1 Introduction

In 1962 during the height of the Cold War, President John F. Kennedy delivered a famous speech, declaring "we choose to go to the moon." With these words, NASA is put under immense pressure to put the US – and humanity – on the moon. Yet, despite the support and confidence of Americans, technical and statistical issues exist. Rockets must follow precise calculated trajectories from Earth to the moon and back. Math equations do not work precisely enough; assumptions do not hold. Scientists must come up with new models that are able to make precise calculations that determine the spacecraft's true location despite irregular and randomly noisy data from the spaceship.

Enter Rudolph Kalman. Due to his taking 241 (or something equivalent), he is able to approximate continuously changing systems that output uncertain information.

We will explore the inner workings of the Kalman filter and how to implement it using Julia.

2 How Kalman Filters Work

2.1 Notational Note

For sections 2 and 3, we will mostly refer to the covariance matrices as Σ instead of the more common P. This is because I want to differentiate between Σ and P such that Σ is a semi-positive matrix with 1s along the diagonal. In reality, there is not much difference between the usage of Σ and P as they both describe covariances. In our coding section, we will be using P, which does not deviate from this derivation of the Kalman filter.

2.2 The State

Each continuously changing system has a state $\vec{x}$ that describes position and velocity. Let

$$\vec{x} = \begin{pmatrix} p \\ v \end{pmatrix}$$

such that $p \in \mathbb{R}$ represents position and $v \in \mathbb{R}$ represents velocity. Because there is variances between p, v, they can be different values. However, we do know that there exists a most likely p, v given the state and past data.

Here, for our Kalman filter, we assume that position and velocity and both random and *Gaussian* distributed. Thus, there exists a

$$\vec{\mu} = \begin{pmatrix} \mu_p \\ \mu_v \end{pmatrix}$$

such that $\vec{\mu}$ is the most likely state, or the *mean state estimate*, and at the center of the distribution (and $\mu_p, \mu_v \in \mathbb{R}$ are the mean state estimates for the position and velocity respectively). We also have σ_p^2 and σ_v^2 representing uncertainty, and they define the state covariance matrix

$$P = \begin{pmatrix} \sigma_p^2 & \sigma_{pv} \\ \sigma_{pv} & \sigma_v^2 \end{pmatrix}.$$

These values can be derived from our assumption of Gaussian distributed variables and statistics' principles.

We also note that sometimes, p and v are correlated. For instance, in our spaceship example, a spaceship may be faster 30 miles above the ground compared to 10 miles. Note that this is important in gathering uncertain and sparse information to make the best predictions.

2.3 Covariance Matrix and Eigenvalues

We can then write a covariance matrix for each pair of comparisons. A covariance matrix Σ determines correlation between state variables. We denote Σ_{ij} as the correlation between the ith and jth state variable. In our current example, we have two state variables, position and velocity, so Σ determines the correlation between position and velocity.

We can further explore some properties of these covariance matrices. Since correlation is commutative, we have that $\Sigma_{ij} = \Sigma_{ji}$, which by definition of transpose, indicates that $\Sigma = \Sigma^T$. Thus, we see that Σ is symmetric.

Also, we see that Σ_{ii} is full correlation, as i is equal to i. Thus, we have that $\Sigma_{ii} = 1$. Thus, the entries on the diagonal are 1s, and since correlation has a maximum magnitude of 1, each element in the off-diagonals have value $-1 \leq \Sigma_{ij} \leq 1$, or $|\ \Sigma_{ij}\ | \leq 1$.

In our position-velocity example, we have that Σ is 2×2. Suppose

$$\Sigma = \begin{pmatrix} 1 & a \\ b & 1 \end{pmatrix}$$

for $a, b \in \mathbb{R}, -1 \leq a, b \leq 1$. From this, we can see that the determinant is $1 - ab$. Thus, we can use this to find the eigenvalues and eigenvectors. If $\vec{x}$ is an eigenvector, $\Sigma\vec{x} = \lambda\vec{x} \implies (\Sigma - \lambda I)\vec{x} = 0$. Thus, the null space of $\Sigma - \lambda I$ is non-trivial, and thus

$$\det(\Sigma - \lambda I) = \det \begin{pmatrix} 1 - \lambda & a \\ b & 1 - \lambda \end{pmatrix} = 0.$$

We can then get that the characteristic polynomial is $\det(\Sigma - \lambda I) = (1-\lambda)^2 - ab = 0$, which gives $1 - ab - 2\lambda + \lambda^2 = 0$. Applying the quadratic formula, we get that $\lambda = 1 \pm \sqrt{ab}$.

