On the Verification of Synthesized Kalman Filters

Ruben Gamboa* John Cowles* Jeff Van Baalen*
Computer Science Department
University of Wyoming
{ruben,cowles,jvb}@cs.uwyo.edu

Abstract

The Kalman Filter is a powerful technique that combines noisy information from predictions and observations to estimate an unknown value. This has tremendous practical applications; for example, Kalman Filters are used to estimate the position of a spaceship given a mathematical model of the ship's trajectory and inertial measurements. A feature of the Kalman Filter is that while it can be coded in remarkably few lines, practice has shown that it is difficult to code correctly. Moreover, for the sake of efficiency and ease of embedding into flight controllers and other devices, it is often necessary to code a Kalman Filter instance that is specific to an application. As a result, NASA has automated the process of generating these code instances. The complexity of the code generator is such that NASA has discounted the idea of proving the generator correct. Instead, it promotes the view that the generated code itself should be verified. NASA has tried various techniques to perform this validation. In this paper, we explore the use of ACL2 to validate the generated code.

1 Overview

The estimation problem can be described as follows. Suppose we are interested in a quantity x that can not be measured directly. If we are able to measure a related quantity z, how can we deduce the value of x from z? This situation arises naturally in many problems. For example, a pitot tube measures the difference in pressure between the air rushing against the front of a plane and the surrounding air. Airspeed is estimated from this pressure difference.

Because of its great practical importance, many techniques have been proposed for the solution of the estimation problem. The Kalman Filter is one of these techniques, and it is applicable when the variable of interest x changes linearly over time and the observable variable z is a linear function of x. Such systems can be described mathematically as follows:

*This work was supported by NASA grant NAG 2-1570

$$
\begin{aligned}
x_t &= \Phi x_{t-1} + w \quad (1)\\
z_t &= H x_t + v \quad (2)
\end{aligned}
$$

where w and v are random variables used to account for uncertainty in the model. In the general case, x and z are vectors. Φ is a matrix that describes how the vector x changes over time. Similarly, H is a (not necessarily square) matrix that describes how the vector x affects the observable vector z.

An interesting aspect of the problem is that once the value of x_t is known, it is possible to use the first equation to predict the value of $x_{t+1} \approx \Phi x_t$. This estimate is called the optimal "a priori" estimate, since it does not take into account the value of the observed vector z_{t+1}. After the vector x_{t+1} is known, it is possible to find a different estimate of x_{t+1} (essentially by solving the second linear equation). In this view, the Kalman Filter weighs these two estimates in order to find an optimal (linear) estimate of x_{t+1}. Moreover, since H is not necessarily square, the Kalman Filter can be viewed as a generalization of the least squares technique used to solve overconstrained systems of equations.

Analysis of the Kalman Filter requires sophisticated mathematics, including linear algebra, matrix calculus, and probability theory. Probability theory is needed to reason about the uncertainties in the vectors v and w, matrix calculus is needed to reason about optimality, and linear algebra is the language used in the formalization.

In the remainder of this paper we will present a formalizaton of the Kalman Filter in ACL2. This formalization is not complete in the following sense. While the key arguments of the correctness of the Kalman Filter have been verified, we have not created a complete mechanical formalization of matrix calculus and probability theory. Rather, we have assumed some important theorems from these fields. Continuing the ACL2 formalization of matrix calculus should not present any significant challenges, other than the usual problems with any mechanical formalization. Formalizing multivariate probability theory, on the other hand, presents a bigger challenge in ACL2. The appropriate way to do that is an interesting research question left to future work.

An important aspect of this work is to judge whether ACL2 can be an effective platform for verifying synthesized versions of the Kalman Filter. Such a judgement is suggested by the relative ease with which it proved the correctness of a specific Kalman Filter implementation in section 5. In [11], a mechanical formalization of Kalman Filters in Maude is presented. Maude is an executable specification system supporting both rewriting and membership equational logic [3]. There are many similarities between our approach and the one taken in [11]; indeed, we owe much to the presentation given there. However, a key point of departure is that the formalism in [11] proves the Kalman Filter in the context of an actual implementation; i.e., the proof proceeds by considering a specific program computing a specific filter. Our approach is to develop the mathematics first. The proof of the implementation then becomes a matter of associating the variables of the program with the mathematical en-

Module	#Lines	#Defns	#Thms	Section
linalg.lisp	1,077	14	120	2
array2.lisp	1,161	0	47	2
matrix.lisp	9,872	79	510	2
matalg.lisp	2,710	63	114	2
kalman-defs.lisp	754	21	82	3
kalman-proof.lisp	2,299	5	80	4
kalman-demo.lisp	161	1	6	5

Figure 1: Effort of work

tities of the problem. As may be expected, the mathematical formalization of the Kalman Filter was very challenging. However, when it came to proving a specific implementation, the payoff was immediate. The proof of the correctness of the mechanically generated algorithm was itself mechanical.

An idea of the complexity of the source files can be gleaned from figure 1. Clearly, the bulk of the complexity lies in the mathematical development. Note specifically the first four files, which present the formalization of linear algebra.

This paper is structured as follows. In section 2, we give a brief presentation of the formalization of linear algebra in ACL2. This formalization is described in more detail in [4]. In section 3, we present a formalization of the Kalman Filter in ACL2, including the necessary facts from probability theory. In section 4, we present a proof of the optimality of the Kalman Filter. In section 5 we show how this mathematical theory can be used to reason about a specific computer program implementing the Kalman Filter in a loop. Finally, in section 6 we address the assumptions made in the formalization.

2 Linear Algebra

The formalization of the Kalman Filter requires a large body of background knowledge about linear algebra. In this section, we provide a quick tour of the formalization of linear algebra. A more detailed account is presented in [4].

Our approach to formalizing linear algebra was two-fold. First, we used `encapsulate` to provide the definitions and theorems from linear algebra that are needed to prove the correctness of the Kalman Filter. The correctness proof proceeded from these encapsulated events. In parallel, we formalized actual matrix algebra using the encapsulated definitions and theorems as a target. In the ACL2 tradition, the second formalization is executable. That is, it is possible to define matrices, multiply them, find inverses and determinants, etc, so we can actually compute Kalman Filters. Thinking ahead to provinding fast computations, we decided to define matrices using ACL2 arrays, which support destructive read/write operations while providing a purely functional semantics.

