PRINCIPLES OF RADAR TRACKING

Luke Anderson, Ankur Bakshi, Kareem Elnahal, Joe Kelly, David Kim, Vikram Modi, Adam Pantel, and Joe Park, Alex Schnayder, Alex Sood, Andrew Weintraub

Advisor: Randy Heuer
Assistant: Karl Strohmaier

ABSTRACT

Radar alone gives noisy estimates of a target's location and is incapable of directly measuring velocity. To rectify these shortcomings, we researched linear estimators, a class of algorithms that more accurately estimate position and velocity. We chose the Kalman filter because of its simplicity, efficiency, and low memory requirements. We developed a Visual Basic .NET console application that returned the target's positions and velocities. The program typically provided position estimates within a radius of half a mile of the true positions and returned velocity estimates within a range of three miles per hour.

INTRODUCTION

Process

When most people think of radar, they think of its depiction in movies as a foolproof way of immediately finding a target's exact location. However, it is actually much more complex than simply bouncing a wave off a target and measuring its return. Due to various types of "noise", or errors, equations and algorithms must be created to make the raw measurements from the radar system give a more accurate estimate of the target's position.

Our project involved creating and implementing an algorithm known as the Kalman filter. We used the filter in five different scenarios involving either a single or dual radar system that measured the target's position. We used the filter to help refine the measurements and bring them closer to the target's actual position by accounting for both driving noise, the variation in the flight path of the target; and measurement noise, the accuracy of the radar itself. The scenarios increased in difficulty each time, moving from simple one-dimensional range coordinates to Cartesian coordinates and then to polar coordinates. Polar coordinates include range and bearing from a set normal line pointing north and are closer to what a true radar system would measure.

After filtering the measurements and obtaining our calculated data, we analyzed it on a comparative basis with the true position and velocity values that were given to us. To check the accuracy of the filter, we used a residual graph—a graph that shows the differences between actual and predicted points at each point in time. When doing this, we looked to see that the filter's predicted values increased in accuracy as time progressed, that the filter did not contain a bias in any one direction, and that the filtered data were consistently more accurate than the raw measured values.

Background

Rudolf Kalman, an electrical engineer by training, is most famous for his co-invention of the filter that now bears his name, the Kalman filter. The Kalman filter is a digital mathematical signal processing technique which uses recursion to estimate the state of a dynamic system from a series of incomplete and noisy measurements. The roots of this equation can be traced back to Carl Friedrich Gauss's 1795 work.

Kalman was born in Budapest, Hungary, on May 19, 1930. He obtained his bachelor's and master's degrees from MIT in 1953 and 1954, respectively, and his doctorate from Columbia in 1957 [1]. Kalman's ideas for the filter were first met with so much resistance that he had to publish the results in a mechanical journal rather than an electrical one. However, after Kalman visited Stanley Schmidt at the NASA Ames Research Center in 1967, his filter was used in trajectory estimation for the Apollo program's navigation system [2].

Since then, the Kalman filter has gained a wide variety of uses in a diverse range of fields. Some of the areas in which it is used include nuclear power plant instrumentation, demographic modeling, manufacturing, detection of underground radioactivity, fuzzy logic (a branch of logic in which truth is not absolute), neural network training, and econometrics [3]. The number of applications has increased rapidly in recent years with the advent of new computer technologies, and it has now entered into the development of sophisticated weapon delivery systems, satellite surveillance systems, and non-military tracking systems such as Air Traffic Control [4]. It is also being used today in three-dimensional environment technologies to track the movement of targets.

Scenario Overviews

- Case 1: One Dimensional Tracking
- Case 2: Two Dimensional Tracking
- Case 3: Polar Coordinate Tracking
- Case 4: Dual Radar Tracking
- Case 5: Maneuvering Target Tracking

We were given five different cases in which to use the Kalman filter. Each had an increasing level of difficulty and complexity. In the first case, we had to track a target moving only in one dimension. For the second case, the target was moving in two dimensions, and the measured data was given to us in Cartesian coordinates. In the third case, we were again told to track a target moving in two dimensions, but the measured data was given to us in polar coordinates, which corresponds more closely to the data given by real radars. The fourth case introduced the problem of having multiple radars tracking a single target. In the final case, we had to track a maneuvering target that switched velocity twice during its course.

KALMAN FILTER EQUATIONS

True Position and Radar Input

The true position of the object at time $k+1$, given the position at time k is:

$$\mathbf{x}(k+1) = \Phi\mathbf{x}(k) + \mathbf{q}(k) \tag{1}$$

The state vector $\mathbf{x}(k)$ is:

$$\mathbf{x}(k) = \begin{bmatrix} s_1 \\ \vdots \\ s_n \\ \hline v_1 \\ \vdots \\ v_n \end{bmatrix}$$

and the state transition model Φ is:

$$\mathbf{\Phi} = \left[\begin{array}{c|c} \mathbf{I}_n & \Delta t \cdot \mathbf{I}_n \\ \hline 0 & \mathbf{I}_n \end{array}\right]$$

$\mathbf{I_n}$ represents the $n \times n$ identity matrix.