We can then find the eigenvectors by finding the null space of

$$\begin{pmatrix} 1-\lambda & a \\ b & 1-\lambda \end{pmatrix} = \begin{pmatrix} 1-1\pm\sqrt{ab} & a \\ b & 1-1\pm\sqrt{ab} \end{pmatrix} = \begin{pmatrix} 1-\lambda & a \\ b & 1-\lambda \end{pmatrix} = \begin{pmatrix} \pm\sqrt{ab} & a \\ b & \pm\sqrt{ab} \end{pmatrix}$$

.

Through calculations, we get that our eigenvectors are $\left(\pm\sqrt{\frac{a}{b}}, 1\right)$

We also note that $1 \pm \sqrt{ab}$ such that $-1 \leq ab \leq 1$ gives that $1 \pm \sqrt{ab} \geq 0$. This means that we have that the λ values are non-negatives, which means that Σ is positive semi-definite by definition (as we remember that Σ is symmetric and has non-negative eigenvalues). This also means that if $a \neq 1 \wedge b \neq 1$, then we have that $\lambda > 0$, which means that Σ is then positive-definite. Generally, however, we note that Σ is positive semi-definite.

This is also related to the matrix P we defined above as they both relate to the correlation; P just describes the variance.

2.4 Predictions

Let $\vec{x}_k$ then be the most likely state of a system at time k. We can then predict $\vec{x}_{k+1}$. We don't need to know exactly what $\vec{x}_k$ is, but we can calculate a different error.

We can describe a transformation from $\vec{x}_k$ to $\vec{x}_{k+1}$ by linear transformation F_k such that there are different variances associated as well. After the prediction step, we expect that the variances increase if no new information comes to light.

Let's consider the simple case of no acceleration. We only have position and velocity to consider.

Using kinematics and physics, we derive equations $\vec{p}_{k+1} = \vec{p}_k + \Delta t \vec{v}_k$ where we get that $F_k = \begin{pmatrix} 1 & \Delta t \\ 0 & 1 \end{pmatrix}$. Thus, we get that $\vec{x}_{k+1} = F_k \vec{x}_k$.

Now, we move onto how the covariance matrix changes. We will prove this identity in the next section, but for now, we know that if $Cov(\vec{x}) = \Sigma$, then $Cov(A\vec{x}) = A\Sigma A^T$.

Thus, we have that since

$$Cov(\vec{x}_{k-1}) = \Sigma_{k-1},$$
$$\Sigma_k = Cov(F_k \vec{x}_{k-1}) = F_k \Sigma_{k-1} F_k^T.$$

2.5 More Complicated Predictions

Now that we have the basis for when there is no acceleration, let's add acceleration.

Using physics and kinematics, we have a similar-looking equation from last time:

$$\vec{p}_{k+1} = \vec{p}_k + \Delta t \vec{v}_k + \frac{1}{2} a \Delta t$$
$$\vec{v}_{k+1} = \vec{v}_k + a\Delta t$$

where a is the acceleration.

We can put all of this in terms of $\vec{x}$, which is our state vector. Note that $\vec{x} = \begin{pmatrix} p \\ v \end{pmatrix}$. From the last section, we have that without acceleration, $\vec{x}_{k+1} = F_k \vec{x}_k$. Adding acceleration gives

$$\vec{x}_{k+1} = \begin{pmatrix} 1 & \Delta t \\ 0 & 1 \end{pmatrix} \begin{pmatrix} p_k \\ v_k \end{pmatrix} + \begin{pmatrix} \frac{\Delta t^2}{2} \\ \Delta t \end{pmatrix} a = F_k \vec{x}_k + \begin{pmatrix} \frac{\Delta t^2}{2} \\ \Delta t \end{pmatrix} a$$

If we let $B_k = \begin{pmatrix} \frac{\Delta t^2}{2} \\ \Delta t \end{pmatrix}$ and $\vec{u}_k = a$, we get

$$\vec{x}_{k+1} = F_k \vec{x}_k + B_k \vec{u}_k$$

We define B as the control matrix and $\vec{u}$ as the control vector, where we decide that $\vec{u}$ as external influences of acceleration, while B is dependent on the time elapsed.