In the remainder of this section, we will describe the abstract formalization: i.e., the definitions and theorems from linear algebra needed to reason about the Kalman Filter. Our purpose is to give the reader an idea of the theorems involved and to fix the notations for the remainder of this paper.

The predicate `m-matrixp` is used to recognize matrices of a given size. For convenience, we define the functions `l` and `c` which return the number of rows (i.e., lines) and columns of a matrix, respectively. Also introduced are the basic operations of matrix algebra, such as `m-+`, `m-*`, `s-*`, for matrix addition, multiplication, and scalar multiplication, respectively. The function `m-unary--` defines the additive inverse for matrices, and `m--` defines matrix subtraction in terms of `m-+` and `m-unary--`. Also defined is `m-inv`, the multiplicative inverse for matrices. The predicate `m-singular` tests if a matrix is singular; i.e., if it does not have an inverse. The remaining matrix operation defined is `m-trans` for transposing matrixes. The functions `m-zero` and `m-id` construct the zero and identity matrices of a given size.

In addition to these basic definitions, we provide many lemmas about these functions. Most of these will be familiar to ACL2 users. The following lemma, formalizing the associativity of matrix multiplication, is typical:

```
(defthm assoc-*
  (implies (and (equal (c P) (l Q))
                (equal (c Q) (l R)))
           (m-= (m-* (m-* P Q) R)
                (m-* P (m-* Q R)))))
```

In order to formalize the Kalman Filter, we required a comprehensive set of facts about linear algebra. Details of this formalization are in [4].

3 Formalizing the Kalman Filter

The Kalman Filter is a common solution to the estimation problem. It is applicable when the problem can be described as follows:

$$\begin{aligned} x_{k+1} &= \Phi_k x_k + w_k && (3)\\ z_k &= H_k x_k + v_k && (4)\\ E[w_k] &= 0 && (5)\\ E[v_k] &= 0 && (6)\\ E[w_k {w_i}^{\mathrm{T}}] &= \delta_{k-i} Q_k && (7)\\ E[v_k {v_i}^{\mathrm{T}}] &= \delta_{k-i} R_k && (8) \end{aligned}$$

The state vector x changes (almost) linearly over time, as specified by the matrix Φ. The random vector w denotes an imperfection in the linear model describing the change of x. This error vector is assumed to have zero mean, and to be uncorrelated over time. The covariance matrix Q describes the noise characteristics of w. The vector z is a vector of measurements. The matrix H explains how the measurements in z reflect the value of x. The measuring process is assumed to be imperfect, and v is a random vector modeling this imperfection.

As with the vector w, v is assumed to have zero mean and to be uncorrelated over time. Its noise characteristics are described by the covariance matrix R.

The Kalman Filter is a recursive process that continuously updates the current best estimate of the vector x by taking into account the previous best estimate of x and the current value of z. These two estimates are averaged, using weights derived from their relative uncertainties. These uncertainties are also updated, using the values of Q and R. Because the process depends only on the current estimate of x and the current observed value of z, as opposed to the entire history of estimates or observations, the Kalman Filter is of extreme practical importance, prompting Casti to count it as one of the five greatest results of 20th Century Mathematics [2].

We begin with $\overline{x}_0$, the initial estimate of the vector x, and $\overline{P}_0 = E[(x_0 - \overline{x}_0) \times (x_0 - \overline{x}_0)^{\mathrm{T}}]$ which measures our confidence in this initial estimate. Using $\overline{x}_0$ and H_0, we can compute $\overline{z_0}$ which is an estimate of the observable vector z at time 0. Since the vector z can be observed directly, we can judge the quality of our estimate $\overline{x}_0$ by considering the residual $z_0 - \overline{z}_0$. Moreover, we can use the information in this residual to correct the vector $\overline{x}_0$. If we limit ourselves to corrections that are linear functions of the residual, we can show that the optimal correction is given by the Kalman gain matrix, defined as

$$K_k = \overline{P}_k {H_k}^{\mathrm{T}} \left(H_k \overline{P}_k {H_k}^{\mathrm{T}} + R_k \right)^{-1}. \tag{9}$$

Thus, we can compute a better estimate of x_0 using the following equation:

$$\hat{x}_0 = \overline{x}_0 + K_0(z_0 - \overline{z}_0). \tag{10}$$

The accuracy of this estimate can be estimated as $P_0 = E[(x_0 - \hat{x}_0) \times (x_0 - \hat{x}_0)^{\mathrm{T}}]$, and that can be computed from the equation

$$P_k = (I - K_k H_k)\,\overline{P}_k. \tag{11}$$

Finally, using $\hat{x}_0$ and P_0, we can estimate the vector x_1 as

$$\overline{x}_1 = \Phi_0 \hat{x}_0. \tag{12}$$

Our confidence in this estimate is given by $\overline{P}_1 = E[(x_1 - \overline{x}_1) \times (x_1 - \overline{x}_1)^{\mathrm{T}}]$ which can be computed from the equation

$$\overline{P}_{k+1} = \Phi_k P_k {\Phi_k}^{\mathrm{T}} + Q_k. \tag{13}$$

This gives us an initial estimate of x_1 and a measure of the accuracy of this estimate. At this point we can repeat the process to find $\hat{x}_1$, the optimal estimate of x_1, etc.

The formal model of the Kalman Filter in ACL2 follows the description above very closely. The first step in the formalization is to introduce the input parameters to the process:

```
(encapsulate
 (((x-0) => *)                ; initial value of x
  ((phi *) => *)              ; steps through an iteration of x
  ((ww *) => *)               ; iteration step noise
  ((q *) => *)                ; covariance of step noise
  ((h *) => *)                ; matrix transforming observable to x
  ((v *) => *)                ; observation noise
  ((r *) => *)                ; covariance of observation noise
  ((xhatmin-0) => *)          ; initial guess for best estimate of x
  ((pminus-0) => *)           ; initial guess for covariance of estimate
  ((n) => *)                  ; dimension of x
  ((m) => *)                  ; dimension of y
  ((m-mean *) => *)           ; expected value of an expression
  )
 ...)
```

The vector x itself is not considered an input vector, since it can be computed using equation 3. Instead, the constant vector `xhatmin-0` is used to specify the initial value of x, i.e, $\overline{x}_0$. Similarly, z is not considered an input vector, since it is defined by equation 4. For example, x is defined as follows:

```
(defun x (k)
  (if (zp k)
      (x-0)
    (m-+ (m-* (phi (1- k)) (x (1- k)))
         (ww (1- k)))))
```

We now consider the constraints on these symbols. Some of the constraints have to do with the type of the objects, i.e., the dimension of the vectors and matrices. For example, we have that `phi` is an $n \times n$ matrix.

