^+_{k+1}. Intermediate variables, such as y and P^-, are used in these computations but not stored in the DTMC state since their values are not needed in subsequent time steps. These variables will differ for other types of filter and for specific

Table 1. Intervals according to the granularity level.

`gLevel`	**Intervals**
2	$[-\infty, \mu]$, $[\mu, +\infty]$
3	$[-\infty, -2\sigma]$, $[-2\sigma, +2\sigma]$, $[+2\sigma, +\infty]$
4	$[-\infty, -2\sigma]$, $[-2\sigma, \mu]$, $[\mu, +2\sigma]$, $[+2\sigma, +\infty]$
5	$[-\infty, -2\sigma]$, $[-2\sigma, -\sigma]$, $[-\sigma, +\sigma]$, $[+\sigma, +2\sigma]$, $[+2\sigma, +\infty]$
6	$[-\infty, -2\sigma]$, $[-2\sigma, -\sigma]$, $[-\sigma, \mu]$, $[\mu, +\sigma]$, $[+\sigma, +2\sigma]$, $[+2\sigma, +\infty]$

implementations. For example, in a square root filter implementation, P^+ can be either reconstructed from the Cholesky factor C^+, or not by propagating C^+ in each time step. We show an example of the DTMC for the conventional Kalman filter later in this section.

Figure 1 illustrates how the state of the DTMC evolves in a single time step; when we construct the model, we do so over a finite horizon for a specified number of time steps. The grey circles are states of the DTMC, from time step k on the left-hand side to time step $k+1$ on the right-hand side. The white circles are intermediate states which we create when constructing the Markov chain but then collapse to save storage. The intermediate transitions between these states correspond to the three updates explained above, and are labelled with the corresponding functions `system`, `measure` and `filter`. Notice that there are no cyclic dependencies in these updates, so the DTMC takes the form of a directed acyclic graph. As mentioned above, the `filter` update is deterministic, so is represented by a single transition in Figure 1. The transitions labelled with `system` and `measure` represent the result of stochastic noise, with the different possible outcomes represented by the branches of the DTMC. These are *discretised* into a finite number of possible values, which we now describe in more detail.

Noise discretisation. The number of outgoing transitions and their probability values are determined by a *granularity level* of the noise, which we denote `gLevel`. The Gaussian distribution of the noise is discretised into `gLevel` disjoint intervals. The intervals used for each granularity level are shown in Table 1.

The measure used to determine these intervals is the standard deviation σ, which is a common practice in statistical contexts; see for example the so-called $68 - 95 - 99.7$ rule, which states that, assuming the data are normally distributed, then 68%, 95% and 99.7% of them will fall between one, two and three standard deviations of the mean, respectively. This statement can be expressed probabilistically as well by computing the cumulative distribution function (CDF) of a normally distributed random variable X, usually by converting it to its *standard* counterpart and using the so-called standard normal tables. While computing the probability that a noise value will fall inside an interval is relatively easy, the computation of its expected value is slightly more difficult. This is because we can choose to either truncate the distribution to intervals which contain the mean value of the distribution, which is the easier case, or to intervals which do not.

Usually, for those cases, one might use a simple heuristic such as dividing the sum of the two endpoints of the interval by two, which is actually quite common. However, this might not be representative of the actual expected value since it does not weigh the values lying inside the interval according to the corresponding value of the density correctly. In other words, since the mean is also interpreted as the *"centre of gravity"* of the distribution [BT08], in the case of truncated intervals which do not contain the mean, more accurate techniques are needed.

The probabilities of the Markov chain for a given granularity level are computed by first standardising the random variable, the noise in our case, and then evaluating its CDF at the two endpoints of the corresponding interval. Then, by subtracting them, we obtain the probability that it will fall within a certain interval.

Once the probabilities have been computed, it remains to find the expected value of the random variable for the corresponding intervals. In order to avoid the situation described earlier, and obtain the mean in a more accurate way, we have used the *truncated normal distribution* to compute the mean for the respective intervals. Formally, if a random variable X is normally distributed and lies within an interval $[a, b]$, where $-\infty \leq a \leq b \leq +\infty$, then X conditioned on $a < X < b$ has a truncated normal distribution. The probability density function (PDF) of a normal truncated random variable X is characterised by four parameters: (i-ii) the mean μ and standard deviation σ of the *original* distribution and (iii-iv) the lower and upper truncation points, a and b. Compactly, the mean value of the noise for a corresponding interval can be expressed as the

conditional mean, $E[X|a < X < b]$, given by the following formula [JKB94]:

$$E[X|a < X < b] = \mu + \sigma \frac{\phi(\frac{a-\mu}{\sigma}) - \phi(\frac{b-\mu}{\sigma})}{\Phi(\frac{b-\mu}{\sigma}) - \Phi(\frac{a-\mu}{\sigma})} \tag{7}$$

Note that in the expression above, ϕ and Φ denote the PDF and CDF of the standard normal distribution, respectively. Also note that the denominator has already been computed in the previous step, when the transition probabilities were computed. As a result, the computation of the transition probabilities and the conditional mean values for each of the corresponding intervals can be done in a unified manner.

Example. We illustrate the DTMC modelling of a conventional Kalman filter with an example. For this, we use a kinematic system model which is currently 5 metres away from the origin, moving with a velocity of 5 metres per second, and which is perturbed by noise having a normal distribution with $\mu = 0$ and $\sigma^2 = 0.025$. The sensor, which is assumed to be more accurate, is also perturbed by normally distributed noise with $\mu = 0$ and $\sigma^2 = 0.01$. The task of the Kalman filter is to estimate the system model's state vector, comprising both its position and velocity. It only directly observes (measures) the position, but will also be able to derive an estimate of the velocity.

In order to construct the Markov chain, assuming a granularity level of `gLevel`=3 and the noise values for μ and σ^2 defined earlier, the first step is to compute the transition probabilities for the given `gLevel` value, as has been described above. For example, for the interval $[-2\sigma, +2\sigma]$, the probability that a random variable (e.g., noise), denoted by X, will fall inside this interval is computed as follows:

$$P(-2\sigma \leq X \leq +2\sigma) = \Phi(X < +2\sigma) - \Phi(X < -2\sigma) \approx 0.9545 \tag{8}$$

Similarly, the probabilities for the other two intervals ($[-\infty, -2\sigma]$ and $[+2\sigma, +\infty]$) are both approximately 0.02275, since $P(X \leq -2\sigma) = P(X \geq +2\sigma)$. Next, we consider the computation of the mean noise value, which denotes the amount of noise by which the system model is perturbed. For the interval $[-2\sigma, +2\sigma]$, this is already known to us (it is 0), because the mean is contained in the interval considered. However, for the sake of this example it is also computed. This is expressed as the conditional mean of X given that it can only take values inside this interval, using Equation 7:

$$E[X|a < X < b] = \mu + \sigma \frac{\phi(\frac{a-\mu}{\sigma}) - \phi(\frac{b-\mu}{\sigma})}{\Phi(\frac{b-\mu}{\sigma}) - \Phi(\frac{a-\mu}{\sigma})} = 0 + 0.158 \frac{0.05 - 0.05}{0.977 - 0.022} = 0 \tag{9}$$

The mean noise values of the other two intervals are computed in a similar manner.

Figure 2 shows a fragment of the DTMC modelling this Kalman filter. For clarity, we use the same format as Figure 1, including intermediate states, shown in white, which are later removed, leaving only the grey states. As described above, each state of the Markov chain is of the form $(k, x, z, \hat{x}, f)$. More precisely, in this example: $\hat{x}$ is $\hat{x}_k^+$, the a posteriori state estimation vector at time k; and f, which represents any variables needed by the Kalman filter, is just the a posteriori estimation-error covariance matrix P_k^+. In the initial state, at time step $k = 0$, we have: a system state (position and velocity) of $x = [5\ 5]^T$; $z = 0$ and $\hat{x} = \hat{x}_k^+ = [0\ 0]^T$, since no measurement has yet occurred; and $f = P_k^+ = 10 \cdot I$, since there is no information about the correlation between state variables.

From this initial state, there are `gLevel`=3 transitions, representing the `system` updates described earlier. This updates x according to the left part of Equation (1), i.e.:

$$x_1 = Fx_0 + w = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} [5\ 5]^T + w \tag{10}$$

The three transitions are labelled with the probabilities 0.02275, 0.9545 and 0.02275, representing the cases where the process noise w is -0.375, 0 and $+0.375$, respectively, as computed above. Next, there are `gLevel`=3 transitions for the `measure` updates, setting z according to the right part of Equation (1):

$$z_1 = Hx_1 + v = [1\ 0]\ x_1 + v \tag{11}$$

The transition probabilities are as for the first step, but represent now the possible values for measurement noise v: -0.237, 0 and $+0.237$.

Lastly, the (deterministic) `filter` steps occurs, which update $\hat{x}^+$ and P^+ as described in Equations (2) to (6) in Section 2.1. At this point, the innovation y_k and the innovation covariance S_k are computed.