Here, we are calculating the expected best estimate at time $k+1$, which is an estimation based off of the previous best estimate at time k. We also have $\vec{u}_k$ as known influences that we can use to calculate the estimates.

Again, in this case, we assume that we know what sources of controlled forces there are. For instance, we know and can calculate the acceleration at take-off for a rocket. However, what about other forms of error?

2.6 Uncertainties

Now that we modelled what we can predict, we need to still account for uncertainty. For instance, how will sudden gusts of wind change the rocket's trajectory? What if aliens suddenly started pulling at the space ship?

We should add uncertainty to each prediction step. Every exact state (that we cannot know exactly) resides in a range of possibilities. We take all covariances of the noise (unknown external influences) and put them in an covariance matrix Q_k. Note that we keep the same mean state of the predict, just expand the covariances to cover other external influences.

Simply can just add this covariance matrix to our Covariance of $\vec{x}_k$ to get our new covariance:

$$Cov(\vec{x}_k) = \Sigma_k = F_k \Sigma_{k-1} F_k^T + Q_k$$
$$\vec{x}_{k+1} = \mathbb{F}_k \vec{x}_k + B_k \vec{u}_k$$

.

Thus, we now have that our uncertainty is derived from our previous uncertainty added to some uncertainty from the environment.

Although now we have estimates of both our prediction as well as the uncertainty, we now want to update and narrow our estimations given data.

2.7 Sensors

Our sensors give us readings that produces some information about a state. However, sensors might work a different way than how we modelled our state. Thus, we must create a new structure to keep track of sensor readings. We organize our sensor models with matrix H_k.

Given a state, predictions, and variances, a sensor can product an expected reading with variances too. Note that a sensor has imprecisions too, which give them variance. We model it by the sensor taking in a state and giving sensor outputs.

We let $\vec{z}_k$ be the actual sensor reading and R_k be the matrix of covariances that come inherently with the sensor.

Here, $\vec{\mu}$ stands for expected sensor reading. S stands for the matrix of covariances for sensor noise and general uncertainty in sensors.

$$\vec{\mu} = H_k \vec{x}_k$$
$$S = H_k \Sigma_k H_k^T$$

Again, we can derive S. The $Cov(x) = \Sigma \implies Cov(A\vec{x}) = A\Sigma A^T$. Thus, we get that since $Cov(x_k) = \Sigma_k$, we have that $S = Cov(H_k \vec{x}_k) = H_k \Sigma_k H_k^T$. I note again that the proof for this property is in section 3.

We notice that R_k, S both depend on the inherent sensor imprecision and *also* the variations of our predictions from the previous steps. We also see that $\vec{\mu}$ depends on the state that is *inputted* into the sensors to produce readings.

2.8 Change of Basis

In a sense, we can think of this as a change-of-basis. We are turning our current state in our "standard" coordinate system that we defined into a sensor basis. If we have 2 sensors that read position and velocity, we note that H_k maps from $\mathbb{R}^2 \to \mathbb{R}^2$ (if we are assuming the simple case that position and velocity are real numbers; if not, we can extrapolate).

If H_k is linearly independent – which if the sensors are different, they are – then there exists an inverse. Thus, we can think of $\vec{x}_k$, the state, as coordinates in H_k, the coordinate basis of our prediction state that may vary with the actual exact state. We then want to turn it into the standard basis, which produces $\vec{\mu}$ as it gives deterministic measurements. In other words, we can change the coordinate basis of $\vec{\mu}$ to $\vec{x}_k$ by applying H_k as we have learned about in class. In this case, $\vec{\mu}$ represents the states that our sensors *actually* outputted as well.

2.9 Reconciliation through Probabilities

Now, since we have $\vec{z}_k, R_k$ along with $\vec{x}_k, \Sigma_k$ which gives us $\vec{\mu}, S$, we need to update by reconciliation between the two different readings. We note that they both have errors associated with them with distributions on likelihood. Thus, our intuition tells us that there *should* be a most likely area that our *actual* state is in.