Other constraints state key assumptions used by the Kalman Filter. For example, we stated earlier that

$$R_k \quad = \quad E[v_k \times {v_k}^{\mathrm{T}}]. \tag{14}$$

This assumption is given below:

```
(defthm mean-of-v-vtrans
  (m-= (m-mean (m-* (v k) (m-trans (v k))))
       (r k)))
```

The next part of the ACL2 formalization defines the functions `pminus`, `pplus`, and `gain`, corresponding to $\overline{P}_k$, P_k, and K_k, respectively. A look at equations 13, 11, and 9 reveals that these functions are mutually recursive. However, it is possible to break the mutual recursion by unfolding some of the definitions. We could have used ACL2's support for mutual recursion to define these functions, but we considered it prudent to build our theory without mutual recursion, since ACL2's behavior for simple recursive functions is more predictable. Our approach can be illustrated with the definition of `pplus`:

```
(defun pplus (k)
  (if (zp k)
      (m-* (m-- (m-id (l (x k)))
                (m-* (m-* (pminus-0)
                          (m-* (m-trans (h k))
                               (m-inv
                                (m-+ (m-* (h k)
                                          (m-* (pminus-0)
                                               (m-trans (h k))))
                                     (r k)))))
                     (h k)))
           (pminus-0))
    (m-* (m-- (m-id (l (x k)))
              (m-* (gain-body k) (h k)))
         (pminus-body k))))
```

The recursive part of the definition follows equation 11 exactly. However, note the presence of `gain-body` and `pminus-body` instead of the expected `gain` and `pminus`. These are macros that expand the (recursive) definition of `gain` and `pminus` in terms of `pplus`:

```
(defmacro gain-body (k)
  `(m-* (pminus-body ,k)
        (m-* (m-trans (h ,k))
             (m-inv (m-+ (m-* (h ,k)
                              (m-* (pminus-body ,k)
                                   (m-trans (h ,k))))
                         (r ,k))))))

(defmacro pminus-body (k)
  `(if (zp ,k)
       (pminus-0)
     (m-+ (m-* (phi (1- ,k))
               (m-* (pplus (1- ,k))
                    (m-trans (phi (1- ,k)))))
          (q (1- ,k)))))
```

Note how these macros follow equations 9 and 13 closely.

The base case of the definition of `pplus` is similar to the recursive body of the function, except the bodies of `gain` and `pminus` are expanded completely in order to avoid any recursive references to `pplus`.

Once `pplus` is defined, it is possible to define the function `pminus` without using mutual recursion. In fact, this is precisely what the macro `pminus-body` did. Therefore, `pminus` can be defined directly with `pminus-body`. Similarly, it is possible to define `gain` using `gain-body`. However, in this case we prefer to write a new definition explicitly, using `pminus` instead of `pminus-body`:

```
(defun gain (k)
  (m-* (pminus k)
       (m-* (m-trans (h k))
            (m-inv (m-+ (m-* (h k)
                             (m-* (pminus k)
                                  (m-trans (h k))))
                        (r k))))))
```

It is useful to compare this definition with the definition of the macro `gain-body`.

Although these definitions suffice to introduce `pplus`, `pminus`, and `gain`, subsequent development of the proof is simplified if we have direct analogues of equations 13, 11, and 9. For this, we use ACL2's support for multiple definitions of a function. The following theorem is precisely equivalent to equation 11:

```
(defthm pplus-recdef
  (implies (and (integerp k)
                (<= 0 k))
           (equal (pplus k)
                  (m-* (m-- (m-id (l (x k)))
                            (m-* (gain k)
                                 (h k)))
                       (pminus k))))
  :hints ...
  :rule-classes ((:definition ...)))
```

Similar theorems give recursive definitions of `pminus` and `gain`.

A similar development can be used to define `xhat` and `xhatmin`, equivalent to $\hat{x}_k$ and $\overline{x}_k$, respectively. As before, these two functions appear mutually recursive, but the mutual recursion can be broken by unfolding the definition of `xhat`. Thus, we define `xhatmin` as follows:

```
(defun xhatmin (k)
  (if (zp k)
      (xhatmin-0)
    (m-* (phi (1- k)) (xhat-body (1- k)))))
```

This follows equation 12 very closely. Again notice how `xhat-body` is used instead of `xhat`, to unfold the body of that definition. The function `xhat` is defined as follows:

```
(defmacro xhat-body (k)
  `(m-+ (xhatmin ,k)
        (m-* (gain ,k)
             (m-- (z ,k)
                  (m-* (h ,k) (xhatmin ,k))))))

(defun xhat (k)
  (xhat-body k))
```

As before, we can give a recursive definition of `xhatmin` that mirrors equation 12 precisely.

With these definitions, it is possible to state the final assumptions made by the Kalman Filter. In particular, it is assumed that the observation noise is orthogonal to the error in $\overline{x}_k$, and that the process noise is orthogonal to the error in $\hat{x}_k$. A representative constraint is the following:

```
(defthm mean-of-x-xhatmin*vtrans
  (m-= (m-mean (m-* (m-+ (x k)
                         (m-unary-- (xhatmin k)))
                    (m-trans (v k))))
       (m-zero (n) (m))))
```

In addition, now that `xhatmin` is defined, it is possible to state the initial constraint on `pminus-0`, that it is our best estimate on the error of `xhatmin-0`:

```
(defthm pminus-0-def
  (m-= (pminus-0)
       (m-mean (m-* (m-- (x 0) (xhatmin-0))
                    (m-trans (m-- (x 0) (xhatmin-0))))))
  :hints ...)
```

The remaining theorems specify the behavior of the `m-mean` function, which computes the expected value of a matrix expression involving random vectors. A typical theorem states how to take the expected value of a sum:

```
(defthm mean-+
  (implies (and (equal (l p) (l q))
                (equal (c p) (c q)))
           (equal (m-mean (m-+ p q))
                  (m-+ (m-mean p) (m-mean q)))))
```

As long as the sum is well-defined, according to this theorem, the expected value of the sum is the same as the sum of the expected values.