The state vector keeps track of the target's positions and velocities in different dimensions (usually the *x* and *y* dimensions). The purpose of the Kalman filter is to estimate the true state vector given a series of discrete radar measurements.

The state transition model updates the state vector each timestep. The state transition model updates each position by adding the time interval between each radar measurement multiplied by the velocity in the same dimension.

Because of mechanical and pilot error, however, it is impossible for a flying object to maintain a constant velocity. This is called driving noise and is represented by **q**(*k*). It is added to the state vector of each timestep to account for such driving irregularities. Mathematically, this noise, while zero on average, is a random Gaussian noise process with known covariance matrix:

$$\mathbf{Q} = \left[\begin{array}{ccc|ccc} \mathrm{var}(q_{s_1}) & \cdots & \mathrm{cov}(q_{s_1}, q_{s_n}) & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ \mathrm{cov}(q_{s_n}, q_{s_1}) & \cdots & \mathrm{var}(q_{s_n}) & 0 & \cdots & 0 \\ \hline 0 & \cdots & 0 & \mathrm{var}(q_{v_1}) & \cdots & \mathrm{cov}(q_{v_1}, q_{v_n}) \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & \mathrm{cov}(q_{v_n}, q_{v_1}) & \cdots & \mathrm{var}(q_{v_n}) \end{array}\right]$$

In general, the covariance of a vector of random variables is defined as:

$$\mathrm{cov}(\mathbf{X}) \equiv E\left((\mathbf{X} - E(\mathbf{X}))(\mathbf{X} - E(\mathbf{X}))^{\mathrm{T}}\right). \tag{2}$$

Driving noise between positions and velocities is uncorrelated, which explains the zeros in the bottom left and top right quadrants of $\mathbf{Q}$.

Because radar can only measure position and not velocity, the state vector must be converted into a measurement vector by the following equation. The measurement vector is a function of the state vector plus a random noise process:

$$\mathbf{y}(k) = \mathbf{H}\mathbf{x}(k) + \mathbf{r}(k), \tag{3}$$

where the measurement vector $\mathbf{y}(k)$ is:

$$\mathbf{y}(k) = \begin{bmatrix} s_1 \\ \vdots \\ s_n \end{bmatrix},$$

and the observation model $\mathbf{H}$ is:

$$\mathbf{H} = \left[\mathbf{I}_n \mid 0\right].$$

In order to convert the state vector into a measurement vector, all the velocities must be eliminated since they cannot be measured. This is accomplished by multiplication with the observation model, which removes every velocity by effectively cutting the state vector in half.

Just as driving noise was added to the state vector, measurement noise must be added to the measurement vector. Intuitively, this noise represents the inability of the radar tracking device to precisely measure the object's position. This could be due to several technical problems, from limitations in the radar screen's resolution to vibrations in the equipment. Mathematically, measurement noise represents the standard deviation σ between the positions that should be measured and the positions that are actually measured. Therefore, $\mathbf{r}(k)$ is a Gaussian random process that follows a multivariate normal distribution with covariance matrix:

$$\mathbf{R} = \begin{bmatrix} \sigma_{s_1}^2 & \cdots & \sigma_{s_1}\sigma_{s_n} \\ \vdots & \ddots & \vdots \\ \sigma_{s_n}\sigma_{s_1} & \cdots & \sigma_{s_n}^2 \end{bmatrix}.$$

Prediction

The first phase of each iteration of the Kalman filter is the prediction stage, in which the algorithm gives both predictions of the object's state vector and an estimate of how reliable the prediction is.

Predictions of the object's state vector are given using the following equation:

$$\hat{\mathbf{x}}(k \mid k-1) = \Phi\hat{\mathbf{x}}(k \mid k), \tag{4}$$

which is simply the prediction analog of Eq. (1). (The notation $\hat{\mathbf{x}}(m \mid n)$ means the prediction of vector $\mathbf{x}$ at time m made at time n.) The Kalman filter must estimate both position and velocity even though radar can only track position.