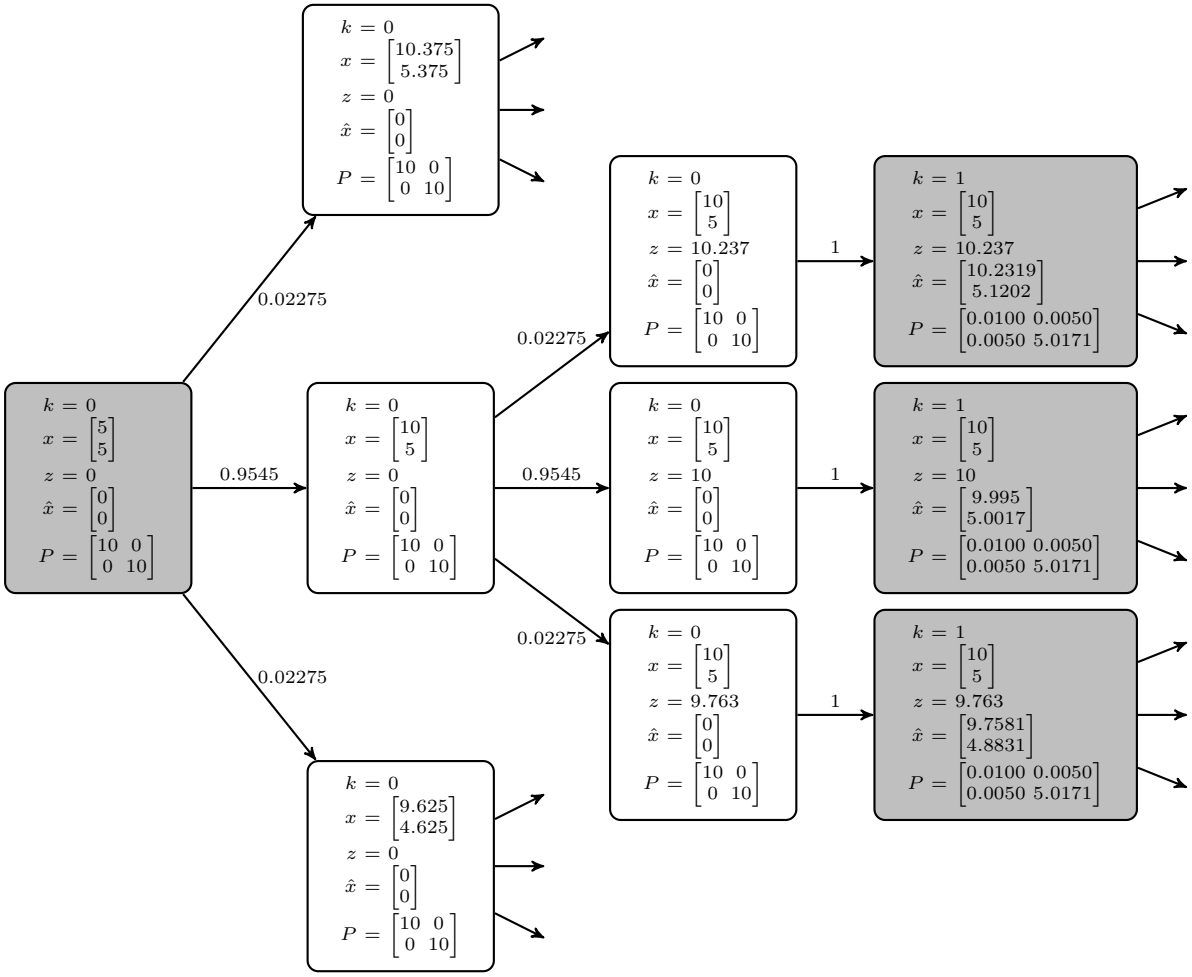


Fig. 2. Illustration of the construction of the DTMC modelling a conventional Kalman filter.

These variables will be needed for the computation of the Kalman gain K_k. The a priori estimation-error covariance matrix P_k^- is also computed and used here. The measurement noise covariance matrix R_k, also used to compute K_k, is here the scalar 0.01 since we assume normally perturbed noise with $\sigma^2 = 0.01$.

For the three example transitions shown in the right-hand part of Figure 2, the innovation y_1 is the same as the measurement z_1, since the filter's a priori state estimate $\hat{x}_1^-$ is zero; both take the value 10 since the measurement had zero noise on this branch. Consider the transition to the middle of the three grey states. The a posteriori state estimate is computed as follows:

$$\hat{x}_1^+ = \hat{x}_1^- + K_1 y_1 = \begin{bmatrix}0\\0\end{bmatrix} + \begin{bmatrix}0.99950\\0.50017\end{bmatrix}[10] = \begin{bmatrix}9.995\\5.0017\end{bmatrix} \tag{12}$$

and the a posteriori estimation-error covariance matrix is:

$$P_1^+ = (I - K_1 H)P_1^- = \left(\begin{bmatrix}1 & 0\\0 & 1\end{bmatrix} - \begin{bmatrix}0.99950\\0.50017\end{bmatrix}[1 \quad 0] \right) \begin{bmatrix}20.0083 & 10.0125\\10.0125 & 10.0250\end{bmatrix} = \begin{bmatrix}0.0100 & 0.0050\\0.0050 & 5.0171\end{bmatrix} \tag{13}$$

3.2. Verification of Numerical Stability

Next, we discuss how to capture numerical stability properties for our Kalman filter models (see the earlier summary in Section 2.3) using the probabilistic temporal logic [FKNP11] of the PRISM model checker [KNP11]. See Section 2.6 for a description of the logic and [FKNP11] for full details.

Verifying positive definiteness. In order to analyse this property, we perform an eigenvalue-eigenvector decomposition of P^+ into the matrices $[V, D]$. The eigenvalues are obtained from the diagonal matrix D, and their positivity is determined and used to label each state of the Markov chain accordingly: we use an atomic proposition *isPD* for states in which P^+ is positive definite. We can then specify the probability that the matrix remains positive definite for the duration of execution of the filter using the formula $\mathtt{P}_{=?}[\,\mathtt{G}\,\mathit{isPD}\,]$.

Examining the condition number of the estimation-error covariance matrix. The verification of certain numerical properties, such as those related to positive definiteness, is a challenging task and should be treated with caution. This is because, while convenient, focusing the verification on whether an event will occur or not, might not capture inherent numerical difficulties related to the numerical stability of state estimation algorithms. In other words, it does not suffice to check whether P^+ is positive definite or not by checking its eigenvalues because, as mentioned earlier, if they are in close proximity to zero, then round-off errors could cause them to become negative [GA14].

For example, it is often the case that estimation practitioners want to detect matrices that are close to becoming *singular*, a concept which is often referred to as *"detecting near singularity"* [Bie77]. In other words, since a positive definite matrix is nonsingular, one wants to determine the "goodness" of P^+ in terms of its "closeness" to singularity, within some level of tolerance, usually the machine precision [GA14]. A matrix is said to be *well-conditioned* if it is "far" from singularity, while *ill-conditioned* describes the opposite. In order to quantify the goodness of P^+, we use the so-called *condition number*, which is a concept used in numerical linear algebra to provide an indication of the sensitivity of the solution of a linear equation (e.g., $Ax = b$), [GA14, May82a]. In our case, this concept is used to obtain a measure of the goodness of P^+.

The condition number of P^+ is given as $\kappa(P^+) = \sigma_{max}/\sigma_{min}$, where σ_{max} and σ_{min} are the maximum and minimum singular values, respectively [Gib11, May82a]. These can be obtained by performing the singular value decomposition (SVD) of P^+. A "small" condition number indicates that the matrix is well-conditioned and nonsingular, while a "large" condition number indicates the exact opposite. Note that the smallest condition number is 1, when $\sigma_{max} = \sigma_{min}$.

We express this property as the formula $\mathtt{R}^{cond}_{=?}[\,\mathtt{I}^{=k}\,]$, which gives the expected value of the condition number after k time steps. We assign the condition number to each state of the DTMC using a reward function *cond* and we set k to be `maxTime`, the period of time for which we verify the filter.

Providing bounds on numerical errors. Another useful aspect of the condition number is that it can be used to obtain an estimate of the precision loss that numerical computations could cause to P^+. For instance, for a single-precision and a double-precision floating-point number format, the precision is about 7 and 16 decimal digits, respectively. Since our computations take place in the decimal number system, the logarithm of the condition number (e.g., $log_{10}(\kappa(P^+))$), gives us the ability to define more concretely when a condition number will be considered "large" or "small" [BS87, May82a, Sim06]. For example, $log_{10}(\kappa(P^+)) > 6$ and $log_{10}(\kappa(P^+)) > 15$ could indicate possible numerical problems in the estimation-error covariance computation and render P^+ ill-conditioned when implemented in a single and a double precision floating-point number format, respectively.

So, to verify this property we construct a closed interval whose endpoints will be based on the appropriate values of the numerical quantity of $log_{10}(\kappa(P^+))$. This lets us label states whose value of $log_{10}(\kappa(P^+))$ will fall within "acceptable" values in the interval, when, for instance, double precision is used. We then use the property $\mathtt{P}_{=?}[\,\mathtt{G}\,\mathit{isCondWithin}\,]$, in a similar fashion to the first property above, where *isCondWithin* labels the "acceptable" states. A probability value of less than 1 should raise an alarm that numerical errors may be encountered.

3.3. Verification of Modelling Error Compensation Techniques

In this section, we discuss how to to quantitatively analyse a second aspect of the effectiveness of Kalman filters: the detection of filter divergence in the presence of *modelling errors* (see the earlier discussion in

Section 2.4). Quantitative verification is used to verify *consistency* properties of Kalman filters. This can be used to drive the so-called *filter tuning* process, which optimises parameters of the filter, and produce guarantees on the chosen parameter values.

Verifying that the innovation is bounded by its variance. We consider two statistical tests. In the first, we verify that the fraction of times that the innovations (see Section 2.1) fall within two standard deviations ($\approx 1.96\sqrt{S}$) of the mean is at least 95%. In order to express this property as a temporal logic formula, we use a reward function *inRange*, which assigns a reward of 1 to the states in which the value of the innovation y_k falls within two standard deviations, and 0 otherwise.

Then, the reward-based property used to verify that 95% of the times the *inRange* reward falls within two standard deviations, is expressed as a formula of the form the $\mathtt{R}^{inRange}_{=?}[\,\mathtt{C}^{\leq maxTime}\,]$. The result of the reward is then compared to a *bound*, which is determined upon the maximum time the model runs. This bound is computed as the product of 0.95 with the `maxTime` variable, taking into consideration that measurements are processed as scalars. For instance, if the model runs for 20 time steps, the bound on the reward will be 19 (0.95×20), since $19/20 \approx 95\%$.

In our case, we assume that there are no measurements available during the first two time steps. The reason is due to the way we index time, starting from 0, since the first two states are used in the initialisation of the Markov chain. This means that when the model runs for 20 time steps (i.e. `maxTime`=20), the bound which should be exceeded by the result of the *inRange* reward, is ≈ 17. As a result, if the result of the formula is a value less than 17, then this is an indication that the Kalman filter is inconsistent.