Focusing on $\vec{z}_k$, our sensor reading state, we now have 2 associated probabilities. We have the probability that $\vec{z}_k$ is an accurate reading of the sensors such that the sensor variations did not change too much. We also have the probability that our previous prediction believes that $\vec{z}_k$ is an accurate reading.

The probability of $\vec{z}_k$ accurate and $\vec{z}_k$ fits in with our prediction is the probability of each one multiplied together. Note that these two events are independent because knowing one does not affect the other; they are in different categories.

Note that when we multiply probabilities, we also change the Gaussian curves associated with them with a new mean and covariance. We can find this new mean and covariance given the multiplied means and covariances (which we have). We show the math behind it in section 3.

2.10 Kalman Gain

Through multiplication, we have our Kalman gain (which we can later derive the optimal Kalman gain) as

$$K = H_k \Sigma_k H_k^T (H_k \Sigma_k H_k^T + R_k)^{-1}$$

More intuitively, Kalman gains are "weights" that are added to update our mean and lower our uncertainty.

Now, let $(H_k \vec{x}_k, H_k \Sigma_k H_k^T)$ be our predicted measurements such that we follow (mean, variance) format. Let $(\vec{z}_k, R_k)$ be our actual measurements following the same format.

We then have that for an arbitrary mean $\vec{\mu}' = \mu_0 + K(\mu_1 - \mu_0)$ (also derived in section 3). The variances changes by $\Sigma' = \Sigma_0 - K\Sigma_0$.

Now, we can plug it in and compile what we have. Let $\vec{x}'_k$ be our updated best estimate and let Σ'_k be our covariance matrix along with it.

We get from what we derived in part 3:

$$H_k \vec{x}'_k = H_k \vec{x}_k + K(\vec{z}_k - H_k \vec{x}_k)$$
$$H_k \Sigma'_k H_k^T = H_k \Sigma_k H_k^T - K H_k \Sigma_k H_k^T$$

we also know

$$K = H_k \Sigma_k H_k^T (H_k \Sigma_k H_k^T + R_k)^{-1}$$

2.11 Linearity and Update Step

Now, we simplify.

We first note that H_k is linearly independent because we are assuming the sensors are different which give different specifications. Thus, this means that it is invertible (as it is square), which also means that it has full column rank and full row rank by rank-nullity theorem. Thus, we can use H_k^{-1}.

Because it has full column and full row rank, we have that H_k^T is also invertible. This means that $(H_k^T)^{-1}$ also exist – which we will be using below.

Also for the sake of simplicity, let's let $K' = \Sigma_k H_k^T (H_k \Sigma_k H_k^T + R_k)^{-1}$. We note that this is defined to be the *optimal* Kalman gain.

First, we look at the first equation:

$$
\begin{aligned}
H_k\vec{x}_k' &= H_k\vec{x}_k + K(\vec{z}_k - H_k\vec{x}_k) && \text{(given above)} \\
\implies H_k\vec{x}_k' &= H_k\vec{x}_k + H_k\Sigma_k H_k^T (H_k\Sigma_k H_k^T + R_k)^{-1}(\vec{z}_k - H_k\vec{x}_k) && \text{(substitution of K)} \\
\implies H_k\vec{x}_k' &= H_k(\vec{x}_k + \Sigma_k H_k^T (H_k\Sigma_k H_k^T + R_k)^{-1}(\vec{z}_k - H_k\vec{x}_k)) && \text{(linearity)} \\
\implies H_k^{-1}H_k\vec{x}_k' &= H_k^{-1}H_k(\vec{x}_k + \Sigma_k H_k^T (H_k\Sigma_k H_k^T + R_k)^{-1}(\vec{z}_k - H_k\vec{x}_k)) && (H_k \text{ invertible}) \\
\implies \vec{x}_k' &= \vec{x}_k + \Sigma_k H_k^T (H_k\Sigma_k H_k^T + R_k)^{-1}(\vec{z}_k - H_k\vec{x}_k) && \text{(inverse matrix)} \\
\implies \vec{x}_k' &= \vec{x}_k + K'(\vec{z}_k - H_k\vec{x}_k) && \text{(definition of } K')
\end{aligned}
$$