However, there is at present no support for random variables in ACL2. Hence the formalization of `m-mean` is incomplete. In particular, the following theorems are problematic:

```
(defthm mean-*
  (implies (equal (c p) (l q))
           (equal (m-mean (m-* p q))
                  (m-* (m-mean p) (m-mean q)))))

(defthm mean-delete
  (equal (m-mean p) p))
```

From the second theorem, it is clear that the only ACL2 function that will satisfy the requirements of the `m-mean` function is the identify function. The reason is that `mean-delete` lacks an important hypothesis, namely that the expression `p` not contain a random variable. Similarly, the first theorem omits the hypothesis that the expressions `p` and `q` be independent. These hypothesis can not be stated in ACL2. Therefore, we disable these theorems, allowing ACL2 to use them only under very controlled circumstances.

The need to control the application of these theorems undermines the correctness results derived in the next section. What is needed is a way to reason about random variables directly in ACL2. Notice that a random variable is usually defined as a function from an event space to the reals. Thus reasoning about random variables is more natural in a higher-order logic. Capturing their salient properties in a context that can be used in ACL2 is a major challenge.

4 Correctness of the Kalman Filter

In this section, we present a proof in ACL2 of the correctness of the Kalman Filter. Three main results are required to show that the Kalman Filter is correct:

- $\hat{x}_k$ is the best possible (linear) estimate of x_k;
- $\overline{P}_k = E[(x_k - \overline{x}_k) \times (x_k - \overline{x}_k)^{\mathrm{T}}]$; and
- $P_k = E[(x_k - \hat{x}_k) \times (x_k - \hat{x}_k)^{\mathrm{T}}]$.

The first claim states explicitly the correctness of the Kalman Filter approximation. The other two claims are needed to prove the first.

Therefore, we begin with a proof of the second and third claims. As it turns out, these claims depend on each other. As was the case with the mutually recursive definitions in section 3, it is possible to break this dependency.

Consider the claim that $\overline{P}_k = E[(x_k - \overline{x}_k) \times (x_k - \overline{x}_k)^{\mathrm{T}}]$. Certainly, this is the case when $k = 0$, since the claim is assumed to be true for the initial values in the assumed constraint `pminus-0-def`. Formally in ACL2, we have the following theorem:

```
(defthm pminus-as-mean-case-0
  (implies (= k 0)
           (m-= (pminus k)
                (m-mean (m-* (m-- (x k) (xhatmin k))
                             (m-trans (m-- (x k) (xhatmin k)))))))
  :hints ...)
```

For positive k, we can proceed as follows. Expanding the definitions of x_k and $\overline{x}_k$, we find that $x_k - \overline{x}_k = \Phi_{k-1}(x_{k-1} - \hat{x}_{k-1}) + w_{k-1}$. Therefore, for positive k we have that

$$\begin{aligned}
&(x_k - \overline{x}_k) \times (x_k - \overline{x}_k)^{\mathrm{T}} \\
&\quad = \quad \Phi_{k-1} \times (x_{k-1} - \hat{x}_{k-1}) \times (x_{k-1} - \hat{x}_{k-1})^{\mathrm{T}} \times {\Phi_{k-1}}^{\mathrm{T}} + \\
&\qquad\quad (\Phi_{k-1} \times (x_{k-1} - \hat{x}_{k-1})) \times {w_{k-1}}^{\mathrm{T}} + \\
&\qquad\quad w_{k-1} \times (x_{k-1} - \hat{x}_{k-1})^{\mathrm{T}} \times (\Phi_{k-1})^{\mathrm{T}} + \\
&\qquad\quad w_{k-1} \times {w_{k-1}}^{\mathrm{T}}.
\end{aligned} \tag{15}$$

Taking the expected value of both sides of this equation yields

$$\begin{aligned}
&E[(x_k - \overline{x}_k) \times (x_k - \overline{x}_k)^{\mathrm{T}}] \\
&\quad = \quad E[\Phi_{k-1} \times (x_{k-1} - \hat{x}_{k-1}) \times (x_{k-1} - \hat{x}_{k-1})^{\mathrm{T}} \times {\Phi_{k-1}}^{\mathrm{T}}] + \\
&\qquad\quad\ E[(\Phi_{k-1} \times (x_{k-1} - \hat{x}_{k-1})) \times {w_{k-1}}^{\mathrm{T}}] + \\
&\qquad\quad\ E[w_{k-1} \times (x_{k-1} - \hat{x}_{k-1})^{\mathrm{T}} \times (\Phi_{k-1})^{\mathrm{T}}] + \\
&\qquad\quad\ E[w_{k-1} \times {w_{k-1}}^{\mathrm{T}}] \qquad (16) \\
&\quad = \quad \Phi_{k-1} \times E[(x_{k-1} - \hat{x}_{k-1}) \times (x_{k-1} - \hat{x}_{k-1})^{\mathrm{T}}] \times {\Phi_{k-1}}^{\mathrm{T}} + \\
&\qquad\quad\ Q_{k-1}. \qquad (17)
\end{aligned}$$

This step used the properties of $E[\cdot]$ as well as the Kalman Filter assumptions `mean-of-x-xhat*wtrans` and `mean-of-w*trans-of-x-xhat`. Next notice that $E[(x_{k-1} - \hat{x}_{k-1}) \times (x_{k-1} - \hat{x}_{k-1})^{\mathrm{T}}]$ is equal to P_{k-1} is precisely the third claim specialized for time $k-1$. Using this specialized claim, it is easy to complete the proof of the second claim. In ACL2, we have proved the following:

```
(defthm pminus-as-mean-almost
  (implies (and (integerp k)
                (< 0 k)
                (m-= (pplus (1- k))
                     (m-mean (m-* (m-- (x (1- k))
                                       (xhat (1- k)))
                                  (m-trans (m-- (x (1- k))
                                                (xhat (1- k))))))))
           (m-= (pminus k)
                (m-mean (m-* (m-- (x k) (xhatmin k))
                             (m-trans (m-- (x k) (xhatmin k)))))))
  :hints ...)
```

Notice how the theorems `pminus-as-mean-case-0` and `pminus-as-mean-almost` are reminiscent of the base and induction steps of a proof by induction. The only difference is that the inductive hypothesis in `pminus-as-mean-almost` is about P_{k-1} instead of $\overline{P}_{k-1}$. This is as expected, since the functions are mutually recursive. These two theorems, which combine to form a weak version of the second claim, are enough to prove the third claim. Once that proof is done, it will be trivial to complete the proof of the second claim.