Verifying that the magnitude of the innovations is proportionate to the covariance. The second statistical test we consider revolves around the concept of hypothesis testing, in which the *null* hypothesis H_0 describes the *status quo* or the default theory [Was10], and the *alternative* hypothesis H_1 the exact opposite. The overall concept of hypothesis testing is to test H_0 against H_1 and decide whether to reject H_0, based on the "amount" of evidence we acquire from the data, which in our case corresponds to the measurements emitted at each time step. It is important to note that there does not exist any symmetry between H_0 and H_1, meaning that failure to obtain the required evidence, does not imply that H_0 is *true*, rather than the fact that enough evidence has not been acquired to reject it. In our context, the null hypothesis H_0 states that the Kalman filter is consistent, or more specifically that the magnitude of the innovations y_k is proportionate to the covariance computed by the Kalman filter. Furthermore, the alternative hypothesis, H_1, describes the hypothesis that the Kalman filter is inconsistent.

The *statistic* that we structure the statistical test around is the so-called *normalised innovation squared* (NIS) [BSL01], given as: $\epsilon^2_{y_k} = y_k^T S^{-1} y_k$. Formally, a statistic is a function of the data, and since it is a random variable it has a certain distribution [Was10]. Note that $\epsilon^2_{y_k}$ conforms totally to the formal definition of a statistic, since $\epsilon^2_{y_k}$ is a random variable, it can be expanded to:

$$\epsilon^2_{y_k} = \left(z_k - H\hat{x}_k^-\right)^T \left[HP_k^- H^T + R_k\right] \left(z_k - H\hat{x}_k^-\right) \tag{14}$$

and it follows a χ^2 distribution. However, since the information at a particular time instant k might not be sufficient to draw reliable conclusions, [BSL01] proposed that the average of K time steps should be taken into account. Therefore, the statistic we consider is the *time-average normalised innovation squared*, given by $\bar{\epsilon}^2_{y_k} = \frac{1}{K}\sum_{i=1}^{K} \epsilon^2_{y_k}$.

Under the assumption that H_0 has been retained, or in other words that there is no sufficient evidence to declare the Kalman filter as inconsistent, $K\bar{\epsilon}^2_{y_k}$ is χ^2 distributed with Kn degrees of freedom, where n is the dimension of the measurement vector [BSL01, Rei01]. Note that the χ^2 distribution, unlike the Gaussian distribution, is indexed by one integer parameter, the degrees of freedom. We fail to reject H_0 as long as $\bar{\epsilon}^2_{y_k} \in [r1, r2]$, where the confidence region is defined as $Pr\left\{\bar{\epsilon}^2_{y_k} \in [r1, r2] \mid H_0\right\} = 1 - \alpha$, and the significance *level* α is set to 0.05 as before. In general, a confidence region, or a confidence interval for the univariate case, of a level $1 - \alpha$ where $\alpha = 0.05$, defines a 95% interval in which we expect the unknown value of the parameter we try to estimate, in this case the NIS, to fall within this interval with probability at least 95%.

In other words, $\alpha = 0.05$ represents the probability of making a type I error (a false positive decision), by rejecting the null hypothesis when it is true. A type I error in our case means that we incorrectly declare the Kalman filter as inconsistent, and a level $\alpha = 0.05$ defines an upper bound on the probability of committing such an error. For example, assuming 40 time steps ($K = 40$) and a one-dimensional measurement vector, the *two-sided confidence region* for the χ^2_{40} variable is $[\chi^2_{40}(0.025), \chi^2_{40}(0.975)]$, which is defined as $[24.433, 59.342]$.

Table 2. User inputs for each of the models

Input	Description	Used in	Type
$\hat{x}_0^+$	A posteriori state estimate vector	Filter	`RealVector`
P_0^+	A posteriori estimation-error covariance matrix	Filter	`RealMatrix`
x	State vector	System	`RealVector`
w	Process noise vector	System	`RealVector`
v	Measurement noise vector	System	`RealVector`
F	State transition matrix	Shared	`RealMatrix`
Q	Process noise covariance matrix	Filter	`RealMatrix`
H	Measurement matrix	Shared	`RealMatrix`
R	Measurement noise covariance matrix	Shared	`RealMatrix`
`gLevel`	Granularity of the noise	Shared	`int`
`decPlaces`	Number of decimal places	Shared	`int`
`maxTime`	Maximum time the model will run	Shared	`int`
`filterType`	Type of filter variant	Shared	`int`

Then, since we want to test whether the time-average normalised innovation squared statistic falls within the interval we divide the lower ($r1$) and upper ($r2$) endpoints by the number of time steps, in this case $K = 40$, to obtain the following interval: $[0.6108, 1.4835]$. For instance, if $\bar{\epsilon}^2_{y_k} = 0.85$, hypothesis H_0 is not rejected (i.e., H_0 is retained), and if it falls outside of the above region (e.g., $\bar{\epsilon}^2_{y_k} = 2.67$), H_0 is rejected.

The aforementioned statistical test is constructed by specifying a formula of the form $\mathtt{R}^{nis\_avg}_{=?}[\,\mathtt{C}^{\leq maxTime}\,]$. First, a formula *nis* is used which calculates the normalised innovation squared statistic. Then, we define a reward function *nis_avg* which divides the *accumulated* result obtained from the previous formula, *nis*, by the maximum time the model runs, and assigns the result to the states of the model. In each transition the cumulative reward is updated and when it reaches the terminal state, the time-average normalised innovation squared statistic (*nis_avg*) is obtained. The terminal state of the model is defined by the predicate `t=maxTime`. Note that the time-average normalised squared value is obtained by dividing by `(maxTime - 2)`, rather than by `maxTime`. This is because we do not want to take into account in the computation the values of the initial states of the model (when $t = 0$ or when $t = 1$). The value of `(maxTime - 2)` essentially defines the degrees of freedom of the χ^2 distribution, which means that when `maxTime=20`, the lower and upper bounds are computed for 18 degrees of freedom.

4. Tool Support: VerFilter

Next, we provide some details about the tool, VerFilter, which is the software implementation of the framework defined in Section 3. The VerFilter tool is written in the Java programming language in order to be seamlessly integrated with the PRISM libraries, which are also mostly written in Java. The tool and supporting files for the results in the next section are available from [Sup].

VerFilter Inputs. In Table 2 we show the user inputs available to VerFilter, distinguishing which of those refer to the system and measurement model, which refer specifically to the filter models and which are shared between them. The `RealVector` and `RealMatrix` shown in Table 2 are implemented as one-dimensional and two-dimensional arrays of type `double`, respectively. VerFilter also takes as inputs four extra parameters: (i) `gLevel`, which takes an integer between 2 and 6, and has been discussed in Section 3.1; (ii) `decPlaces`, which allows the user to specify an integer between 2 and 15, the number of decimal places to which the numerical values used in the computations will be rounded; (iii) `maxTime`, which is an integer and determines the maximum time the model will run; and (iv) `filterType`, which is the type of filter to be executed.

VerFilter Algorithms. In this paper, we focus on four filter variants: the conventional Kalman filter (`CKFilter`), the Carlson-Schmidt square-root filter (`SRFilter`), the Bierman-Thornton U-D filter (`UDFilter`) and the steady-state Kalman filter (`SSFilter`). In VerFilter, several of the numerical linear algebra computations for implementing Kalman filters are done using the Apache Commons Math library [apa], while other parts have been manually implemented. In `CKFilter`, for example, the library is used for "basic" matrix operations and for the eigen and singular value decomposition of P. For `SRFilter`, algorithms im-

plemented manually include the upper-triangular Cholesky factorisation and Carlson's measurement update with Schmidt's time update using *Householder transformations*.

5. Experimental Results

We now illustrate results from the implementation of our techniques on the four filters mentioned above. For the system models in our experiments, we use two distinct *kinematic state models* which describe the motion of objects as a function of time. For the first, the *discrete white noise acceleration model* (DWNA), the initial estimation-error covariance matrix P_0^+ is defined as $10{\cdot}I$. Defining P_0^+ as a diagonal matrix is quite common, since it is initially unknown whether the state variables are correlated to each other. The process noise covariance matrix is given by $Q = \Gamma\sigma_w^2\Gamma^T$ where the noise gain matrix $\Gamma = [\frac{1}{2}\Delta t^2\ \Delta t]^T$ is initialised by setting the sampling interval Δt to 1, which results in $\Gamma = [0.5\ 1]^T$. The variance σ_w^2 is set to 0.001 initially. For the second model, the *continuous white noise acceleration model* (CWNA), σ_w^2 is initially set to 0.001. Note that each of these models results in a different process noise covariance matrix Q. For more details on these models, see [Sup]. Apart from the experiments described in Section 5.3, where we investigate efficiency and scalability, we fix the noise discretisation granularity at `gLevel`=2.

5.1. Verification of Numerical Stability

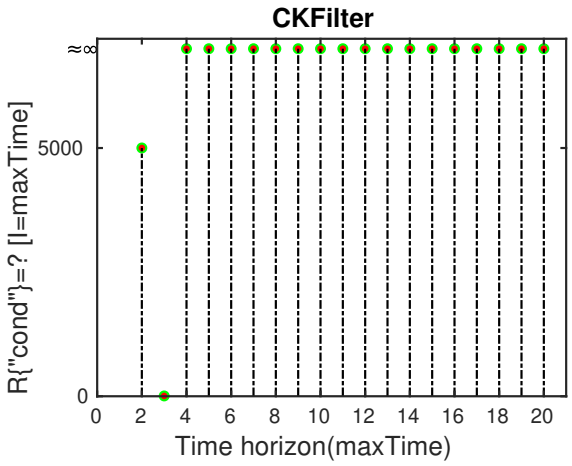
Condition numbers. In the first set of experiments, results for which are shown in Figure 3, we analyse the condition numbers of P^+ for the conventional (`CKFilter`) and U-D (`UDFilter`) filters, and C^+ for the square-root filter (`SRFilter`). This is in order to verify that P^+, or the matrices which constitute P^+, remain well-conditioned in terms of maintaining their nonsingularity as they are being propagated forward in time (as discussed in Section 3.2). Note that, for the U-D filter, we verify the reconstructed matrix P^+ at a specific time step, while for the square-root filter, we use the Cholesky factor C^+. We check this property while varying two parameters. The first is the numerical precision in terms of the number of decimal places, which we vary from 3 to 6 inclusive. The second is the time horizon of the model, which in our case is measured in discrete time steps and is varied from 2 to 20.