By Eq. (2), the estimate of the predicted state vector's reliability is given by $\mathbf{P} \equiv E\left(\left(\mathbf{x}-\hat{\mathbf{x}}\right)\left(\mathbf{x}-\hat{\mathbf{x}}\right)^{\mathrm{T}}\right)$, which is the covariance of the difference between predicted and actual states. This difference should be zero on average, but $\mathbf{P}$ itself will never be less than $\mathbf{Q}$. Covariance is a measure of the degree to which numbers vary. In other words, applied to the state covariance matrix, covariance measures how spread out the errors are. Expanding this definition of the state covariance matrix gives:

$$\mathbf{P}=\left[\begin{array}{ccc|ccc} \mathrm{var}\left(\varepsilon_{s_1}\right) & \cdots & \mathrm{cov}\left(\varepsilon_{s_1},\varepsilon_{s_n}\right) & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ \mathrm{cov}\left(\varepsilon_{s_n},\varepsilon_{s_1}\right) & \cdots & \mathrm{var}\left(\varepsilon_{s_n}\right) & 0 & \cdots & 0 \\ \hline 0 & \cdots & 0 & \mathrm{var}\left(\varepsilon_{v_1}\right) & \cdots & \mathrm{cov}\left(\varepsilon_{v_1},\varepsilon_{v_n}\right) \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & \mathrm{cov}\left(\varepsilon_{v_n},\varepsilon_{v_1}\right) & \cdots & \mathrm{var}\left(\varepsilon_{v_n}\right) \end{array}\right],$$

where $\varepsilon \equiv \mathbf{x}-\hat{\mathbf{x}}$. The terms along diagonals in the upper left and bottom right quadrants denote the variances of errors in positions $s_1,\ldots,s_n$ and velocities $v_1,\ldots,v_n$. These numbers have a practical application in that they give the formula of an ellipse in which the errors have a certain probability of lying. In two dimensions, the equation of an ellipse that has not been rotated is:

$$\frac{x^2}{a^2}+\frac{y^2}{b^2}=1. \tag{5}$$

Rotating the coordinate plane by angle θ gives the transformation:

$$\begin{cases} x = x\cos\theta + y\sin\theta \\ y = y\cos\theta - x\sin\theta \end{cases},$$

and substituting this into Equation (5) gives:

$$x^2\left(\frac{\cos^2\theta}{a^2}+\frac{\sin^2\theta}{b^2}\right)+2xy\cos\theta\sin\theta\left(\frac{1}{a^2}-\frac{1}{b^2}\right)+y^2\left(\frac{\sin^2\theta}{a^2}+\frac{\cos^2\theta}{b^2}\right)=1. \tag{6}$$

The distance between a measured position $\left(s_x,s_y\right)$ and the predicted position $\left(\hat{s}_x,\hat{s}_y\right)$, measured in units of standard deviation squared, is given by:

$$\left(n\sigma\right)^2=\mathbf{s}^{\mathrm{T}}\mathbf{H}\mathbf{P}\mathbf{H}^{\mathrm{T}}\mathbf{s},$$

where $\mathbf{s}=\begin{bmatrix} s_x-\hat{s}_x \\ s_y-\hat{s}_y \end{bmatrix}$. Multiplying this out gives:

$$(n\sigma)^2 = \frac{(s_x - \hat{s}_x)^2\ \mathrm{var}(s_y) - 2(s_x - \hat{s}_x)(s_y - \hat{s}_y)\mathrm{cov}(s_x, s_y) + (s_y - \hat{s}_y)^2\ \mathrm{var}(s_x)}{\det(\mathbf{HPH}^{\mathrm{T}})}. \qquad (7)$$

However, this is the equation of an ellipse, so:

$$\frac{\mathrm{var}(s_y)}{\det(\mathbf{HPH}^{\mathrm{T}})} = \frac{\cos^2\theta}{a^2} + \frac{\sin^2\theta}{b^2}$$

$$\frac{\mathrm{var}(s_x)}{\det(\mathbf{HPH}^{\mathrm{T}})} = \frac{\sin^2\theta}{a^2} + \frac{\cos^2\theta}{b^2}$$

$$-\frac{\mathrm{cov}(s_x, s_y)}{\det(\mathbf{HPH}^{\mathrm{T}})} = \sin\theta\cos\theta\left(\frac{1}{a^2} - \frac{1}{b^2}\right)$$

