ECSE 4961 Project 4

Aidan Westphal

December 2023

Purpose

The purpose of this project was to implement the Kalman Filtering algorithm to track features over time in a video of a Rubik's cube.

Theory

Kalman Filtering is a probabilistic method of feature tracking that combines two probability distributions: a prediction based on previous locations of the object and a measurement with error bounds, to determine the object's location. It is an iterative process in which, as it progresses, the variance of its prediction shrinks.

Kalman Filtering makes the following assumptions: States change linearly and uncertainty follows a Gaussian distribution. We define a tracking point as $p = (c_t, r_t)$ at time t (or frame at time t) and its velocity $v_p = (v_{ct}, v_{rt})$. We then define a state and the following state as follows:

$$s_t = \begin{bmatrix} c_t \\ r_t \\ v_{ct} \\ v_{rt} \end{bmatrix} \text{ and } s_{t+1} = \begin{bmatrix} c_t + v_{ct} \\ r_t + v_{rt} \\ v_{ct} \\ v_{rt} \end{bmatrix}$$

Note how we use the assumption that states transition linearly. We can better encode this transition by using a state transition matrix, where $s_t = \Phi s_{t-1} + w_t$. Note w_t represents system pertubation (error source) and is normally distributed as $w_t \sim N(0, Q)$ where Q is a covariance matrix. This leads to the following probability distribution: $p(s_t|s_{t-1}) = N(\Phi s_{t-1}, Q)$.

$$\Phi = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

We now develop a measurement model, where $z_t = (z_{ct}, z_{rt})$ is a measured target position at the time (frame) t. We note that $z_t = Hs_t + \epsilon_t$, which the matrix H pulls the row and column values from s_t and defines error ϵ_t between the predicted state and the measured state. We let $\epsilon_t \sim N(0, R)$

where R is measurement uncertainty and can define our measurement model as: $p(z_t|s_t) = N(Hs_t, R)$.

$$H = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

So to wrap up terminology, s_t is a state which contains the row and column of a tracking point along with its velocity. z_t is a measurement at time instance t. We can predict the next state $s_{t+1} = \Phi s_t + w_{t-1}$ which resembles the following distribution: $p(s_t|s_{t-1}) = N(\Phi s_{t-1}, Q)$. Similarly, we can take a measurement value and note $z_t = Hs_t + \epsilon_t$ where we describe the error between our measurement value and our actual state s_t, which resembles the following distribution: $p(z_t|s_t) = N(Hs_t, R)$. Thus, our end goal is to find the distribution of our new state s_t given all previous (and current) measurements, i.e. $p(s_t|z_{1:t})$.

Since Kalman Filtering is an iterative process, also noting that we never precisely know a value s_t due to uncertainty, we are given $p(s_{t-1}|z_{1:t-1}) = N(U_{t-1}, \Sigma_{t-1})$, the distribution of the previous state given all previous measurements. We can split $p(s_t|z_{1:t})$ as follows:

$$p(s_t|z_{1:t}) \propto p(s_t|z_{1:t-1})p(z_t|s_t)$$

Where we have a term for prediction based on previous measurements and a term for measurement given the current state. We start with $p(s_t|z_{1:t-1}) = N(U_t^-, \Sigma_t^-)$, which can be found to equal the following after some derivation (involves integrating the product of probability distributions):

$$U_t^- = \Phi U_{t-1} \; \Sigma_t^- = \Phi \Sigma_{t-1} \Phi^T + Q$$

For notation, this is known as the "Temporal Prior of s_t." We then take our measurement distribution term, $p(z_t|s_t)$, known as the "Likelihood of s_t". We have already calculated this above, as $N(Hs_t, R)$. We now note the proportion defined earlier and note $p(s_t|z_{1:t}) = N(U_t^-, \Sigma_t^-)N(Hs_t, R)$. After more derivation...

$$U_t = U_t^- + K_t(z_t - HU_t^-) \; \Sigma_t = (I - K_t H)\Sigma_t^-$$

where $K_t = \arg\min trace(\Sigma_t) = H^T(H\Sigma_t^- H^T + R)^{-1}$ represents a weight/gain matrix that measures the relative contribution between the prediction and

the measurement components and is obtained by minimizing the uncertainty of the final prediction.