Our goal is twofold. Firstly, we examine whether an increase in numerical precision has a meaningful effect on how accurately the condition number is computed. This is important since, as we show in Section 5.3, a decrease in the numerical precision usually makes verification more efficient. Being able to consider an appropriate threshold above which an increase in the numerical precision will not have an effect on the property to be verified can determine the applicability of these verification mechanisms in realistic settings. Secondly, we examine whether letting the model evolve for a greater amount of time could have an impact on the property that is being verified.

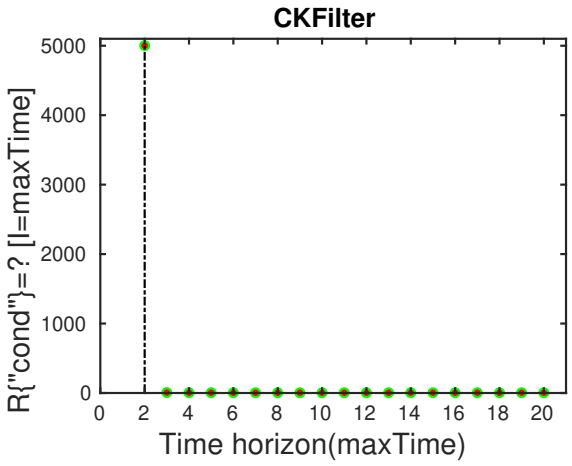
The first observation for the `CKFilter` as shown in Figures 3a and 3b is that the increased numerical precision actually affects the verification result. For example, we note that for `maxTime` values in the range of $4-20$, when the input to our model for the numerical precision is 3 decimal places, the instantaneous reward jumps to infinity. An infinite reward in this case means that the condition number of P^+ is $\approx$ 1.009e+16, which practically means that P^+ is "computationally" singular and consequently positive definiteness is not being preserved. This is in contrast to the result obtained for both the `SRFilter` and `UDFilter`, where positive definiteness is preserved, irrespective of the numerical precision and the value of `maxTime` used. For the `SRFilter`, as shown in Figures 3c and 3d, the highest reward value is $\approx$ 70, while for the `UDFilter`, Figures 3e and 3f, it is $\approx$ 5000. This means that the positive definiteness property of P^+ is maintained throughout the execution of these two types of filters, and the increase in the numerical precision did not have any significant effect on the propagated matrix P^+.

On the other hand, for the `CKFilter`, positive definiteness is preserved when we increase the numerical precision to a value > 4, and the instantaneous reward assigned to the states fluctuates around small values close to zero. Another interesting observation is that, for the three filters considered, the instantaneous rewards stabilise to a value of $\approx$ 2, irrespective of whether the numerical precision is 4, 5 or 6. In fact, for the three filter variants, the actual absolute difference of the rewards over the states in which positive definiteness is preserved between a numerical precision of 5 and 6 decimal places, is $\approx$ 0.1.

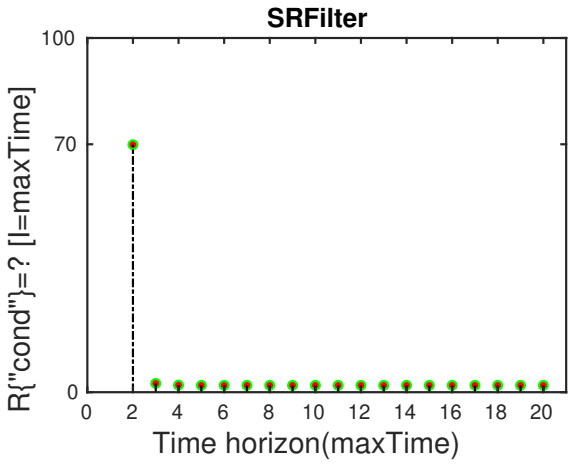
Filter tuning. In the second set of experiments, we consider `CKFilter` and `SRFilter`, while the system



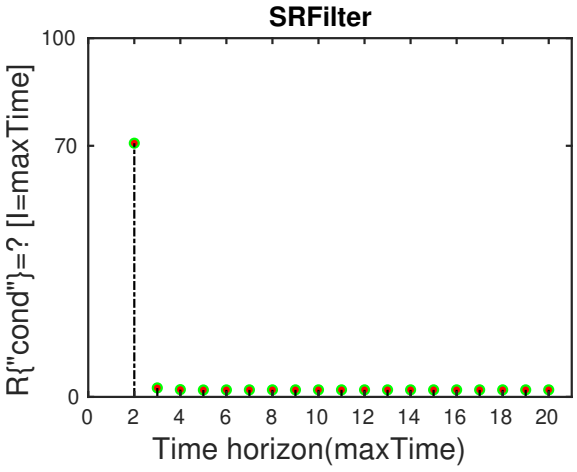
(a) 3 decimal places



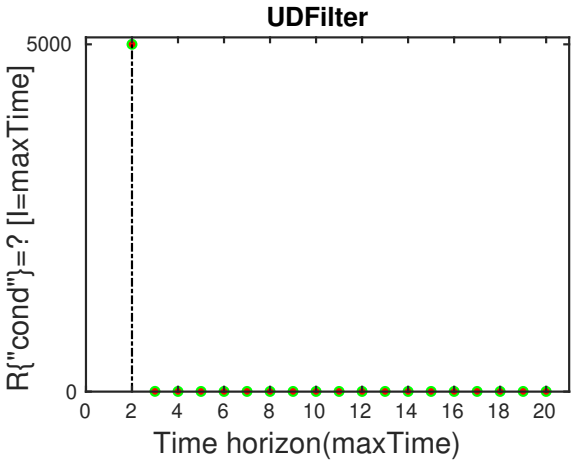
(b) 4 - 6 decimal places



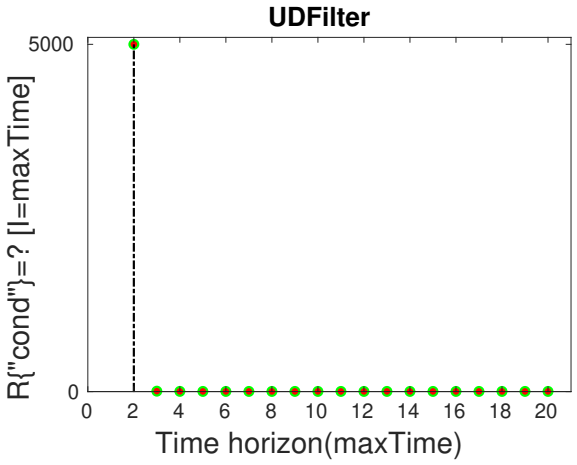
(c) 3 decimal places



(d) 4 - 6 decimal places



(e) 3 decimal places



(f) 4 - 6 decimal places

Fig. 3. Condition number of $P+$ over time under various degrees of precision.

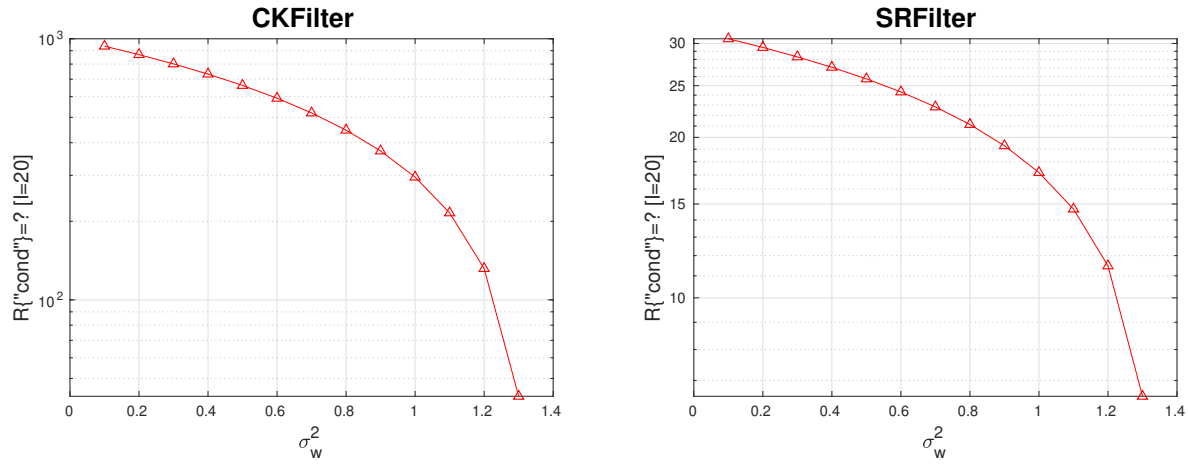


Fig. 4. Verifying the goodness of P^+

model is a continuous white noise acceleration kinematic model. Our goal is to examine how VerFilter can be used to examine heuristic-based approaches and ad-hoc methods such as artificial noise injection in terms of their usefulness in correcting potential numerical problems in P^+. This is also helpful in situations where it is challenging to determine the elements of Q, by performing an automatic search over those values which will produce an optimal performance, in this case in terms of the numerical robustness of P.

To this end, we verify whether P^+ will remain well-conditioned or not, by varying the elements of Q. Similar to the first set of experiments we verify C^+ directly. For both of the filters, the noise variance σ_w^2, which determines the elements of Q, is the input to our model, P^+ (and C^+) is being verified against. We do not vary the maximum time; rather, we let the Markov chain evolve to a fixed `maxTime`= 20 time steps, which corresponds to $\approx 1 \times 10^6$ states. Also, the numerical precision is fixed to 3 decimal places.

Figure 4 shows the effects of increasing the variance of the noise by small increments, which is then multiplied with the elements of Q, for the `CKFilter` and the `SRFilter`, respectively. The first point of the plot $(0.1, 1000)$ in Figure 4a, means that for a value of $\sigma_w^2 = 0.1$, the corresponding instantaneous reward which corresponds to the condition number of P^+ in a set of states where `maxTime`=20, is 1000. Similarly, the first point of the plot $(0.1, 70)$ in Figure 4b, implies that the corresponding instantaneous reward for C^+ for a `maxTime`=20 is ≈ 70. For both of the filter types, a similar trend is observed. As we increase σ_w^2, the reward values decrease which implies an increase on the "quality" of P^+s. In the end, the reward values reach a condition number of ≈ 43 and ≈ 1.94 for the two filters, respectively.