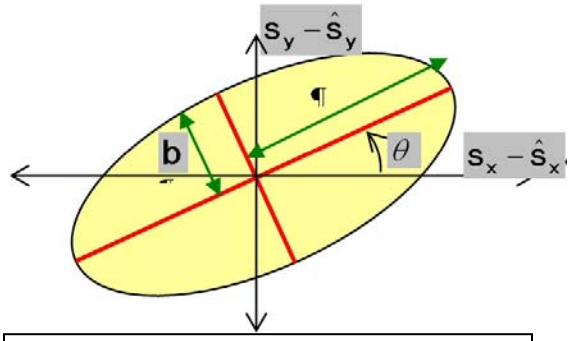

Fig. 1: Error ellipse given by state covariance matrix

If $n\sigma = 1$. Solving for a, b, and θ gives:

$$a = \sqrt{\frac{\mathrm{var}(s_x) + \mathrm{var}(s_y) + \sqrt{(\mathrm{var}(s_x) - \mathrm{var}(s_y))^2 + (2\,\mathrm{cov}(s_x, s_y))^2}}{2}}$$

$$b = \sqrt{\frac{\mathrm{var}(s_x) + \mathrm{var}(s_y) - \sqrt{(\mathrm{var}(s_x) - \mathrm{var}(s_y))^2 + (2\,\mathrm{cov}(s_x, s_y))^2}}{2}}$$

$$\theta = \frac{1}{2}\tan^{-1}\left(\frac{2\,\mathrm{cov}(s_x, s_y)}{\mathrm{var}(s_x) - \mathrm{var}(s_y)}\right).$$

When $\mathrm{var}(s_x) = \mathrm{var}(s_y)$, the angle of rotation is irrelevant because the ellipse reduces to a circle. A graph of the ellipse shows this information (Fig. 1).

The distance given by Eq. (7) is used in determining whether an object has maneuvered, or changed course while the Kalman filter is running. A certain tolerance level (usually 2σ or 3σ) is built into an implementation of the filter. If the distance between the measured and predicted positions exceeds the tolerance level during a set number of concurrent timesteps, the Kalman filter must be reinitialized using the last two measurements. Fig. 2 shows a graph of a maneuvering object with error ellipses and predictions superimposed on the object's course.

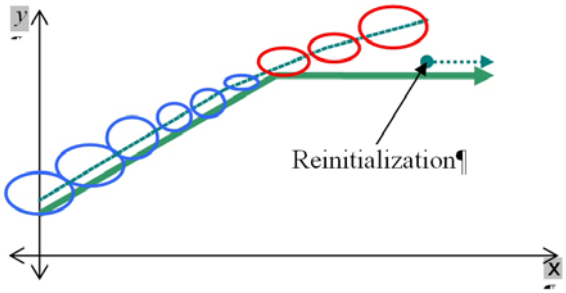

Fig. 2: Course of a maneuvering target with superimposed error ellipses and Kalman predicted positions

Estimating initial values for the state covariance matrix is one of the most difficult parts of running the Kalman filter algorithm. Reasonable values are chosen for each element in the initial state covariance matrix $\mathbf{P}(0 \mid 0)$ based on what is known about the system. They are updated by the following equation:

$$\mathbf{P}(k+1 \mid k) = \Phi \mathbf{P}(k \mid k) \Phi^{\mathbf{T}} + \mathbf{Q} \tag{8}$$

The matrix will give a better prediction of error as the algorithm goes through more iterations.

Update

As the update stage begins, time k becomes $k+1$. The measurement residual $\mathbf{s}(k)$, the difference between the actual measured positions and the predicted positions, is given by:

$$\mathbf{s}(k) = \mathbf{y}(k) - \mathbf{H}\hat{\mathbf{x}}(k \mid k-1), \tag{9}$$

and the covariance of $\mathbf{s}(k)$ is:

$$\mathbf{S} = \operatorname{cov}(\mathbf{s}(k)) = \mathbf{H}\mathbf{P}(k \mid k-1)\mathbf{H}^{\mathbf{T}} + \mathbf{R}. \tag{10}$$

The residual covariance matrix is similar to the state covariance matrix, except that it only accounts for positions and it measures covariance between predicted and measured states rather than between predicted and actual states. When updating state vector estimates, the Kalman filter includes a weighting factor known as the Kalman gain matrix, given by:

$$\mathbf{K}(k) = \mathbf{P}(k \mid k-1)\mathbf{H}^{\mathbf{T}}\mathbf{S}^{-1}. \tag{11}$$

The state vector estimate is then updated by the equation:

$$\hat{\mathbf{x}}(k \mid k) = \hat{\mathbf{x}}(k \mid k-1) + \mathbf{K}(k)\mathbf{s}(k). \tag{12}$$

Combining Eq. (11) and Eq. (12) shows intuitively that the greater the covariance matrix $\mathbf{S}$ is, the less the Kalman gain matrix is. Accordingly, the difference $\mathbf{s}(k)$ between measured and predicted states is weighted less when added to the new state vector prediction because there is a much higher possibility of error, especially from measurement noise. The state covariance matrix is updated by:

$$\mathbf{P}(k \mid k) = (\mathbf{I} - \mathbf{K}(k)\mathbf{H})\mathbf{P}(k \mid k-1). \tag{13}$$