So we turn our attention to the third claim. We wish to show that $P_k = E[(x_k - \hat{x}_k) \times (x_k - \hat{x}_k)^{\mathrm{T}}]$. Expanding the definitions of x_k and $\hat{x}_k$, we find that $x_k - \hat{x}_k = (I - K_k H_k) \times (x_k - \overline{x}_k) - K_k v_k$. Transposing and carrying out the multiplications, it follows that

$$\begin{aligned}
&(x_k - \hat{x}_k) \times (x_k - \hat{x}_k)^{\mathrm{T}} \\
&\quad = \quad (I - K_k H_k) \times (x_k - \overline{x}_k) \times (x_k - \overline{x}_k)^{\mathrm{T}} \times (I - K_k H_k)^{\mathrm{T}} + \\
&\qquad\quad\ (-(I - K_k H_k) \times (x_k - \overline{x}_k) \times {v_k}^{\mathrm{T}} {K_k}^{\mathrm{T}}) + \\
&\qquad\quad\ (-K_k v_k \times (x_k - \overline{x}_k)^{\mathrm{T}} \times (I - K_k H_k)^{\mathrm{T}}) + \\
&\qquad\quad\ K_k v_k \times {v_k}^{\mathrm{T}} {K_k}^{\mathrm{T}}. \qquad (18)
\end{aligned}$$

Taking the expected value of both sides and dropping the zero terms, we can conclude that

$$\begin{aligned}
&E[(x_k - \hat{x}_k) \times (x_k - \hat{x}_k)^{\mathrm{T}}] \\
&\quad = \ (I - K_k H_k) \times E[(x_k - \overline{x}_k) \times (x_k - \overline{x}_k)^{\mathrm{T}}] \times (I - K_k H_k)^{\mathrm{T}} + \\
&\qquad K_k R_k {K_k}^{\mathrm{T}}. \qquad (19)
\end{aligned}$$

Notice the expression $E[(x_k - \overline{x}_k) \times (x_k - \overline{x}_k)^{\mathrm{T}}]$ above. Using the theorems `pminus-as-mean-almost` and `pminus-as-mean-case-0`, we can reduce that to $\overline{P}_k$, provided we can establish that $P_{k-1} = E[(x_{k-1} - \hat{x}_{k-1}) \times (x_{k-1} - \hat{x}_{k-1})^{\mathrm{T}}]$. This latter claim is precisely equal to the induction hypothesis that we will have when we try to prove the third claim by induction. Thus, we can assume that this is already established and simplify our expression as follows:

$$\begin{aligned}
&E[(x_k - \hat{x}_k) \times (x_k - \hat{x}_k)^{\mathrm{T}}] \\
&\quad = \ (I - K_k H_k) \times \overline{P}_k \times (I - K_k H_k)^{\mathrm{T}} + K_k R_k {K_k}^{\mathrm{T}} \qquad (20) \\
&\quad = \ \overline{P}_k + (-K_k H_k \overline{P}_k) + (-\overline{P}_k {H_k}^{\mathrm{T}} {K_k}^{\mathrm{T}}) + \\
&\qquad K_k H_k \overline{P}_k {H_k}^{\mathrm{T}} {K_k}^{\mathrm{T}} + K_k R_k {K_k}^{\mathrm{T}} \qquad (21) \\
&\quad = \ \overline{P}_k + (-K_k H_k \overline{P}_k) + (-\overline{P}_k {H_k}^{\mathrm{T}} {K_k}^{\mathrm{T}}) + \\
&\qquad K_k \times (H_k \overline{P}_k {H_k}^{\mathrm{T}} + R_k) \times {K_k}^{\mathrm{T}} \qquad (22) \\
&\quad = \ \overline{P}_k + (-K_k H_k \overline{P}_k) + (-\overline{P}_k {H_k}^{\mathrm{T}} {K_k}^{\mathrm{T}}) + \overline{P}_k {H_k}^{\mathrm{T}} {K_k}^{\mathrm{T}} \qquad (23)
\end{aligned}$$

Notice that the last step is justified only if the matrix $(H_k \overline{P}_k {H_k}^{\mathrm{T}} + R_k)$ is non-singular. In fact, its inverse is part of the definition of K_k. In the formal ACL2 theory, that this inverse exists was added explicitly as an axiom; we will have more to say about this in section 6. At this point, the proof is nearly complete. It is only necessary to rearrange the terms to make the final argument:

$$\begin{aligned}
&E[(x_k - \hat{x}_k) \times (x_k - \hat{x}_k)^{\mathrm{T}}] \\
&\quad = \ \overline{P}_k - K_k H_k \overline{P}_k \qquad (24) \\
&\quad = \ (I - K_k H_k) \times \overline{P}_k \qquad (25) \\
&\quad = \ P_k. \qquad (26)
\end{aligned}$$

Next, the third correctness claim can be proved using induction on k:

```
(defthm pplus-as-mean
   (implies (and (integerp k)
                 (<= 0 k))
            (m-= (pplus k)
                 (m-mean (m-* (m-- (x k) (xhat k))
                              (m-trans (m-- (x k) (xhat k)))))))
   :hints ...)
```

Moreover, once this theorem is established, it is trivial to complete the proof of the second correctness claim:

```
(defthm pminus-as-mean
  (implies (and (integerp k) (<= 0 k))
           (m-= (pminus k)
                (m-mean (m-* (m-- (x k) (xhatmin k))
                             (m-trans (m-- (x k) (xhatmin k)))))))
  :hints ...)
```

Now that these two claims have been verified, we can direct our attention to the main claim, namely that $\hat{x}_k$ is the best possible (linear) estimate of x_k. Before proceeding, however, we have to state formally what we mean by best possible estimate.