For this specific case, the optimal value is $\sigma_w^2 = 1.3$ for both filters. For `CKFilter` in particular, it is important to check, when performing verification on Markov chains whose trajectories evolve over multiple states, that the positive definiteness of P^+ is not destroyed between successive states (i.e., successive time steps). To this end, it is advisable to use a property of the form $\mathtt{P}_{=?}[\,\mathtt{G}\; isPD\,]$ and reject models in which this is not satisfied with probability one.

Filter comparison. In Table 3, we compare three of the Kalman filter variants available in VerFilter: `CKFilter`, `SRFilter` and `UDFilter`. In this set of experiments, the setup is similar to the first one in terms of the system model and the filter parameters used. First, our purpose is to demonstrate the correctness of our approach by comparing the condition numbers of P^+ and C^+ between the three filter types. For instance, when the positive definiteness property is not violated, and as the reward values begin to converge to certain values, then the difference of the condition number of P^+ between `CKFilter` and `UDFilter` should be small. The same can also be said for the difference between the condition number of P^+ of `CKFilter` (and `UDFilter`), and the squared value of the condition number of C^+ of `SRFilter`. For example, one can quickly observe in Table 3 that the reward values of `CKFilter` are nearly the same as the respective values of `UDFilter`. In particular, as the time horizon increases, the values of these two filters become equal to each other. Similarly, one can observe that the squared reward values of `SRFilter` are approximately the same as the reward values obtained for the other two filters. For instance, for a numerical precision of 6 decimal places and a `maxTime` value of 10, the reward value for `SRFilter` is $\mathtt{R}^{cond}_{I=10} = 1.966205$. Squaring this gives $1.966205^2 \approx 3.866$, which is a very close approximation to the reward values of the other two filters.

Table 3. Performance comparison between three filter variants.

decPlaces	CKFilter $\mathtt{P}_{=?}[\mathtt{G}\, isPD]$	SRFilter $\mathtt{P}_{=?}[\mathtt{G}\, isPD]$	UDFilter $\mathtt{P}_{=?}[\mathtt{G}\, isPD]$	CKFilter $\mathtt{R}^{cond}_{=?}[\mathtt{I}^{=maxTime}]$	SRFilter $\mathtt{R}^{cond}_{=?}[\mathtt{I}^{=maxTime}]$	UDFilter $\mathtt{R}^{cond}_{=?}[\mathtt{I}^{=maxTime}]$
3,maxTIme=2	1	1	1	5001	69.875	5000
3,maxTIme=3	1	1	1	6.854102	2.479506	3.472570
3,maxTIme=4	0	1	1	$+\infty$	2.016643	2.370931
3,maxTIme=5	0	1	1	$+\infty$	1.943289	2.200921
3,maxTIme=6	0	1	1	$+\infty$	1.943289	2.192302
3,maxTIme=7	0	1	1	$+\infty$	1.943289	2.192302
3,maxTIme=8	0	1	1	$+\infty$	1.943289	2.192302
3,maxTIme=9	0	1	1	$+\infty$	1.943289	2.192302
3,maxTIme=10	0	1	1	$+\infty$	1.943289	2.192302
3,maxTIme=11	0	1	1	$+\infty$	1.943289	2.192302
3,maxTIme=12	0	1	1	$+\infty$	1.943289	2.192302
3,maxTIme=13	0	1	1	$+\infty$	1.943289	2.192302
3,maxTIme=14	0	1	1	$+\infty$	1.943289	2.192302
3,maxTIme=15	0	1	1	$+\infty$	1.943289	2.192302
3,maxTIme=16	0	1	1	$+\infty$	1.943289	2.192302
3,maxTIme=17	0	1	1	$+\infty$	1.943289	2.192302
3,maxTIme=18	0	1	1	$+\infty$	1.943289	2.192302
3,maxTIme=19	0	1	1	$+\infty$	1.943289	2.192302
3,maxTIme=20	0	1	1	$+\infty$	1.943289	2.192302
4,maxTIme=2	1	1	1	5001	70.768988	5000.800100
4,maxTIme=3	1	1	1	6.376508	2.505185	6.784395
4,maxTIme=4	1	1	1	3.439411	2.040158	4.519086
4,maxTIme=5	1	1	1	3.614225	1.939357	3.907120
4,maxTIme=6	1	1	1	3.614225	1.951946	3.866359
4,maxTIme=7	1	1	1	3.614225	1.964276	3.866359
4,maxTIme=8	1	1	1	3.614225	1.962769	3.866359
4,maxTIme=9	1	1	1	3.614225	1.962769	3.866359
4,maxTIme=10	1	1	1	3.614225	1.962769	3.866359
4,maxTIme=11	1	1	1	3.614225	1.962769	3.866359
4,maxTIme=12	1	1	1	3.614225	1.962769	3.866359
4,maxTIme=13	1	1	1	3.614225	1.962769	3.866359
4,maxTIme=14	1	1	1	3.614225	1.962769	3.866359
4,maxTIme=15	1	1	1	3.614225	1.962769	3.866359
4,maxTIme=16	1	1	1	3.614225	1.962769	3.866359
4,maxTIme=17	1	1	1	3.614225	1.962769	3.866359
4,maxTIme=18	1	1	1	3.614225	1.962769	3.866359
4,maxTIme=19	1	1	1	3.614225	1.962769	3.866359
4,maxTIme=20	1	1	1	3.614225	1.962769	3.866359
5,maxTIme=2	1	1	1	5001.060113	70.722645	5000.810100
5,maxTIme=3	1	1	1	6.307259	2.507737	6.322909
5,maxTIme=4	1	1	1	4.164446	2.031345	4.116405
5,maxTIme=5	1	1	1	3.758763	1.932001	3.801081
5,maxTIme=6	1	1	1	3.866359	1.954296	3.852360
5,maxTIme=7	1	1	1	3.866359	1.966188	3.866359
5,maxTIme=8	1	1	1	3.866359	1.966978	3.866359
5,maxTIme=9	1	1	1	3.866359	1.966224	3.866359
5,maxTIme=10	1	1	1	3.866359	1.966224	3.866359
5,maxTIme=11	1	1	1	3.866359	1.966224	3.866359
5,maxTIme=12	1	1	1	3.866359	1.962769	3.866359
5,maxTIme=13	1	1	1	3.866359	1.962769	3.866359
5,maxTIme=14	1	1	1	3.866359	1.962769	3.866359
5,maxTIme=15	1	1	1	3.866359	1.962769	3.866359
5,maxTIme=16	1	1	1	3.866359	1.962769	3.866359
5,maxTIme=17	1	1	1	3.866359	1.962769	3.866359
5,maxTIme=18	1	1	1	3.866359	1.962769	3.866359
5,maxTIme=19	1	1	1	3.866359	1.962769	3.866359
5,maxTIme=20	1	1	1	3.866359	1.962769	3.866359
6,maxTIme=2	1	1	1	5001.062113	70.720408	5000.812100
6,maxTIme=3	1	1	1	6.292646	2.507687	6.283849
6,maxTIme=4	1	1	1	4.129379	2.031178	4.125259
6,maxTIme=5	1	1	1	3.735914	1.931918	3.732897
6,maxTIme=6	1	1	1	3.831732	1.954872	3.820638
6,maxTIme=7	1	1	1	3.874810	1.966389	3.868675
6,maxTIme=8	1	1	1	3.866359	1.966605	3.867762
6,maxTIme=9	1	1	1	3.866359	1.966262	3.866359
6,maxTIme=10	1	1	1	3.866359	1.966205	3.866359
6,maxTIme=11	1	1	1	3.866359	1.966205	3.866359
6,maxTIme=12	1	1	1	3.866359	1.966205	3.866359
6,maxTIme=13	1	1	1	3.866359	1.966205	3.866359
6,maxTIme=14	1	1	1	3.866359	1.966205	3.866359
6,maxTIme=15	1	1	1	3.866359	1.966205	3.866359
6,maxTIme=16	1	1	1	3.866359	1.966205	3.866359
6,maxTIme=17	1	1	1	3.866359	1.966205	3.866359
6,maxTIme=18	1	1	1	3.866359	1.966205	3.866359
6,maxTIme=19	1	1	1	3.866359	1.966205	3.866359
6,maxTIme=20	1	1	1	3.866359	1.966205	3.866359

The superiority of `SRFilter` and of `UDFilter` compared to `CKFilter` is demonstrated when the numerical precision is 3 decimal places and mainly from the fact that, for the same set of parameters, the numerical robustness of P^+ is preserved. This can be seen by comparing the computed results of the reward-based properties as shown in the fifth, sixth and seventh columns of Table 3. We note that, for a numerical precision of 3 decimal places, when choosing `CKFilter`, the reward value shoots up to $+\infty$, representing a covariance matrix in which the positive definiteness property is destroyed, while in the `SRFilter` and `UDFilter` cases, the corresponding reward values settle around the small values of 1.94 and 2.20, respectively. This is also evident by observing the second, third and fourth columns of Table 3, which tell us whether the *isPD* invariant will be maintained in all states of the model. Notably, the positive definiteness property in `CKFilter` does not hold for every state, in fact the probability is zero, while for `SRFilter` the positive definiteness property holds for every state with probability one.

Moreover, when the numerical precision is increased in the range of $4-6$ decimal places, the performance of the three filters in terms of the numerical stability of P^+ is roughly the same, and the positive definiteness property of P^+ is maintained in every state considered. One can also note that the reward values of `CKFilter` and of `SRFilter` converge to the same value for a numerical precision ≥ 5 decimal places.