Polar Transformations

Although the two dimensional Kalman filter requires measurements to be in Cartesian coordinates, radar systems measure objects' positions using polar coordinates. To accommodate this problem, the measurement vector $\mathbf{y}(k) = \begin{bmatrix} r \\ \theta \end{bmatrix}$ must be converted using the transformation:

$$\begin{cases} x = r\cos\theta \\ y = r\sin\theta \end{cases}.$$

Because the matrices Φ, $\mathbf{H}$, $\mathbf{Q}$, and $\mathbf{P}$ do not involve transformations from polar to Cartesian coordinates, they do not change from the forms listed above. However, because $\mathbf{R}$ measures the covariance of measurement error, which is given in polar coordinates, a new $\mathbf{R}$ is required. Measurement noise in *x* and *y* positions can be estimated by taking differentials of the above transformation:

$$\begin{aligned} \sigma_{s_x}^2 &\approx dx = d(r\cos\theta) = r\cdot d(\cos\theta) + \cos\theta\cdot dr \\ &= r\sin\theta\cdot d\theta - \cos\theta\cdot dr \\ &\approx r\sigma_\theta \sin\theta - \sigma_r\cos\theta \end{aligned}$$

$$\begin{aligned} \sigma_{s_y}^2 &\approx dy = d(r\sin\theta) = r\cdot d(\sin\theta) + \sin\theta\cdot dr \\ &= r\cos\theta\cdot d\theta + \sin\theta\cdot dr \\ &\approx r\sigma_\theta \cos\theta + \sigma_r\sin\theta \end{aligned}$$

$$\mathbf{R} = \begin{bmatrix} \sigma_{\varepsilon_x}^2 & \sigma_{\varepsilon_x}\sigma_{\varepsilon_y} \\ \sigma_{\varepsilon_x}\sigma_{\varepsilon_y} & \sigma_{\varepsilon_y}^2 \end{bmatrix}.$$

DEVELOPMENT OF THE PROGRAM

Program Background

The purpose of the program was to provide a general implementation of the Kalman filter. Initially, the program was very simple and only worked with Case 1. It was entirely linear and had no flexibility. The second version of our program was simply a copy of the first that was modified to work with Case 2. This program, too, was very hard to modify. We rewrote the third version of the program from scratch, in an attempt to deal with the modifications issues. The code that ran the filter was separated from the code that was involved in the initialization and put into its own function. Although this made it easier to modify the code for Case 3, most of the code was unable to be easily reused. It was at this point that we switched to object oriented programming style.

Breaking up the Program

Instead of having large chunks of un-reusable code, we broke down each task into a set of related functions and data, called classes. All of these code segments were easily reused, modified, and extended. The code became very modular, and we discovered that we could get all of the cases into the same program with little extra effort. The time needed to add additional cases also dropped.

The program was divided up into a number of classes. The main parts were the control loop; the KFilter class, which handled the filter operations; the Startup module, which handled initialization; the DataIterator, which reads the file and stores the data; and MatLib and other utility functions.

Structures and Classes

The most basic structure used in the program, called a "Datum," stores a position vector and the time. It is passed around between most of the classes in the program.

DataIterator is an interface that provides two basic methods for accessing data from an arbitrary input source. These two methods are hasNext() and nextDatum(). hasNext() indicates if there is still more data. nextDatum() gives the new piece of information to the calling function if new data exists. This information is stored in a Datum structure.

DataIterator has several implementing classes that perform various operations on the data before they are passed to the filter. The most basic of these classes is the FileReader. It simply reads the data in Cartesian coordinates (which can contain any number of dimensions) and places them into the vector, along with the time.

The PolarFileReader class extends the capabilities of the FileReader class. It reads polar coordinates from the input source and converts them into Cartesian so that the filter can work with them. Unlike the FileReader, it can only accept two dimensions.

The last implementation of DataIterator is the PolarMultiReader. It extends the capabilities of the PolarFileReader class by supporting input data from an arbitrary number of radars. The class is initialized with the coordinates of each radar, and data is converted to rectangular coordinates based on the coordinates of the current radar.

The KFilter class performs the filter's main calculations by carrying out the filter operations. These operations can be broken up into two phases: predict and update. After being initialized with the error covariance and state matrices, the filter's predict method can be called (which carries out the predict stage of the filter). Time is passed as an argument, and the filter predicts the next state of the target based on the time interval. All data is stored within the object instantiation. Once the predict stage is finished, the update stage commences. This involved calling the update method of the KFilter instantiation and passing in the measurement vector (which is returned by a DataIterator). The update method carries out the update stage of the filter. The KFilter class also contains an accessor method called getX(). This allows the current state of the filter to be obtained. Lastly, there is a reset() method that is used when the state and state covariance matrices need to be reset.