Now that we understand how the filtering process works, we need a means of measurement, in our case finding the matching feature point in the next image. For this project, I used the Sum of Squares Differences (SSD) method. How this works is we have a baseline region (a nxn region surrounding a given pixel (feature) on the first image), called W_1, and we search for the nxn region W_2 in the other image which best minimizes the sum of the square differences of each entry in either matrix. In other words, we want to find W_2 given W_1 which minimizes:

$$\sum_{i}^{n}\sum_{j}^{n}([W_2]_{ij} - [W_1]_{ij})^2$$

Given this definition, we can better describe the measurement portion of Kalman Filtering. In the first image frame, we define W_1 as the nxn neighborhood of pixels surrounding a feature (the pixel in the center). In the t_{th} image we search from the pixel U_t^-, the mean of the predicted distribution, and search in a neighborhood defined as roughly $4\sigma, 4\sigma$ region. In other words, we perform SSD on pixels within two standard deviations of our predicted mean and take the coordinate that minimizes the above expression a z_t. A major issue with SSD however is its lack of invariances, making it an unstable method of feature detection. The following are a list of some primary issues with SSD which caused problems with detection for this project:

1. Small window (W_1 and W_2) size leads to many windows appearing the same, such as in large regions with similar color (white tiles on cube) or thick edges (corners vs edges of cube).

2. Rotation variance implies that, as a feature rotates, W_2 no longer aligns with W_1.

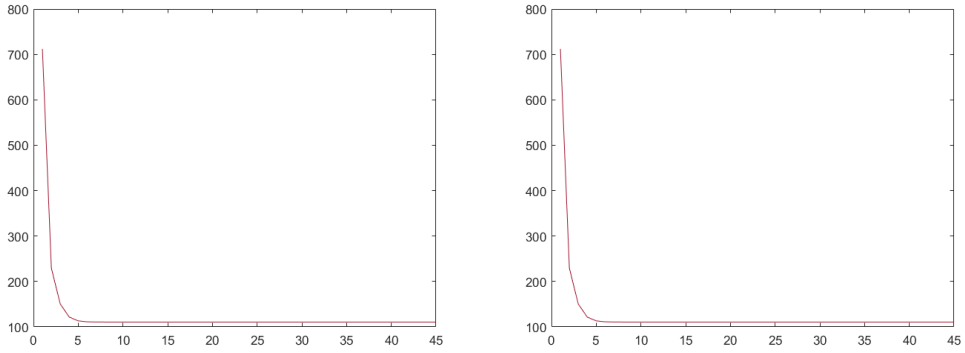
3. Scaling variance means that as a feature moves away from the camera (cube corner rotates away), W_2 changes in ways that W_1 doesn't.

While SSD is primitive and has many issues, the probabilistic nature of Kalman Filtering allows us to add some rigor to the algorithm by significantly reducing a feature's possible location.

The last thing to describe is initialization. I defined the initial state s_0 by hand-picking features and estimating v_{ct}, v_{rt} by matching these pixels to s_1 and taking the difference in row and column values. We also need to initialize Σ_0, which should be set to large numbers. Note that these values will adapt/strengthen/converge as the algorithm progresses. Q and R describe errors for the system and measurement models, which stay the same over the algorithm. Since I used SSD for feature detection, I set R to higher variances.

Results

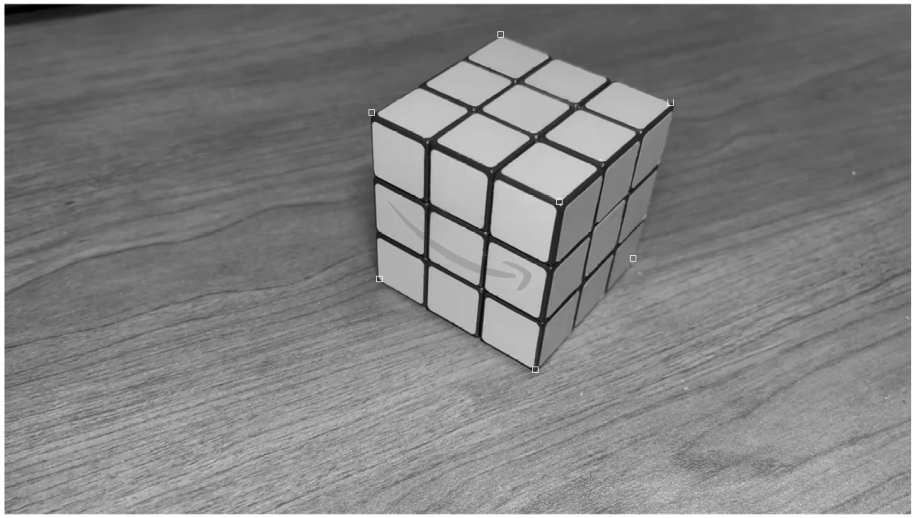
I ran the Kalman Filtering process (in MATLAB) on two sets of features: the corners of the cube and in between individual tiles on the face of the cube. The following is the trace of the covariance matrix over time for both sets of features:



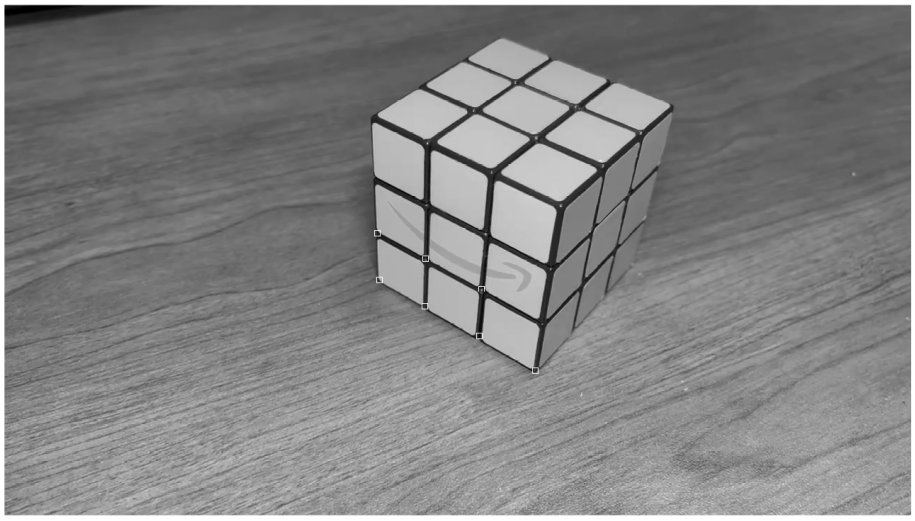
(a) Cube (b) Grid

Figure 1: Trace of Covariance Matrix over Time

Videos of each tracking process can be found attached to the project alongside the images below if they are hard to view. The following are some important snippets:

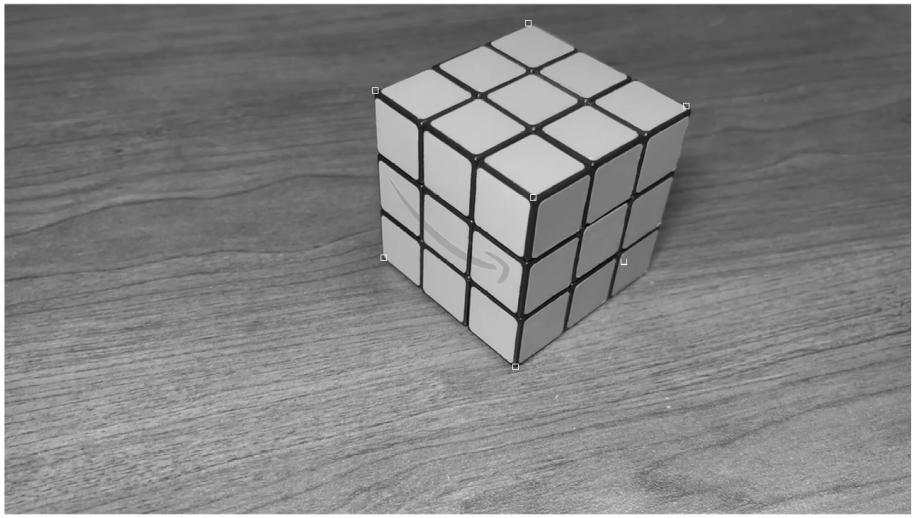


(a) Cube

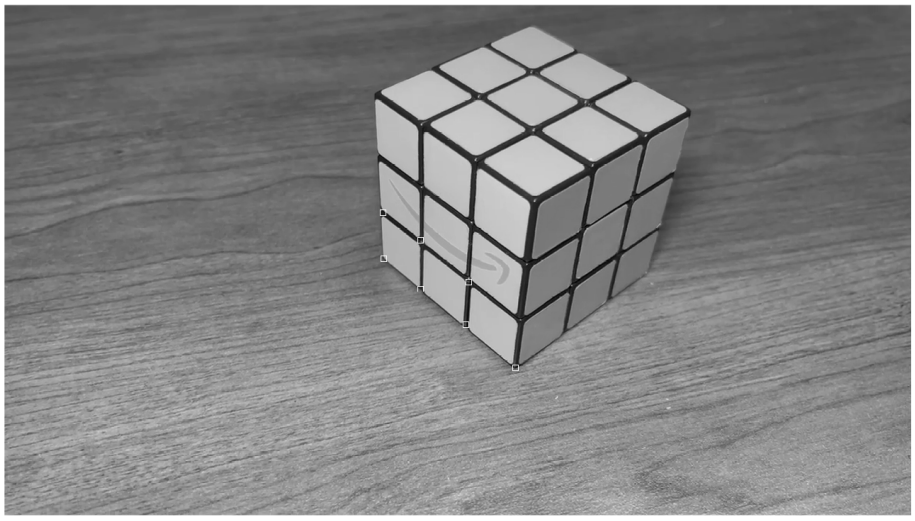


(b) Grid

Figure 2: t = 2

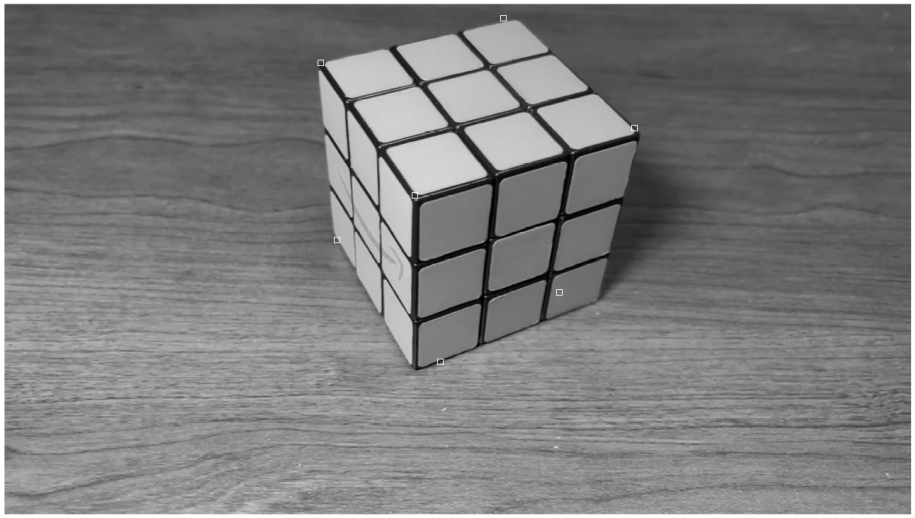


(a) Cube

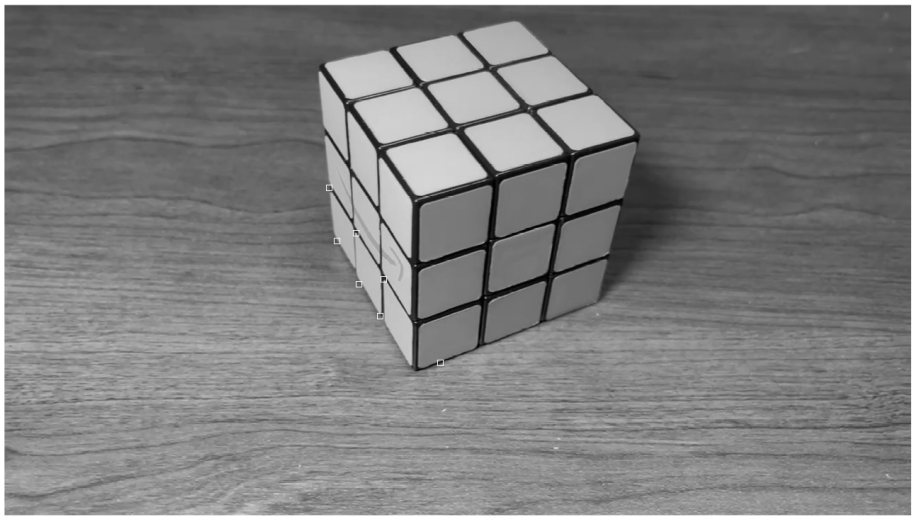


(b) Grid

Figure 3: t = 25



(a) Cube



(b) Grid

Figure 4: t = 45

For the above, we have the following covariance matrices in order up to time t = 11, where the covariance matrix converges to 4 decimal places. Note

that the covariance matrix is not impacted by the choice of features and thus is the same for the grid and cube points.