We have our result of $\vec{x}_k' = \vec{x}_k + K'(\vec{z}_k - H_k\vec{x}_k)$ as the update step for a state $\vec{x}_k$. Now, we must find the updated covariance matrix.

$$
\begin{aligned}
H_k\Sigma_k' H_k^T &= H_k\Sigma_k H_k^T - KH_k\Sigma_k H_k^T && \text{(given above)} \\
\implies H_k\Sigma_k' H_k^T &= H_k\Sigma_k H_k^T - H_k\Sigma_k H_k^T (H_k\Sigma_k H_k^T + R_k)^{-1} H_k\Sigma_k H_k^T && \text{(subst. of K)} \\
\implies H_k(\Sigma_k' H_k^T) &= H_k(\Sigma_k H_k^T - \Sigma_k H_k^T (H_k\Sigma_k H_k^T + R_k)^{-1} H_k\Sigma_k H_k^T) && \text{(linearity)} \\
\implies H_k^{-1}H_k(\Sigma_k' H_k^T) &= H_k^{-1}H_k(\Sigma_k H_k^T - \Sigma_k H_k^T (H_k\Sigma_k H_k^T + R_k)^{-1} H_k\Sigma_k H_k^T) && (H_k \text{ inv.}) \\
\implies \Sigma_k' H_k^T &= \Sigma_k H_k^T - \Sigma_k H_k^T (H_k\Sigma_k H_k^T + R_k)^{-1} H_k\Sigma_k H_k^T && \text{(inverse)} \\
\implies \Sigma_k' H_k^T &= \Sigma_k H_k^T - K' H_k\Sigma_k H_k^T && (K' \text{ definition}) \\
\implies (\Sigma_k' H_k^T)(H_k^T)^{-1} &= (\Sigma_k H_k^T - K' H_k\Sigma_k H_k^T)(H_k^T)^{-1} && (H_k^T \text{ inv.}) \\
\implies \Sigma_k' &= \Sigma_k H_k^T (H_k^T)^{-1} - K' H_k\Sigma_k H_k^T (H_k^T)^{-1} && ((H_k^T)^{-1} \text{ linearity}) \\
\implies \Sigma_k' &= \Sigma_k - K' H_k\Sigma_k && \text{(by inverse)}
\end{aligned}
$$

Thus, we have our result here for $\Sigma_k' = \Sigma_k - K' H_k\Sigma_k$.

Combining both of our results, we can get the equations for the update step in general.

$$\vec{x}'_k = \vec{x}_k + K'(\vec{z}_k - H_k\vec{x}_k)$$
$$\Sigma'_k = \Sigma_k - K'H_k\Sigma_k$$
$$K' = \Sigma_k H_k^T(H_k\Sigma_k H_k^T + R_k)^{-1}$$

2.12 Prediction and updates

Now that we have updated information $\vec{x}'_k$ and Σ'_k, we can launch into another round of update or predict. Iteratively doing this gives us predictions given errors and not too much information to work with.

3 Why Kalman Filters Work

3.1 Covariance and Linear Transformations

In the last part, we used that if $Cov(\vec{x}) = \Sigma$, then $Cov(A\vec{x}) = A\Sigma A^T$ without proof. I will prove it here.

By definition, we note that $Cov(\vec{x}) = \mathbb{E}[(\vec{x} - \mathbb{E}[\vec{x}])(x - \mathbb{E}[\vec{x}])^T]$

Then, we want to find an equivalent form of $Cov(A\vec{x})$.

$$\begin{aligned}
Cov(A\vec{x}) &= \mathbb{E}[(A\vec{x} - \mathbb{E}[A\vec{x}])(Ax - \mathbb{E}[A\vec{x}])^T] && \text{(by definition of Cov)}\\
&= \mathbb{E}[(A\vec{x} - A\mathbb{E}[\vec{x}])(Ax - A\mathbb{E}[\vec{x}])^T] \\
& && \text{(by linearity and expected val definition)}\\
&= \mathbb{E}[(A(\vec{x} - \mathbb{E}[\vec{x}]))(A(x - \mathbb{E}[\vec{x}]))^T] && \text{(by linearity)}\\
&= \mathbb{E}[(A(\vec{x} - \mathbb{E}[\vec{x}]))((x - \mathbb{E}[\vec{x}]))^T A^T] && \text{(by linearity and transpose)}\\
&= A\mathbb{E}[(\vec{x} - \mathbb{E}[\vec{x}]))(x - \mathbb{E}[\vec{x}])^T]A^T && \text{(by linearity and expected value)}\\
&= ACov(\vec{x})A^T && \text{(by definition)}
\end{aligned}$$

Thus, we have shown what we want.