The CommaWriter class stores the results of the filter into a comma delimited text file. This format was chosen because it is easily opened in Microsoft Excel™. CommaWriter is initialized with the name of the output file and the number of dimensions. During initialization, it writes the appropriate headers to the file. CommaWriter also contains the writeLine() method which takes a time and vector as arguments, and writes them to the output file. When the output is finished, the close() method is called, which closes the file.

Additional Functions

The main sub combines the functions of all of these classes into a coherent program. It begins by asking the user which case to run. At this point, it performs case-specific initialization of the error covariance matrices and other variables. It also sets the correct input and output files, and instantiates the correct implementation of DataIterator. It then begins the main loop, which reads a Datum from the DataIterator, calls predict() with the time from the Datum, updates the filter with the measurement vector, and writes the output through the CommaWriter. When no more data is present, the program exits.

The initialize method generates the initial state vector. It is given the first two sets of coordinates from the DataIterator, and it returns a vector with the last position and the average velocity.

Also heavily used in the program is the MatLib [5] library. This library contains functions that can perform basic matrix arithmetic. The only difficulty is that equations need to be converted into prefix notation (as opposed to infix notation) to work with the matrix library. In general, this involves looking at the equations and recursively going through the order of operations backwards. All of the functions are used heavily in the KFilter and initialization functions.

In addition, a number of functions that generate matrices that are dimension dependent were written so that we would not need to hard-code the values in for each case.

Finally, one of the cases requires a function to determine how many standard deviations from the prediction the measurement is. This is done with the nSig() function. Given the x (prediction), y (measurement), and p (error covariance) matrices, it calculates $(n\sigma)^2$.

Each case requires certain assumptions to run properly. For all of the cases, we were given the error of the radar and the assumed driving noise. These values were hard-coded into the program. Each case also required certain adjustments that allowed it to understand the properties of the data read from the file. Therefore, for each case, we hard-coded the specific properties that were necessary.

Putting the Program Together

To implement the Kalman filter, we needed a way to initialize most of the matrices. Based on some guesswork, we were able to hard code most of the values into the program. Once the covariance matrices were initialized, we were still left with the problem of obtaining the initial state. The initialize() function does this by looking at the first two data points that the program receives, and uses them to make linear estimates of the position and velocity. This

initial state served as a platform on which to base future state estimates. One last problem that we ran into was that the time intervals between data points were not constant, although they were very close. This mattered when we were calculating the value of Φ. To find the change in position, we needed to know the change in time. To compensate for this, we created a function that returned the phi matrix based on the current time interval. Once this was completed, we then moved on to running the actual algorithm. For each data point read from the file, we predicted what the state at the next time interval would be. After this, we recalculated the state covariance and moved on to reading the next actual data point from the file. Once we had the data point, we updated the Kalman gain matrix, the state estimate, and the state covariance. The state estimate, along with the time, was printed to file. The program was then ready to repeat the process of reading the time interval, predicting, and correcting.

SCENARIOS

Case 1

Description:

Case 1 was the simplest of the problems we were given. It assumed that the target was a plane that was flying directly over the radar in one dimension. The plane's measured distance from the radar was given to us at each timestep.

Programming Changes:

Case 1 was very basic. Therefore, we only had to set the number of dimensions to one and make sure that the DataIterator was a FileReader.

Results:

Figure 3 shows two residual (difference) graphs. One shows the residuals between the measured positions and the actual positions at each timestep, and the other shows the residual between the predicted positions and the actual positions of the target at each timestep. The positions that were predicted by the Kalman filter were much closer to the actual points at almost every point. The radar-measured points had an average percent error of 14.4%, while the filter-predicted points had an average percent error of 2.32%. In addition, the graph shows how the performance of the Kalman filter improves over time: the predicted points get closer to the actual points, and fewer predictions are very far off. This shows the advantage of using the Kalman gain matrix in the algorithm, which decreases the influence of new measurements as the algorithm gains confidence and our estimates become closer to the actual data.

Figure 4 again shows how the performance of the Kalman filter improves over time. While the predicted velocity originally differed from the actual velocity by over 100 mph, it quickly corrected itself to get closer to the real value. By the end of the measurement time the predicted velocity was very close to the actual velocity, which is shown by the line approaching the x axis. Though the initial predictions were off, the filter adapted and corrected the mistakes after it read some more accurate points.

Case 2

Description:

Case 2 involved a plane moving in two dimensions that passed by the radar. It was moving in a straight path at constant velocity.

Programming Changes:

Case 2 was identical to Case 1 except that the number of dimensions was set to 2.