Recall that $\overline{x}_k$ is supposed to capture the best "a priori" estimate of x_k. This is the best estimate we can make before taking into account the observable vector z_k. Initially, the best a priori estimate is given by $\overline{x}_0$ which is input to the problem. Subsequently, it can be computed by $BAP_k = \Phi_{k-1}B_{k-1}$, where B_{k-1} is the best possible estimate of x_{k-1}. Moreover, once we have an a priori estimate, we find the best possible estimate by adding to the best a priori estimate a linear combination of the difference between the observed vector z_k and the projected observable H_kBAP_k. That is, $B_k = BAP_k + Y \times (z_k - H_kBAP_k)$. The best possible estimate can be found by choosing an appropriate matrix Y, and we can do this by differentiating $E[(x_k - B_k) \times (x_k - B_k)^{\mathrm{T}}]$ with respect to Y and setting the result to 0.

To carry out this argument in ACL2, we need to specialize the higher-order functions B_k and BAP_k to the specific vector x as follows:

```
(defstub best-estimate-of-x (*) => *)

(defun best-prior-estimate-of-x (k)
  (if (zp k)
      (xhatmin k)
    (m-* (phi (1- k))
         (best-estimate-of-x (1- k)))))
```

Notice how `best-prior-estimate-of-x` is defined in terms of `best-estimate-of-x`. To state that `best-estimate-of-x` in turn depends on `best-prior-estimate-of-x`, we introduce the general form of a solution:

```
(defun result-form (y Xp k)
  (m-+ Xp (m-* y (m-- (z k) (m-* (h k) Xp)))))
```

The intent captured by this definition is that the space of possible estimates of x_k is spanned by the vectors Xp (which will correspond to the best a priori estimate), and a linear combination of the difference between the observable z and the predicted observable at Xp. The derivative of the covariance of this estimate could, in principle, be calculated directly. However, that would involve the development of a significant subset of matrix calculus. We chose, therefore, to introduce the derivative directly, without a formal justification:

```
(defun result-form-error-derivative (y Xp k)
  (m-+ (s-* 2 (m-mean (m-* (m-- Xp (x k))
                           (m-trans (m-- (z k)
                                         (m-* (h k) Xp))))))
       (s-* 2 (m-* y
                   (m-mean (m-* (m-- (z k)
                                     (m-* (h k) Xp))
                                (m-trans (m-- (z k)
                                              (m-* (h k)
                                                   Xp)))))))))
```

Once the derivative is found, we can finish stating formally the relationship between `best-estimate-of-x` and `best-prior-estimate-of-x`:

```
(defaxiom best-estimate-of-x-def
  (implies (and (m-= (best-prior-estimate-of-x k) Xp)
                (m-= (result-form-error-derivative y Xp k)
                     (m-zero (n) (m))))
           (m-= (best-estimate-of-x k)
                (result-form y Xp k))))
```

Notice that this relationship must be assumed (not proved), since we did not develop the theory of matrix calculus sufficiently to show when a matrix function achieves its minimum. These assumptions will be discussed in section 6.

To complete the proof, we must show that `best-prior-estimate-of-x` is equal to $\overline{x}_k$, and that `best-estimate-of-x` is equal to $\hat{x}_k$. Because of the structure of `best-prior-estimate-of-x` and `best-estimate-of-x`, we are faced once again with the task of proving two claims that depend on each other. As before, it is possible to separate these claims by first proving a weaker version of one of them.

In this case, it is easy to show that `best-prior-estimate-of-x` is equal to $\overline{x}_k$, as long as we assume that the `best-estimate-of-x` of $k-1$ is equal to $\hat{x}_{k-1}$. The proof simply opens up the respective definitions:

```
(defthm xhatmin=best-prior-almost
  (implies (m-= (xhat (1- k))
                (best-estimate-of-x (1- k)))
           (m-= (xhatmin k)
                (best-prior-estimate-of-x k)))
  :hints ...)
```

Now, consider the expression

```
(result-form-error-derivative (gain k) (xhatmin k) k)
```

We wish to show that this expression is equal to 0. Opening up the expression yields the sum

$$\begin{aligned} &2E[(\overline{x}_k - x_k) \times (z_k - H_k\overline{x}_k)^{\mathrm{T}}] + \\ &2K_k \times E[(z_k - H_k\overline{x}_k) \times (z_k - H_k\overline{x}_k)^{\mathrm{T}}]. \end{aligned} \tag{27}$$

We proceed by considering the first term of the sum:

$$
\begin{aligned}
& E[(\overline{x}_k - x_k) \times (z_k - H_k\overline{x}_k)^{\mathrm{T}}] \\
&= -E[(x_k - \overline{x}_k) \times (H_k x_k - H_k\overline{x}_k + v_k)^{\mathrm{T}}] && (28)\\
&= -E[(x_k - \overline{x}_k) \times (H_k x_k - H_k\overline{x}_k)^{\mathrm{T}}] - E[(x_k - \overline{x}_k) \times {v_k}^{\mathrm{T}}] && (29)\\
&= -E[(x_k - \overline{x}_k) \times (H_k x_k - H_k\overline{x}_k)^{\mathrm{T}}] && (30)\\
&= -E[(x_k - \overline{x}_k) \times (x_k - \overline{x}_k)^{\mathrm{T}} {H_k}^{\mathrm{T}}] && (31)\\
&= -E[(x_k - \overline{x}_k) \times (x_k - \overline{x}_k)^{\mathrm{T}}] \times {H_k}^{\mathrm{T}} && (32)\\
&= -\overline{P}_k {H_k}^{\mathrm{T}} && (33)
\end{aligned}
$$