$$t=1:\begin{bmatrix}256&0&0&0\\0&256&0&0\\0&0&100&0\\0&0&0&100\end{bmatrix}\quad t=2:\begin{bmatrix}23.4606&0&6.1576&0\\0&23.4606&0&6.1576\\6.1576&0&91.3695&0\\0&6.1576&0&91.3695\end{bmatrix}$$

$$t=3:\begin{bmatrix}21.4718&0&13.7637&0\\0&21.4718&0&13.7637\\13.7637&0&53.6761&0\\0&13.7637&0&53.6761\end{bmatrix}\quad t=4:\begin{bmatrix}20.9063&0&11.0430&0\\0&20.9063&0&11.0430\\11.0430&0&39.8865&0\\0&11.0430&0&39.8865\end{bmatrix}$$

$$t=5:\begin{bmatrix}20.2965&0&9.5819&0\\0&20.2965&0&9.5819\\9.5819&0&36.3664&0\\0&9.5819&0&36.3664\end{bmatrix}\quad t=6:\begin{bmatrix}20.0329&0&9.1293&0\\0&20.0329&0&9.1293\\9.1293&0&35.5874&0\\0&9.1293&0&35.5874\end{bmatrix}$$

$$t=7:\begin{bmatrix}19.9547&0&9.0243&0\\0&19.9547&0&9.0243\\9.0243&0&35.4460&0\\0&9.0243&0&35.4460\end{bmatrix}\quad t=8:\begin{bmatrix}19.9372&0&9.0058&0\\0&19.9372&0&9.0058\\9.0058&0&35.4264&0\\0&9.0058&0&35.4264\end{bmatrix}$$

$$t=9:\begin{bmatrix}19.9341&0&9.0035&0\\0&19.9341&0&9.0035\\9.0035&0&35.4247&0\\0&9.0035&0&35.4247\end{bmatrix}\quad t=10:\begin{bmatrix}19.9338&0&9.0033&0\\0&19.9338&0&9.0033\\9.0033&0&35.4246&0\\0&9.0033&0&35.4246\end{bmatrix}$$

$$t=11:\begin{bmatrix}19.9337&0&9.0033&0\\0&19.9337&0&9.0033\\9.0033&0&35.4246&0\\0&9.0033&0&35.4246\end{bmatrix}$$

For the sake of space on this PDF, I will not show every image and matrix. The images can be found in either video

Conclusion

In the end, the algorithm seemed to be working but it is hard to quantify if error is derived from a faulty algorithm, measurement, or a weak algorithm. This explains the first main problem I had with this project. The algorithm was performing significantly worse than expected. I eventually determined

this was the result of a few factors, primarily relating to using SSD for feature detection. The first was that the window size was too small. Normally, a window size of 5-7 is chosen but, especially when working with a Rubik's cube, you have thick black edges and corners which are of said thickness. As such, an SSD algorithm would not be able to differentiate between a 5x5 neighborhood spot on a corner or somewhere on an edge. My solution was to increase the region to 11x11, implying that the window size would bleed into the surrounding areas rather than just the black corner. This saw major improvements but the algorithm still performed subparly. I improved this by increasing the variance (R) for measurement. This led to the results presented.

The algorithm holds on strongly, but still breaks under some expected conditions. For example, as the cube rotates, notable features on the grid plane, such as the thick black lines, thin out to almost nothing, making an algorithm like SSD which compares back to the first timeframe, highly unstable. Other notable error sources came from my hand-picking of features, specifically the bottom right corner in the cube set of features. My eyes had trouble differentiating between the corner and the edge and, as such, I picked a subpar point that quickly fell into a white tile, getting lost.

Other than this, the algorithm held on fairly strong, but once again I believe further tweaking with variances and initial values would improve results. There may also be inaccuracy in the algorithm code itself but I doubt it. It's also worth noting that the trace of the covariance matrix leveled out at a nonzero value. This makes sense due to the addition of Q and R. We will never be exactly certain given the internal error of our system and measurements.

For my final project, I will implement Particle Filtering, which will be interesting to compare with this project. I will keep SSD feature detection for a fair playing ground despite my suspicion that it's the main cause for error. In the end, this project was enjoyable and rewarding for me. I learned a lot.