Results:

In Figure 5, the range (distance from the radar) and measured residuals are compared. Because of noise, the radar measurements were quite distorted. However, the predicted residuals were closer to the actual position. The filter had an error of 0.99% in the x direction and 12.13% in the y direction, while the radar alone had an error of 3.22% in the x direction and 12.91% in the y direction. The filter is still quite powerful in two dimensions.

Figure 6 shows the residual between the predicted velocity and the actual velocity. In the beginning, given only a few points, the filter once again showed a relatively high error. However, once more points came in and time progressed, it becomes evident that the filter becomes more and more accurate in its predictions. This once again shows the advantage of the Kalman filter over time, even when analyzing data in two dimensions.

Case 3

Description:

Case 3 involved a target moving in two dimensions that was tracked by a single radar. Coordinates from the radar were in polar form. The range represented the target's distance from the radar, and the angle represented the target's compass heading.

Programming Changes:

The number of dimensions remained at 2. A PolarFileReader was used in place of a FileReader. The R matrix had to be updated at each timestep between the predict() and update() methods according to the conversion of covariance matrices from polar to Cartesian. This accounted for the changing measurement covariance matrix.

Results:

Figure 7 shows how the Kalman filter helps to improve the measurements and bring them closer to the actual values. The performance, again improved over time. The starting conditions for this case were less than ideal, as shown by the first few predicted points, but by the end of the target's course, the predicted range was much closer to the actual range than was the measured range. The measured range had an average percent error of 6.36%, while the predicted range had an average percent error of 4.51%.

The velocity residual for Case 3 is shown in Figure 8. As with previous velocity residual graphs, the initial velocity residual was very far from 0. This was due to low velocity predictions for the initial condition. However, the radar was able to adjust to the conditions to give fairly accurate predictions of the velocities by the final time steps. The "predict and update" algorithm of the Kalman filter works well when using polar coordinates that are native to radar systems.

Case 4

Description:

In case 4, two radar stations monitored the target, with only one recording data at any given point in time. The measurements were given as polar coordinates relative to the active radar.

Programming Changes:

Case 4 was identical to Case 3 except that it used a PolarMultiReader instead of a PolarFileReader.

Results:

The noise associated with two radars did not throw the filter off. The predicted positions remained much better than the measured positions (see Figure 9). After the first few points, the filter became fairly accurate. It is noteworthy that between 6 and 6.5 minutes, when the radars switched, the filter was only off by 0.5 miles.

As in the previous 3 cases, the initial velocity estimates were very inaccurate. Simply averaging the first two points did not provide an accurate prediction. As the time moved on, however, the filter adapted and corrected the error (see Figure 10).

Case 5

Description:

In Case 5, we were attempting to track a UFO that was maneuvering to elude our radar. The UFO changed velocity twice. Coordinates were given in polar form.

Programming Changes:

Case 5 was very similar to Case 3, except we needed a way to let the filter know when the target maneuvered. In order for the filter to recognize the changes in velocity, it had to check the accuracy of its model at each timestep. To do so, it checked how many standard deviations the measurement was from the prediction based on the error covariance matrices. This was done in the nSig() function. If the measurement was at least four standard deviations from the prediction for three concurrent timesteps, the filter was reset by calling the filter's reset() method. This reinitialized the state covariance matrix, P, and reset the state to the previous measurement.

Results:

Figure 11 shows that the predicted positions were slightly worse after each turn, because the filter believed that the object was moving in a straight path. However, once the error was too big, the filter reset and was able to make more accurate predictions.

The velocity graph (Figure 12) still follows the trend of fairly inaccurate initial predictions. The predicted velocity approached the true velocity until the object turned, causing a large error velocity residual. The filter reset with another fairly inaccurate prediction but once again approached the true velocity. The final turn caused the same problems but at a smaller scale.