5.2. Verification of Consistency Properties

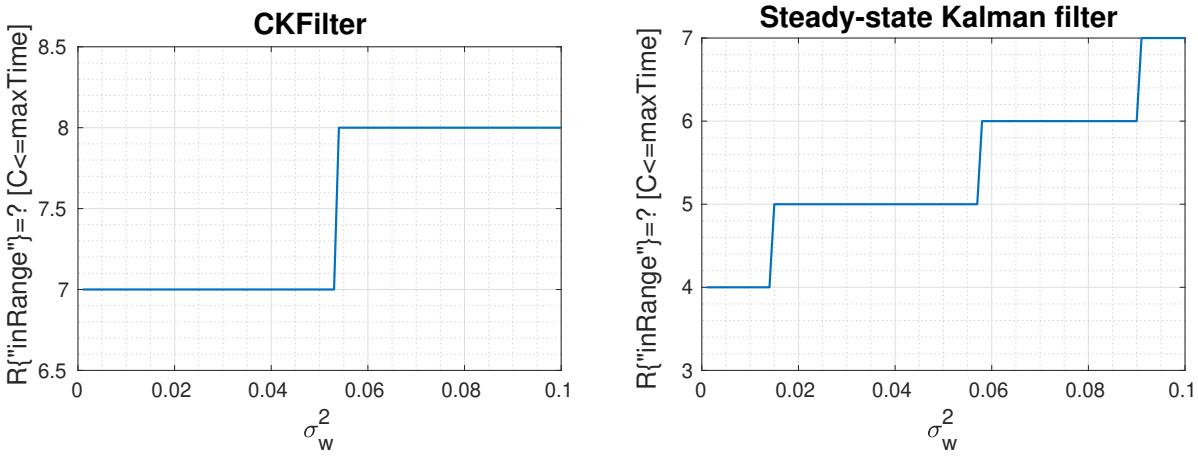
In this section, we focus on verifying the so-called consistency of the conventional (`CKFilter`) and of the steady-state Kalman filters. The consistency criteria of these two different Kalman filter variants are verified under a wide range of different process noise covariance matrices. These criteria are specified in the form of properties, and have been described in Section 3.3. The system model is defined as a continuous white noise acceleration kinematic model, and its noise covariance matrix is initially defined as $Q = \begin{bmatrix} \frac{1}{3}\Delta t^3 & \frac{1}{2}\Delta t^2 \\ \frac{1}{2}\Delta t^2 & \Delta t \end{bmatrix} \sigma_w^2$. The sampling interval, Δt, is set to 1, and the measurement noise variance R to 10.

Alternatively, our goal can be described as verifying the process noise covariance matrix tuning, as an effective filter tuning technique for finding "optimal" noise covariance matrices in terms of satisfying the respective consistency criteria. In particular, we analyse by how much the elements of the process noise covariance matrix should be scaled. Before each one of the Kalman filter variants is passed as an input to VerFilter, to handle the creation of the probabilistic model, their process noise covariance matrices are multiplied by the variance of noise that perturbs the system model (i.e., σ_w^2). Then, we vary σ_w^2 over a range of values verify certain consistency properties.

In our experiments, we vary σ_w^2 between 0.001 and 5.5 inclusive. From 0.001 to 0.1 and from 0.1 to 5.5, the σ_w^2 value is increased in increments of 0.001 and 0.1, respectively. This results in 135 different process noise covariance matrices that the conventional and the steady-state Kalman filters are verified against. Furthermore, since the impact of the numerical precision, in terms of the number of decimal places, on the verification result is also of interest, the `decPlaces` value is also varied between 3 and 6 inclusive. It is important to state again the two important preconditions of controllability and observability, which have to be satisfied before the verification of the steady-state Kalman filter can begin.

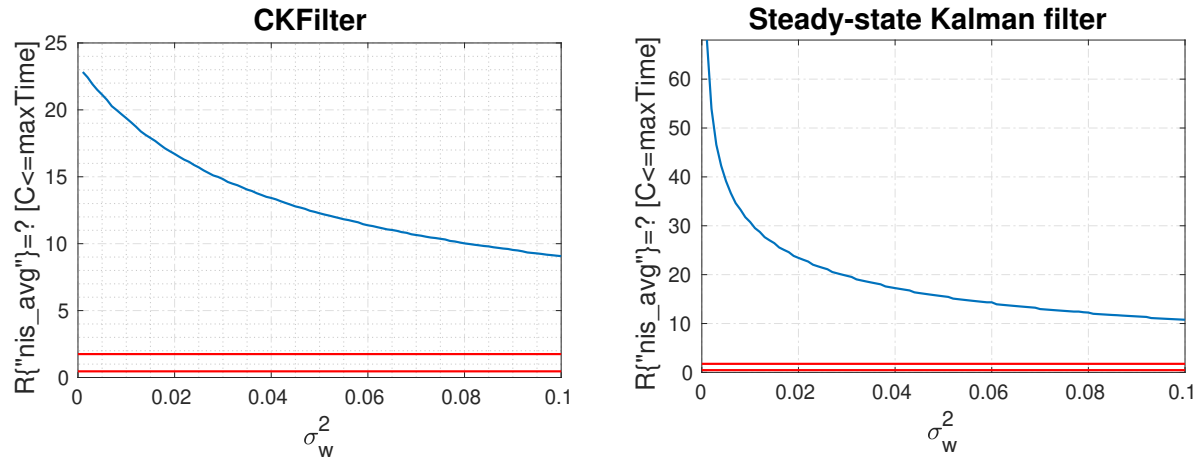
We first present our verification results for the conventional and steady-state Kalman filters. In the first set of experiments, we verify the consistency of these two filter types, for noise variance (σ_w^2) values in the range $[0.001, 0.1]$. The first property we verify is whether the magnitude of innovation is bounded by its variance 95% of the time. Figure 5 shows the expected number of states, for the two filter variants, in which the value of the innovation falls between two standard deviations of the mean, for different σ_w^2 values. This illustrates the impact of tuning the process noise covariance matrix on the "quality" of the filters considered. The parameters `decPlaces` and `maxTime` are set to 3 and 20, respectively. As explained in Section 3.3, for `maxTime`$= 20$, the value of the 95% lower bound we are seeking is 17.

Looking at the results, we note that, for the minimum noise variance value considered, $\sigma_w^2 = 0.001$, the expected value of the *inRange* reward is 7 (i.e., $\approx 39\%$) and 4 (i.e., $\approx 22\%$) for the conventional and the steady-state Kalman filters, respectively. However, these values fall far from the 95% bound we are seeking to satisfy. Moreover, as noise, in increments of 0.001, is injected in the two filters, the *inRange* reward values steadily increase, and reach a value of 8 and 7 for the conventional and steady-state Kalman filter, respectively. This happens when the process noise covariance matrices are constructed by setting a maximum value of σ_w^2, for this set of experiments, that is 0.1. The *inRange* reward values computed for the two filter variants tell us that the number of times the value of the innovations falls within $\pm 2\sqrt{S_k}$, between a σ_w^2



(a) (b)

Fig. 5. Expected number of states in which the innovation falls between two standard deviations of the mean.



(a) (b)

Fig. 6. Time-average normalised innovation squared statistic.

of 0.001 and 0.1, has increased by 5.4% and by 17% for the two filters, approximately. The increase on the computed reward values of 17% for the steady-state Kalman filter indicates that the effect of artificial noise injection is more prominent, compared to the equivalent increase of 5.4% for `CKFilter`. However, the $\approx 44.4\%$ and $\approx 39\%$ values for `CKFilter` and the steady-state filter, when $\sigma_w^2 = 0.1$, are still significantly less than the 95% bound which is required for the two types of filters to be considered consistent. As a result the process noise covariance matrices, and consequently the filters constructed in this range, should be rejected.

The second property we verify, shown in Figure 6, is whether the magnitude of the innovations is proportionate to the computed covariances of the conventional and of the steady-state Kalman filters. We plot the different values of the time-average normalised innovation squared statistic, which has been computed for different σ_w^2 values in the range of 0.001 to 0.1 inclusive. The goal of this experiment is to verify the consistency of the conventional and steady-state Kalman filters under different process noise covariance matrices, and pick a process noise covariance matrix which drives the aforementioned statistic to fall between the appropriate acceptance regions. Those regions are visualised as the lower and upper red lines on the plots, and essentially define the lower (0.46) and upper (1.75) bounds of the χ^2 distribution, respectively.

For `CKFilter` (Figure 6a), as the value of the noise variance σ_w^2 increases, the *nis_avg* values decrease, implying that the consistency of the filter is increasing as more noise is being injected to the filter. This has

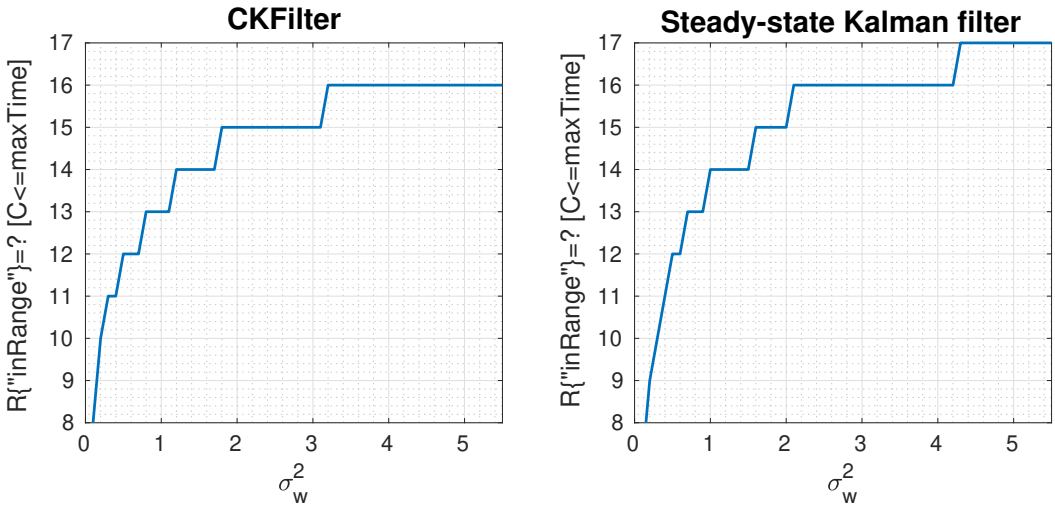


Fig. 7. Expected number of states in which the innovation falls between two standard deviations of the mean.