Next, we consider the second term of the sum:

$$
\begin{aligned}
& K_k \times E[(z_k - H_k\overline{x}_k) \times (z_k - H_k\overline{x}_k)^{\mathrm{T}}] \\
&= K_k \times E[(H_k x_k - H_k\overline{x}_k + v_k) \times (H_k x_k - H_k\overline{x}_k + v_k)^{\mathrm{T}}] && (34)\\
&= K_k \times E[H_k(x_k - \overline{x}_k) \times (x_k - \overline{x}_k)^{\mathrm{T}} {H_k}^{\mathrm{T}} + \\
&\quad H_k(x_k - \overline{x}_k){v_k}^{\mathrm{T}} + v_k(x_k - \overline{x}_k)^{\mathrm{T}} {H_k}^{\mathrm{T}} + v_k{v_k}^{\mathrm{T}}] && (35)\\
&= K_k \times (H_k E[(x_k - \overline{x}_k) \times (x_k - \overline{x}_k)^{\mathrm{T}}]{H_k}^{\mathrm{T}} + E[v_k{v_k}^{\mathrm{T}}]) && (36)\\
&= K_k \times (H_k\overline{P}_k{H_k}^{\mathrm{T}} + R_k) && (37)\\
&= \overline{P}_k{H_k}^{\mathrm{T}} \times (H_k\overline{P}_k{H_k}^{\mathrm{T}} + R_k)^{-1} \times (H_k\overline{P}_k{H_k}^{\mathrm{T}} + R_k) && (38)\\
&= \overline{P}_k{H_k}^{\mathrm{T}} && (39)
\end{aligned}
$$

The last step is justified only when the matrix $(H_k\overline{P}_k{H_k}^{\mathrm{T}} + R_k)$ is non-singular. As stated earlier, this is assumed explicitly in the ACL2 proof.

With the two results above, it follows that the expression

```
(result-form-error-derivative (gain k) (xhatmin k) k)
```

is indeed equal to the zero matrix. The ACL2 theorem is as follows:

```
(defthm gain-minimizes-error
  (implies (and (integerp k) (<= 0 k))
           (m-= (result-form-error-derivative (gain k) (xhatmin k) k)
                (m-zero (n) (m))))
  :hints ...)
```

Now it is easy to show that $\hat{x}_k$ is of the form given by `result-form`. This, together with `gain-minimizes-error`, is sufficient to show that $\hat{x}_k$ is the optimal estimate for x_k. The proof simply uses `gain-minimizes-error` and the lemma that $\overline{x}_k$ is the best a priori estimate of x_k, which was proved in `xhatmin=best-prior-almost`:

```
(defthm xhat=best-estimate
  (implies (and (integerp k)
                (<= 0 k))
           (m-= (xhat k)
                (best-estimate-of-x k)))
  :hints ...)
```

Once this theorem is found, it is trivial to combine it with `xhatmin=best-prior-almost` to find that $\overline{x}_k$ is the best a priori estimate of x_k, with no assumptions on $\hat{x}_k$:

```
(defthm xhatmin=best-prior
  (implies (and (integerp k)
                (<= 0 k))
           (m-= (xhatmin k)
                (best-prior-estimate-of-x k)))
  :hints ...)
```

This concludes the formalization in ACL2 of the mathematical theory of Kalman Filters. In the next section, we will use this formalization to prove a specific algorithm computing the Kalman Filter is correct.

5 Verifying a Kalman Filter Loop Invariant

The following program, taken from [11] illustrates a Kalman Filter implementation as may be generated by NASA's software:

input xhatmin, pminus;
for k ← 0..n **do**
  gain ← pminus * mtrans(h) * minv(h * pminus * mtrans(h) + r);
  xhat ← xhatmin + (gain * (z - (h * xhatmin)));
  pplus ← (id(n) - gain * h) * pminus;
  xhatmin ← phi * xhat;
  pminus ← phi * (pplus * mtrans(phi)) + q;
end for

From first principles, proving the correctness of this program would be very hard. But using the mathematical theory developed in the previous sections, we will be able to present a simple proof of the correctness of this loop. In fact, the proof itself is little more than an association between the variables in the program and the mathematical entities of the previous sections, and this association can be made in the form of hints that are automatically generated along with the code. Such an approach is already in use to help human readers perform code inspections of the generated code.

To prove this algorithm correct in ACL2, we will convert the code to ACL2's functional notation. We chose to represent the body of the loop as a Lisp function that takes two arguments, corresponding to $\overline{x}_k$ and $\overline{P}_k$, and returns five arguments, corresponding to K_k, $\hat{x}_k$, P_k, $\overline{x}_{k+1}$, and $\overline{P}_{k+1}$:

```
(defun kalman-loop-body (xhatmin-prev pminus-prev k)
  (let* ((gain (m-* pminus-prev
                    (m-* (m-trans (h k))
                         (m-inv (m-+ (m-* (h k)
                                          (m-* pminus-prev
                                               (m-trans (h k))))
                                     (r k))))))
```

```
        (xhat (m-+ xhatmin-prev
                   (m-* gain
                        (m-- (z k)
                             (m-* (h k) xhatmin-prev)))))
        (pplus (m-* (m-- (m-id (n))
                         (m-* gain (h k)))
                    pminus-prev))
        (xhatmin (m-* (phi k) xhat))
        (pminus (m-+ (m-* (phi k)
                          (m-* pplus
                               (m-trans (phi k))))
                     (q k))))
   (mv gain xhat pplus xhatmin pminus)))
```

The correctness of this code can be verified by the following theorem:

```
(defthm kalman-loop-invariant
     (implies (and (integerp k)
                   (<= 0 k)
                   (equal xhatmin-prev (xhatmin k))
                   (equal pminus-prev (pminus k)))
              (mv-let (gain xhat pplus xhatmin pminus)
                      (kalman-loop-body xhatmin-prev pminus-prev k)
                      (and (equal gain (gain k))
                           (equal xhat (xhat k))
                           (equal pplus (pplus k))
                           (equal xhatmin (xhatmin (1+ k)))
                           (equal pminus (pminus (1+ k))))))
     :hints ...)
```

The omitted hints simply invoke five lemmas that establish the relationship between each program variable and its associated mathematical entity. All of these lemmas were easy to automate; in fact, the decision to have the lemmas, as opposed to simply introducing the theorem directly, was based solely on style.

6 The Hidden Assumptions

6.1 Existence of Inverses

As mentioned in the previous section we make the explicit assumption that certain matrices are non-singular. Specifically, we make the following assumption:

```
(skip-proofs
 (defthm non-singular-gain-component-2
   (not (m-singular (m-+ (r k)
                         (m-* (h k)
                              (m-* (pminus k)
                                   (m-trans (h k)))))))))
```

In principle, this theorem could be moved into the `encapsulate` defining the functions `r`, `h`, and `pminus`. But in a pragmatic sense, this is not necessary. At each step, the algorithm computes the inverse of this matrix. If the inverse can be found, the matrix is indeed non-singular. On the other hand, if the inverse can not be found, the algorithm will recognize the situation and return an error.