3.2 Gaussian Curve Multiplication (Probability and Algebra)

We can represent mean and covariance in a bell-curve form such that by definition,

$$\mathcal{N}(x, \vec{\mu}, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\vec{\mu})^2}{2\sigma^2}}$$

Thus, we have $\mathcal{N}(x, \mu_0, \sigma_0) \cdot \mathcal{N}(x, \mu_1, \sigma_1) = \mathcal{N}(x, \vec{\mu}', \sigma')$ where we want to find $\vec{\mu}', \sigma'$

We can use the equations and do careful algebra to obtain formulas for $\vec{\mu}'$ and σ'.

$$\vec{\mu}' = \mu_0 + \frac{\sigma_0^2(\mu_1 - \mu_0)}{\sigma_0^2 + \sigma_1^2}$$

$$\sigma'^2 = \sigma_0^2 - \frac{\sigma_0^4}{\sigma_0^2 + \sigma_1^2}$$

We note that we can factor out $k = \frac{\sigma_0^2}{\sigma_0^2+\sigma_1^2}$ to get that

$$\vec{\mu}' = \mu_0 + k(\mu_1 + \mu_0)$$
$$\sigma'^2 = \sigma_0^2 - k\sigma_0^2$$

We can note that $0 \leq k \leq 1$ because square are non-negative and division gives less than or equal to 1.

We can put this in matrix form by generalizing in higher dimensions to get the form that we used in part 2. We do this by adding variables and pair-wise covariances, which creates these equations by Σ being the covariance matrix:

$$\vec{\mu}' = \mu_0 + K(\mu_1 - \mu_0)$$
$$\Sigma' = \Sigma_0 - K\Sigma_0$$
$$K = \Sigma_0(\Sigma_0 + \Sigma_1)^{-1}$$

Because we understand what $\vec{\mu}, \Sigma$ are in this context, we can rewrite it into the form that is most useful in trying to determine our updated states.

$$H_k\vec{x}'_k = H_k\vec{x}_k + K(\vec{z}_k - H_k\vec{x}_k)$$
$$H_k\Sigma'_kH_k^T = H_k\Sigma_kH_k^T - KH_k\Sigma_kH_k^T$$
$$K = H_k\Sigma_kH_k^T(H_k\Sigma_kH_k^T + R_k)^{-1}$$

This eventually leads us to complete our updates, as shown in Section 2.

4 Our Code Explained

We can first look at the rocket position-velocity example to explain the pseudo-code for a Kalman filter. Let Δt be the timestep we are calculating over. We have the

following variables:

State vector:	$\vec{x} = \begin{pmatrix} p \\ v \end{pmatrix}$
Mean state covariance:	$P = \begin{pmatrix} \sigma_p^2 & \sigma_{pv} \\ \sigma_{pv} & \sigma_v^2 \end{pmatrix}$
Transition matrix:	$A = \begin{pmatrix} 1 & \Delta t \\ 0 & 1 \end{pmatrix}$
Mean state estimate (expected sensor reading):	$\vec{\mu} = \begin{pmatrix} \mu_p \\ \mu_v \end{pmatrix} = H_k \vec{x}_k$
Noise (inaccuracy:	Q

The Kalman filter in practice performs a *prediction step* and an *update step*. Let $\vec{x}(k)$ donate the state at time t, $vec\vec{\mu}k)$ donate the mean state estimate at time t, and $P(k)$ denote the mean state covariance at time t. It performs its prediction by multiplying the transition matrix by the mean state estimate vector, much like a Markov chain. It further updates covariance based on the transition matrix and noise , i.e. $P(k+1) = AP(k)A^T + Q$. In pseudocode, we have

Algorithm 1 predict$(\vec{\mu}, P, A, Q)$:

1: $\vec{\mu} \leftarrow A\vec{\mu}$
2: $P \leftarrow APA^T + Q$
3: **return** $(\vec{\mu}, P)$