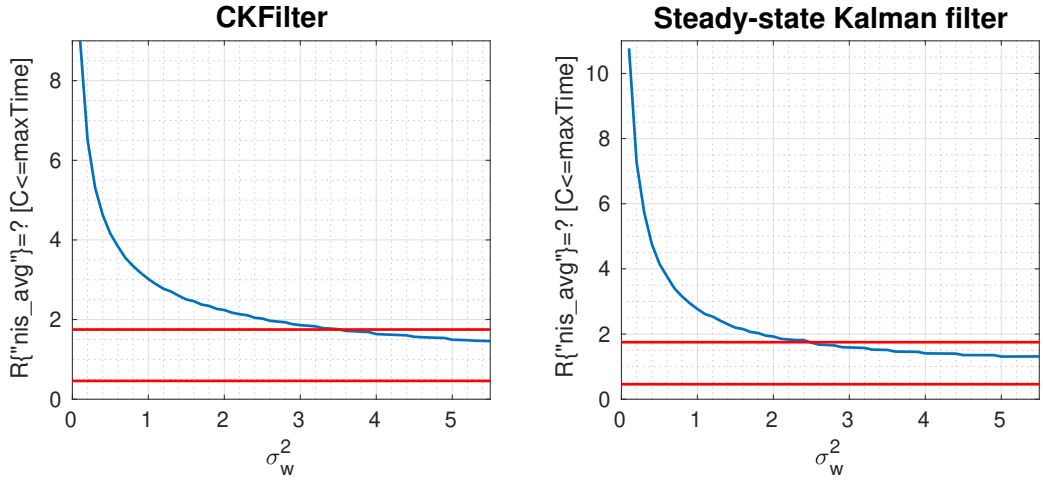
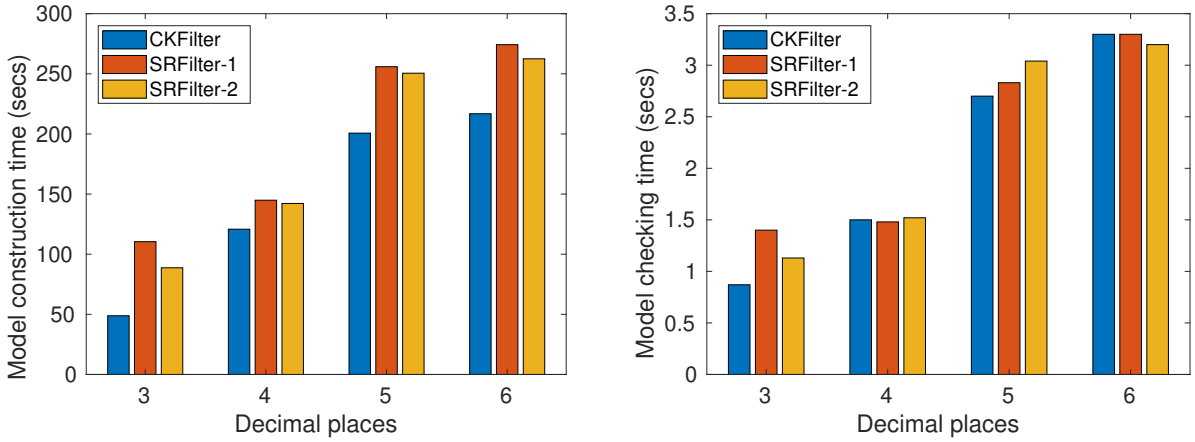


Fig. 8. Time-average normalised innovation squared statistic.

been also observed in the first set of experiments. For example, for $\sigma_w^2 = 0.001$, the respective time-average normalised innovation squared (*nis_avg*) reward value is 23, approximately. On the other extreme of the plot, when $\sigma_w^2 = 0.1$ the *nis_avg* reward value becomes ≈ 9, denoting an overall decrease, in the range of the σ_w^2 values considered, by a factor of 2.5.

A similar increasing trend, in terms of the improved consistency, is also observed for the steady-state Kalman filter (Figure 6b). The *nis_avg* reward value is ≈ 69 for the minimum value of noise variance considered ($\sigma_w^2 = 0.001$). The reward value of 69 is considerably larger than the reward value of 23, computed for `CKFilter` for the same input. The larger reward of the steady-state Kalman filter can be explained by the fact that a suboptimal Kalman gain is being used in the computations at every time step. This is in contrast to `CKFilter` in which the optimal Kalman gain is being used. For example, for this particular case, the steady-state Kalman gain propagated through the Markov chain states is $[0.141, 0.009]^T$. On the other hand, the Kalman gain of the `CKFilter`, computed from the state ($s = 2$, $t = 2$), is $[0.666, 0.333]^T$ and converges to the values of $[0.183, 0.014]^T$ at state ($s = 20$, $t = 20$). In fact, the initial difference, when $\sigma_w^2 = 0.001$, between the *nis_avg* rewards of the two filters, is 46, but falls to a value of less than 3 for $0.44 \leq \sigma_w^2 \leq 0.1$. The gradual decrease in the difference between the filters' rewards indicates that, as more noise is injected, their performance becomes similar.

In the second set of experiments, shown in Figures 7 and 8, the setup is similar to the first set of



(a) Model construction time

(b) Model checking time

Fig. 9. Time comparisons between three filters.

experiments, except that the σ_w^2 values, in the interval $[0.1, 5.5]$, are injected in larger increments of 0.1, instead of 0.001. In Figure 7, we show the results for the *inRange* reward values, which were computed for the specified interval, for the two filters. For `CKFilter` in Figure 7, the *inRange* reward values reach a plateau at 16, for a value of σ_w^2 between 3.2 and 5.5 inclusive. Moreover, the *nis_avg* reward values, start to reside inside the confidence region for σ_w^2 values ≥ 3.5, for the specified interval. This means that on average, in 16 out of the 18 states of a particular path, the value of the innovation is expected to fall between two standard deviations of the mean. This gives us a value of $\approx 89\%$, which is less than the 95% bound we seek to satisfy. In consequence, the verification procedure of this particular consistency criterion was not successful for the 55 different conventional Kalman filters which were verified for this particular range. On the other hand, 13 out of the 55 different steady-state Kalman filters verified, satisfied this particular consistency property when their process noise covariance matrices Q were constructed with a σ_w^2 value between 4.3 and 5.5 inclusive.

In Figure 8, we show the results for the *inRange* reward values, which were computed for the specified interval, for the two filters. For example, the *nis_avg* value is ≈ 1.75 when verifying a `CKFilter` whose process noise covariance matrix, Q, has been constructed using a σ_w^2 value of 3.5. For the subsequent σ_w^2 values $(3.6, 3.7, ...)$, the *nis_avg* values are constantly decreasing. For instance, for the maximum σ_w^2 value we consider (5.5), the computed *nis_avg* reward is 1.46. This means that the time-average normalised innovation squared quantity starts to stabilise itself when it passes the upper bound of the confidence region ($\sigma_w^2 \geq 3.6$). This results in the non-rejection of the null hypothesis, which states that the filter is consistent. In fact, none of the 21 different filters constructed with σ_w^2 values from 3.5 to 5.5 inclusive should be declared as consistent. On the other hand, while the *nis_avg* reward computed for the steady-state Kalman filter is initially much larger than the conventional one, it reaches the acceptance region faster. In fact, its *nis_avg* reward is 1.74 when it is constructed with a $\sigma_w^2 = 2.5$, which is ten times less than its conventional equivalent. It also indicates that, for this particular case, the performance of the steady-state Kalman filter surpasses the conventional filter's one, since less noise has to be injected to satisfy the consistency criterion.

5.3. Scalability and Accuracy

Finally, we report on the scalability of our approach in terms of the model construction and model checking time, using a model of fixed `maxTime=20` time steps corresponding to $\approx 1 \times 10^6$ states, across the three filter variants. The rationale behind this section is to emphasise the careful analysis that needs to be performed to systematically evaluate the trade-offs between the accuracy of the verification result and the speed of the verification algorithms. In Figure 9, we show the time comparisons, for varying degrees of precision, between a model which encodes the conventional Kalman filter (`CKFilter`), and two implementations of the Carlson-Schmidt square-root filter with (`SRFilter-1`) and without (`SRFilter-2`) reconstruction of the

Table 4. Assessing accuracy of results by varying gLevel

gLevel	decPlaces	$\mathtt{R}^{nis\_avg}_{=?}[\mathtt{C}^{\leq 8}]$	$\mathtt{R}^{inRange}_{=?}[\mathtt{C}^{\leq 8}]$	Construction time (secs)
2	6	3.273	3.750	0.1
4	6	3.547	3.848	12
6	6	3.917	3.866	375

estimation-error covariance matrix, respectively. The model checking time refers to the collective time it takes to compute the first and second property of Section 5.1. These sets of experiments were run on a 16GB RAM machine with an i7 processor at 1.80GHz, running Ubuntu 18.04.

It is apparent that the increased numerical precision affects the construction time of the models. The average model construction time of the three filter variants increased by a factor of ≈ 3 from 3 to 6 decimal places. Specifically, the average time is ≈ 83 seconds for 3 decimal places compared to ≈ 249 seconds, when 6 decimal places were used. Moreover, the construction of `CKFilter` was the fastest in all the degrees of precision considered, however, as it was noted in Section 5.1 it produces an inaccurate verification result when the number of decimal places is 3.

Conversely, the construction times of the two square-root filters were about the same, and it seems that the extra computational step ($P = CC^T$) did not have a significant effect on the performance of the model construction. However, it should be borne in mind that these experiments were conducted on systems represented by two-dimensional matrices. The model checking times are shown in Figure 9b and one can observe that they follow a similar pattern with the model construction times shown earlier, in terms of the increase in time from 3 to 6 decimal places. For instance, the average model checking time increases by a factor of ≈ 3 when 6 decimal places are used, compared to 3.

Another observation is that the model checking time appears to be independent of the type of the filter used. This can be seen from the limited variability the model checking time experiences between the three filter variants, since for the degrees of precision considered, it remains at approximately the same level. This is in contrast to the model construction time which appears to be affected by the filter type, since it is considerably less for the `CKFilter` compared to its square root variants. The reason for this seems to be the extra computational steps (e.g., Householder transformations, generation of upper triangular Cholesky factors), which have to be performed in the time update (Schmidt part) and measurement update (Carlson part) of the Carlson-Schmidt square-root filter. In fact, for a precision of 6 decimal places, and once `CKFilter` is chosen as an input, we experience a drop in the model construction time of about 53 seconds. However, for the same amount of precision, the time it takes to model check all the three filters is around 3 seconds.