However, handling the error in the proof would complicate the entire formalization. One thing we would like to see is support for exceptions in ACL2. In particular, if the given matrix were ever not to have an inverse, the code could throw an exception. Proofs would be partial. In particular, when the theorem

```
(defthm thm
  form)
```

is proved in ACL2, it means that the evaluation of `form` is not `nil`. With support for exceptions, ACL2 could introduce the event

```
(defweakthm thm2
  form2)
```

which asserts that either `form2` is not equal to `nil` or the evaluation of `form2` results in an exception. This would mark a significant departure from ACL2 practice, and we present it to the workshop for discussion.

It should also be noted that under suitable starting conditions, all the matrices of the above form can be shown to be positive definite, and hence to have inverses. Hence, it would be possible to address this issue directly at the expense of introducing more matrix theory.

6.2 Determining Optimality

Another assumption used in the proof concerns the optimality criterion for the solution. In particular, we have the following axiom in ACL2:

```
(defaxiom best-estimate-of-x-def
  (implies (and (m-= (best-prior-estimate-of-x k) Xp)
                (m-= (result-form-error-derivative y Xp k)
                     (m-zero (n) (m))))
           (m-= (best-estimate-of-x k)
                (result-form y Xp k))))
```

This axiom makes two assumptions:

1. The function `result-form-error-derivative` is zero precisely at the critical points of `result-form`. This is essentially a claim that it really computes the derivative of the error in `result-form` (as its name suggests), but keep in mind that derivatives here are in the context of matrix calculus.

2. All critical points of `result-form` are in fact minimums.

To relieve the first assumption would require a formalization of matrix calculus in ACL2. Our prior experience formalizing aspects of univariate calculus suggests that this in fact possible. However, since the complexity of matrix calculus is far greater than that of the univariate case, our experience also suggests that such an undertaking will be a long one. For pragmatic reasons, therefore, we decided not to undertake this formalization at this time, choosing instead to evaluate whether the formalization would pay off in applications.

The second assumption is more troubling. Again, the reason why this is valid lies in the fact that the matrices in question are positive definite. However, the proof effort required to show this is considerable.

6.3 Random Variables

The most troubling assumption lies in the definition of the function `m-mean`, or more precisely in the treatment of random variables such as v_k. A random variable is a function, but here it is formalized as a number.

So the essential question is the following: How can we formalize random variables in ACL2? A possible solution is suggested by the formalization of calculus in [5]. In that paper, derivatives were formalized by having two functions, f and fd and explicitly adding the theorem that fd was the derivative of f. In particular, no (second-order) "derivative" operator was ever defined.

Similarly, we may be able to reason about the random variable x as the function $x(k)$, the kth observation of the variable x. The function xm would be introduced and shown to be related to $x(k)$ in a way that corresponds to the intuitive assertion $xm = E[x]$. This relationship may be captured in ACL2 using non-standard probability theory [10].

7 Conclusions

We have used ACL2 to verify a program that computes the Kalman Filter. To simplify the proof of the program, we separated the mathematical argument regarding the correctness of the Kalman Filter from the correctness of the actual algorithm. The correctness of the actual algorithm followed easily.

It would be premature, however, to expect the algorithm correctness proof to be so simple each time. The main challenge of the proof presented in section 4 was in associating the variables of the program with the relevant mathematical entities. However, in a realistic setting this association may be considerably more difficult. For one thing, the mathematical entities may not appear as simple variables in the code. The program in section 5 used mathematical operators, such as `m-+` and `m-*` directly. However, a program may choose to implement matrix addition or matrix multiplication quite differently: instead of calling functions that can be equated with these operations, a program may perform the operations in-line, even to the extent of separating a (fixed-length) vector into its scalar values and putting each scalar in a separate program variable.

The formalization of the mathematics behind the Kalman Filter in ACL2 posed a serious challenge. Formalizing linear algebra proved to be tractable, although figure 1 clearly illustrates the enormity of the task. We do not have complete formalizations of the remaining mathematics needed to derive the Kalman Filter, namely matrix calculus and probability theory. Formalizing matrix calculus should be straightforward in principle, but it will be a significant task in practice. Formalizing probability theory in ACL2 poses a bigger conceptual challenge, and it may require a significant research component.

References

[1] R. G. Brown and P. Hwang. *Introduction to Random Signals and Applied Kalman Filtering: with MATLAB Excercises and Solutions*. John Wiley & Sons, 1997.

[2] J. Casti. *Five More Golden Rules*. Wiley, 2000.

[3] M. Clabel, F. Duran, S. Eker, P. Lincoln, N. Marti-Oliet, J. Meseguer, and J. Quesada. Maude: specification and programming in rewriting logic. Technical report, SRI International, January 1999.

[4] J. Cowles, R. Gamboa, and J. Van Baalen. Using ACL2 arrays to formalize matrix algebra. Under review, 2003.

[5] R. Gamboa. Continuity and differentiability in ACL2. In M. Kaufmann, P. Manolios, and J S. Moore, editors, *Computer-Aided Reasoning: ACL2 Case Studies*, chapter 18. Kluwer Academic Press, 2000.

[6] R. E. Kalman. A new approach to linear filtering and prediction problems. *Transactions of the ASME–Journal of Basic Engineering*, 82(Series D):35–45, 1960.

[7] M. Kaufmann, P. Manolios, and J S. Moore. *Computer Aided Reasoning: An Approach*. Kluwer Academic Publishers, 2000.

[8] B. Kuo. *Digital Control Systems*. Holt, Rinehart and Winston, 1980.

[9] P. S. Maybeck. *Stochastic models, estimation, and control*, volume 141 of *Mathematics in Science and Engineering*. Academic Press, 1979.

[10] E. Nelson. *Radically Elementary Probability Theory*. Princeton University Press, 1987.

[11] G. Rosu and J. Whittle. Towards certifying domain-specific properties of synthesized code. In *Proc of the 17th International Conference on Automated Software Engineering*, 2002.

[12] H. Stark and J. Woods. *Probability, Random Processes, and Estimation Theory for Engineers*. Prentice Hall, 1986.

[13] G. Welch and G. Bishop. The kalman filter home page. `http://www.cs.unc.edu/~welch/kalman`.