Now at some timestep k, the filter must compute the mean state estimate $\vec{\mu}$ and the covariance P given a new measurement, or state vector, $\vec{x}$. Here, we have to introduce some more variables:

Measurement matrix:	H_k
Input measurement covariance:	$R_k = H_k P H_k^T$
Covariance of predictive mean (innovation):	$S = R_k + H_k P H_k^T$
Kalman Gain:	$K = PH^T S^{-1}$

The new state $\vec{\mu}(k+1)$ is computed via

$$\vec{\mu}(k+1) = \vec{\mu}(k) + K(\vec{x} - H\vec{\mu}(k))$$

where $\vec{x}$ is the true measurement. Then the updated mean state covariance matrix $P(k+1)$ is

$$P(k+1) = P(k) - KHP(k).$$

In pseudocode:

We can now run this algorithm using random position and velocity vectors, under the assumption that they are uncorrelated. Let $\Delta t = 0.1$, and

$$H = \begin{pmatrix} 1.0 & 0.0 \\ 0.0 & 1.0 \end{pmatrix}, P = \begin{pmatrix} 5.0 & 0.0 \\ 0.0 & 5.0 \end{pmatrix}, A = \begin{pmatrix} 1 & \Delta t \\ 0 & 1 \end{pmatrix}, Q = \begin{pmatrix} 0.1 & 0.0 \\ 0.0 & 0.3 \end{pmatrix}$$

Algorithm 2 update$(\vec{\mu}, P, \vec{x}, H, R)$:

1: $S \leftarrow R + HPH^T$
2: $K \leftarrow PH^T S^{-1}$
3: $\vec{\mu} \leftarrow \vec{\mu} + K(\vec{x} - H\vec{\mu})$
4: $P \leftarrow P - KHP$
5: **return** $(\vec{\mu}, P)$

We generate our position and velocity measurements based on a randomly generated offset from our previous measurements using the Julia `rand()` function. While random measurements are not realistic for a rocket, they are representative of measurements a Kalman filter may encounter and illustrate its predictive capabilities. Our mean state covariance matrix P is defined as such as σ_v and σ_p have high correlation with themselves, and since they are random, have no correlation with each other. Transition matrix A is defined as the typical position-velocity relationship. The measurement matrix is the I_2 as we are measuring distance (meters) and velocity (meters / second). In other situations, for example, measuring pixels per second, H would not be the identity as we need to translate into meters / second.

Running our Kalman filter over 500 timesteps, we produce the following plots:

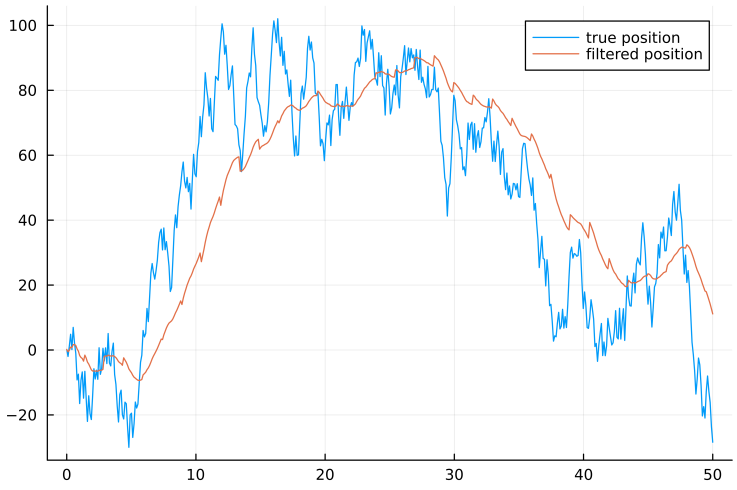

Figure 1: True and estimated position over time

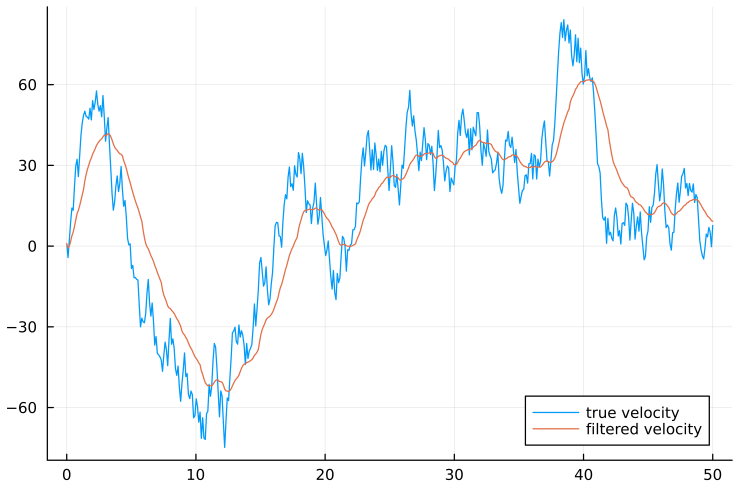

Figure 2: True and estimated velocity over time

Only two images are displayed here, but running this code across any of the on-demand generated random position and velocity vectors yields expected results.

5 Written and Commented Code

The code can be viewed and run at this GitHub repository.

```
using LinearAlgebra
using Plots

# prediction step
function predict(mu, P, A, Q)
    # update the mean state estimate based on the transition matrix: X = AX
    mu = *(A, mu)

    # update covariance based on the transition matrix: P = APA^T
    P = *(A, *(P, transpose(A))) + Q

    # return mean state, mean covariance
    return (mu, P)
end

# update step
function update(mu, P, X, H, R)
    # update the covariance of the predictive mean
    S = R + *(H, *(P, transpose(H)))

    # calculate Kalman gain
```

```
    K = *(*(P, transpose(H)), inv(S))

    # update the new expected sensor reading
    mu = mu + K * (X - *(H, mu))

    # update mean state covariant matrix
    P = P - *(K, *(H, P))

    # return mean state, mean covariance
    return (mu, P)
end

function run_kalman()

    initializing variables
    dt = 0.1 # time step
    x = [rand(); rand()] # original state
    H = [1.0 0.0; 0.0 1.0] # measurement matrix
    P = [5.0 0; 0 5.0] # mean state covariance
    A = [1 dt; 0 1] # transition matrix
    Q = [0.1 0; 0 0.3] # noise

    R = *(H, *(P, transpose(H)))
    mu = *(H, x) # mean state vector

    x_arr = fill(0.0, 500) # true pos measurements for plotting
    y_arr = fill(0.0, 500) # true vel measurements for plotting


    pos_arr = fill(0.0, 500) # inits mu arr for filtered
    vel_arr = fill(0.0, 500) # inits vel arr for filitered

    time_arr = range(0, 50, 500)

    # loop over time to run Kalman filter
    for i in 1:500

        # predicts sparsely
        if (i % 15 == 0)
            (mu, P) = predict(mu, P, A, Q) # predicts
        end

        # updates with every new measurement
        (mu, P) = update(mu, P, x, H, R) # updates

        x_arr[i] = x[1] # fill in true position for comparison
```

```
        y_arr[i] = x[2]  # fill in true vel for comparison
        pos_arr[i] = mu[1] # after predict/update, fill in pos_arr
        ↪ for plotting
        vel_arr[i] = mu[2] # after predict/update, fill in vel_arr
        ↪ for plotting

        x = [x[1] + 20*(rand() - 0.5); x[2] + 20*(rand() - 0.5)] #
        ↪ gets next random datapoint

    end

    # generate and save plots
    plot(time_arr, [x_arr, pos_arr], label=["true position" "filtered
    ↪ position"], dpi=1000)
    savefig("pos_plot.png")
    plot(time_arr, [y_arr, vel_arr], label=["true velocity" "filtered
    ↪ velocity"], dpi= 1000)
    savefig("vel_plot.png")
    return
end

run_kalman()
```

6 Bibliography

1. https://arxiv.org/pdf/1204.0375.pdf
2. https://en.wikipedia.org/wiki/Kalman_filter
3. https://en.wikipedia.org/wiki/Covariance_matrix
4. https://thekalmanfilter.com/kalman-filter-python-example/
5. https://www.kalmanfilter.net/...
6. https://thekalmanfilter.com/kalman-filter-explained-simply/
7. https://www.bzarg.com/p/how-a-kalman-filter-works-in-pictures/
8. https://www.cs.cmu.edu/...
9. https://docs.julialang.org/en/v1/