Closely related to scalability is the issue of accuracy. In Table 4, we show the values of two results obtained from `CKFilter`, by varying the granularity level of the noise, `gLevel`, and by setting the number of decimal places `decPlaces` to 6. We choose a `maxTime` value of 8, since this allows a comparison of different values in reasonable time. We assume that the highest value of `gLevel` considered (i.e., 6) provides the most accurate approximation of the noise, so the differences between these values and those for `gLevel`$= 2$ and `gLevel`$= 4$ give an indication of the improvement in accuracy as `gLevel` increases. Conversely, as expected, higher values of `gLevel` introduce an additional computational burden to the verification process. For example, for the *nis_avg* results, `gLevel` values of 2, 4 and 6 require model construction time of 0.1, 12 and 375 seconds, respectively.

6. Conclusion

We have presented a framework for the modelling and verification of Kalman filter implementations. It is general enough to analyse a variety of different implementations, and various system models, and to study a range of numerical and modelling error issues which may hinder the effective deployment of the filters in practice. We have implemented the techniques in a tool and illustrated its applicability and scalability with a range of experiments.

In general, the evaluation of Kalman filter performance has attracted considerable attention since the early days of their development. However, formal methods such as probabilistic model checking have not been used for their verification. This is, to the best of our knowledge, the first work where these types of

problems are applied to a probabilistic verification setting. Our main contribution in this work is that we show that probabilistic verification can be a promising alternative in verifying these types of systems.

Future work includes: a more in-depth analysis of the trade-off between accuracy and scalability; extending our techniques to work with non-linear models, potentially by incorporating a non-linear version of the Kalman filter such as the extended Kalman filter; and expanding the verification approach to other systems requiring non-trivial linear algebra implementations.

Acknowledgements. This work has been partially supported by an EPSRC-funded Ph.D. studentship (award ref: 1576386), and the PRINCESS project (contract FA8750-16-C-0045) funded by the DARPA BRASS programme.

References

[ACCK93] G. Abdelnour, S. Chand, S. Chiu, and T. Kido. On-line detection and correction of kalman filter divergence by fuzzy logic. In *1993 American Control Conference*, pages 1835–1839, June 1993.

[AM12] B.D.O. Anderson and J.B. Moore. *Optimal Filtering*. Dover Books on Electrical Engineering. Dover Publications, 2012.

[apa] Math - Commons-Math: The Apache Commons Mathematics Library.

[Bat64] Richard H. Battin. *Astronautical guidance*. Electronic sciences. McGraw-Hill, 1964.

[BH12] Robert Grover Brown and Patrick Y. C. Hwang. *Introduction to Random Signals and Applied Kalman Filtering with MATLAB Exercises, 4th Edition*. Wiley Global Education, January 2012.

[Bie75] G. J. Bierman. Measurement updating using the u-d factorization. In *1975 IEEE Conference on Decision and Control including the 14th Symposium on Adaptive Processes*, pages 337–346, Dec 1975.

[Bie77] G. J. Bierman. *Factorization Methods for Discrete Sequential Estimation*. 1977.

[BJ68] Richard S. Bucy and Peter D. Joseph. *processes with applications to guidance*. Interscience Publishers New York, 1968.

[BN76] F. M. Boland and H. Nicholson. Control of divergence in kalman filters. *Electronics Letters*, 12(15):367–369, July 1976.

[BS87] Y. Bar-Shalom. *Tracking and Data Association*. Academic Press Professional, Inc., San Diego, CA, USA, 1987.

[BSL01] Yaakov Bar-Shalom and Xiao-Rong Li. *Estimation with Applications to Tracking and Navigation*. John Wiley & Sons, Inc., New York, NY, USA, 2001.

[BT08] D.P. Bertsekas and J.N. Tsitsiklis. *Introduction to Probability*. Athena Scientific optimization and computation series. Athena Scientific, 2008.

[Car73] Neil A. Carlson. Fast triangular formulation of the square root filter. *AIAA Journal*, 11(9):1259–1265, 1973.

[CD16] C-B. Chang and K-P Dunn. *Applied State Estimation and Association*. The MIT Press, Cambridge, Massachusetts, 2016.

[CG74] The Analytic Sciences Corporation and Arthur Gelb. *Applied Optimal Estimation*. The MIT Press, 1974.

[DZ18] C. D'Souza and R. Zanetti. Information formulation of the udu kalman filter. *IEEE Transactions on Aerospace and Electronic Systems*, pages 1–1, 2018.

[EP19] Alexandros Evangelidis and David Parker. Quantitative verification of numerical stability for Kalman filters. In *Proc. 23rd International Symposium on Formal Methods (FM'19)*, volume 11800 of *LNCS*, pages 425–441. Springer, 2019.

[FKNP11] V. Forejt, M. Kwiatkowska, G. Norman, and D. Parker. Automated verification techniques for probabilistic systems. In M. Bernardo and V. Issarny, editors, *Formal Methods for Eternal Networked Software Systems (SFM'11)*, volume 6659 of *LNCS*, pages 53–113. Springer, 2011.

[GA14] Mohinder S. Grewal and Angus P. Andrews. *Kalman Filtering: Theory and Practice Using MATLAB*. Wiley-IEEE Press, 4th edition, 2014.

[Gib11] Bruce P Gibbs. *Advanced Kalman Filtering, Least Squares and Modeling: A Practical Handbook*. John Wiley & Sons, Inc., 2011.

[GSDW16] Q. Ge, T. Shao, Z. Duan, and C. Wen. Performance analysis of the kalman filter with mismatched noise covariances. *IEEE Transactions on Automatic Control*, 61(12):4014–4019, Dec 2016.

[HJ94] H. Hansson and B. Jonsson. A logic for reasoning about time and reliability. *Formal Aspects of Computing*, 6(5):512–535, 1994.

[Jaz70] A.H. Jazwinski. *Stochastic Processes and Filtering Theory*. Mathematics in Science and Engineering. Elsevier Science, 1970.

[JKB94] Norman Lloyd Johnson, Samuel Kotz, and N. Balakrishnan. *Continuous univariate distributions*. New York: Wiley, 1994.

[Kai80] Thomas Kailath. *Linear Systems*. Prentice-Hall, Englewood Cliffs, N.J, 1980.

[Kal60] R. E. Kalman. A new approach to linear filtering and prediction problems. *ASME Journal of Basic Engineering*, 1960.

[KBS71] P. Kaminski, A. Bryson, and S. Schmidt. Discrete square root filtering: A survey of current techniques. *IEEE Transactions on Automatic Control*, 16(6):727–736, December 1971.

[KNP07] Marta Kwiatkowska, Gethin Norman, and David Parker. Stochastic Model Checking. In Marco Bernardo and

Jane Hillston, editors, *Formal Methods for Performance Evaluation*, number 4486 in Lecture Notes in Computer Science, pages 220–270. Springer Berlin Heidelberg, May 2007.

[KNP11] M. Kwiatkowska, G. Norman, and D. Parker. PRISM 4.0: Verification of probabilistic real-time systems. In G. Gopalakrishnan and S. Qadeer, editors, *Proc. 23rd International Conference on Computer Aided Verification (CAV'11)*, volume 6806 of *LNCS*, pages 585–591. Springer, 2011.

[Kwi07] M. Kwiatkowska. Quantitative verification: Models, techniques and tools. In *Proc. 6th joint meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE)*, 2007.

[LJ03] X. Rong Li and V. P. Jilkov. Survey of maneuvering target tracking. part i. dynamic models. *IEEE Transactions on Aerospace and Electronic Systems*, 39(4):1333–1364, Oct 2003.

[May82a] Peter S Maybeck. *Stochastic models, estimation, and control: Volume 1*. Mathematics in science and engineering. Elsevier Science, Burlington, MA, 1982.

[May82b] Peter S. Maybeck. *Stochastic Models: Estimation and Control: Volume 2*. Mathematics in Science and Engineering. Elsevier Science, 1982.

[MGB03] Mark Moulin, Leonid Gluhovsky, and Eli Bendersky. Formal verification of maneuvering target tracking. *AIAA Guidance, Navigation, and Control Conference and Exhibit*, 2003.

[Rei01] Ian Reid. Estimation ii, 2001.

[RG03] J. Van Baalen R. Gamboa, J. Cowles. On the verification of synthesized kalman filters. In *4th International Workshop on the ACL2 Theorem Prover and Its Applications.*, 2003.

[RVWL03] Grigore Roşu, Ram Prasad Venkatesan, Jon Whittle, and LaurenŢiu Leuştean. *Certifying Optimality of State Estimation Programs*, pages 301–314. Springer Berlin Heidelberg, Berlin, Heidelberg, 2003.

[Sim06] Dan Simon. *Optimal State Estimation: Kalman, H Infinity, and Nonlinear Approaches*. Wiley-Interscience, 2006.

[Sim10] D. Simon. Kalman filtering with state constraints: A survey of linear and nonlinear algorithms. *IET Control Theory Applications*, 4(8):1303–1318, August 2010.

[Sup] Supporting material. www.prismmodelchecker.org/files/fm19kf/.

[Tho76] C. L. Thornton. *Triangular Covariance Factorizations for Kalman filtering*. PhD thesis, University of California at Los Angeles, 1976.

[VD86] M. Verhaegen and P. Van Dooren. Numerical aspects of different kalman filter implementations. *IEEE Transactions on Automatic Control*, 31(10):907–917, October 1986.

[Was10] Larry Wasserman. *All of Statistics: A Concise Course in Statistical Inference*. Springer Publishing Company, Incorporated, 2010.

[WS04] Jon Whittle and Johann Schumann. Automating the implementation of kalman filter algorithms. *ACM Trans. Math. Softw.*, 30(4):434–453, December 2004.

[ZM15] Paul Zarchan and Howard Musoff. *Fundamentals of Kalman filtering : a practical approach*. American Institute of Aeronautics and Astronautics, Reston, VA, 4 edition, 2015.