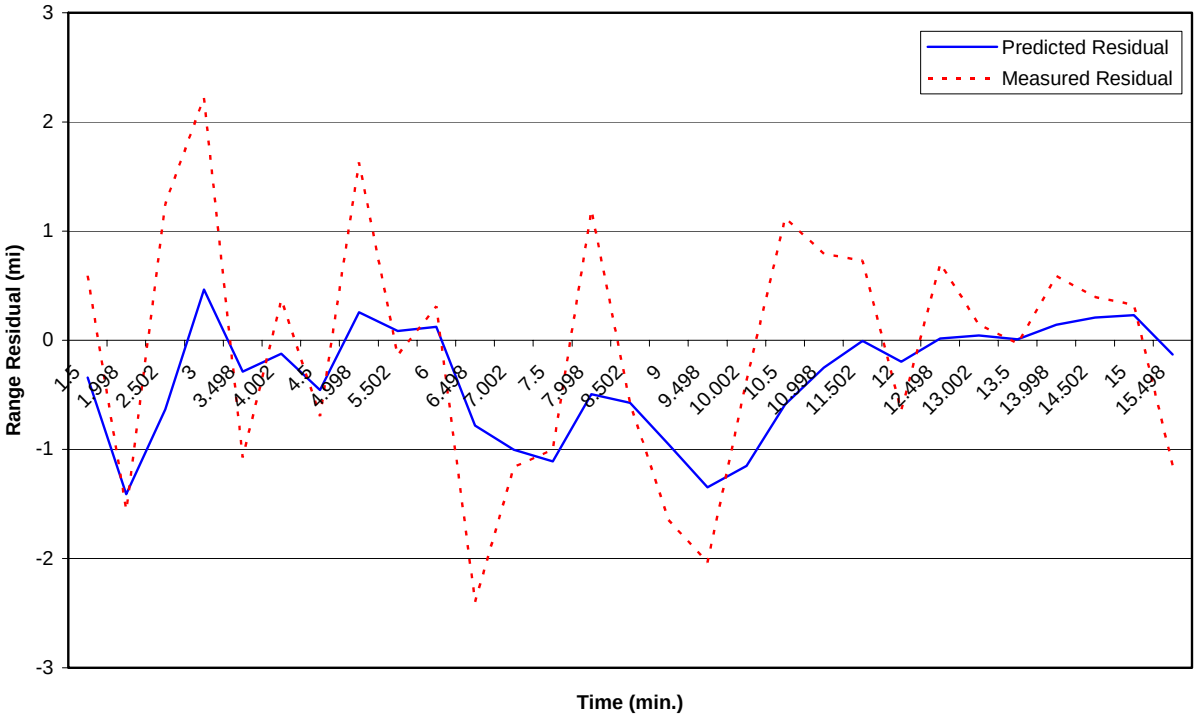

Fig. 3: Range Residuals for Case 1

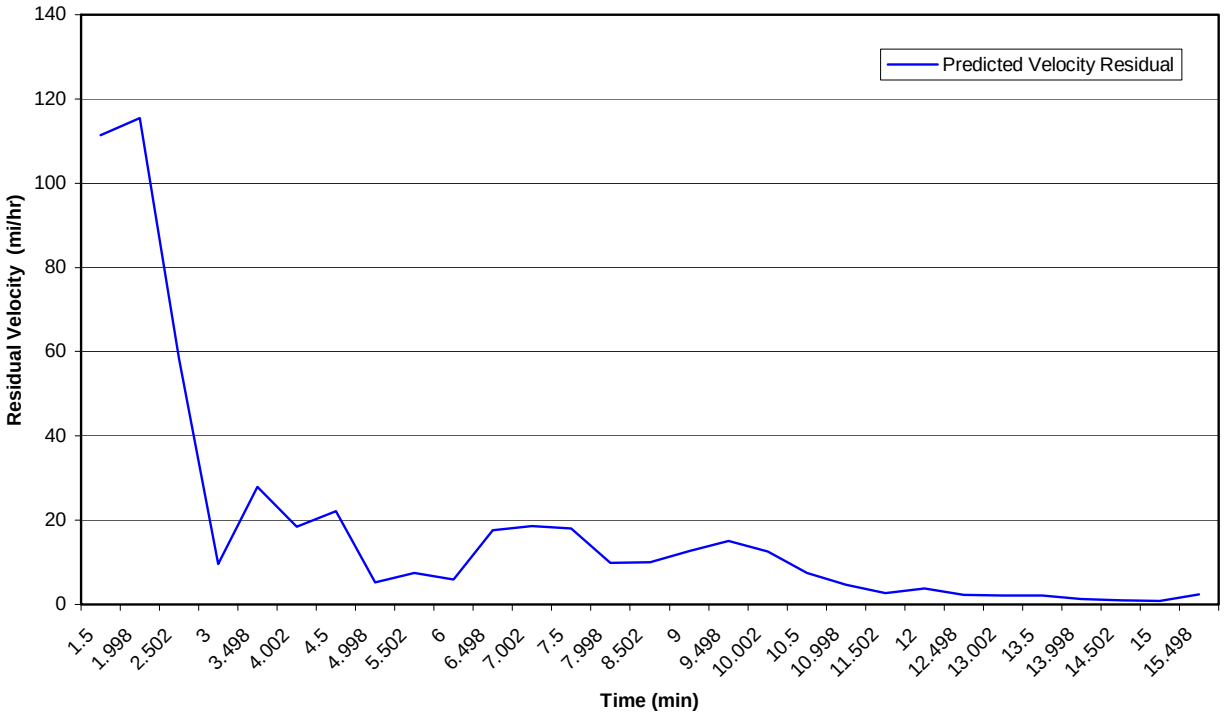

Fig. 4: Velocity residuals in Case 1

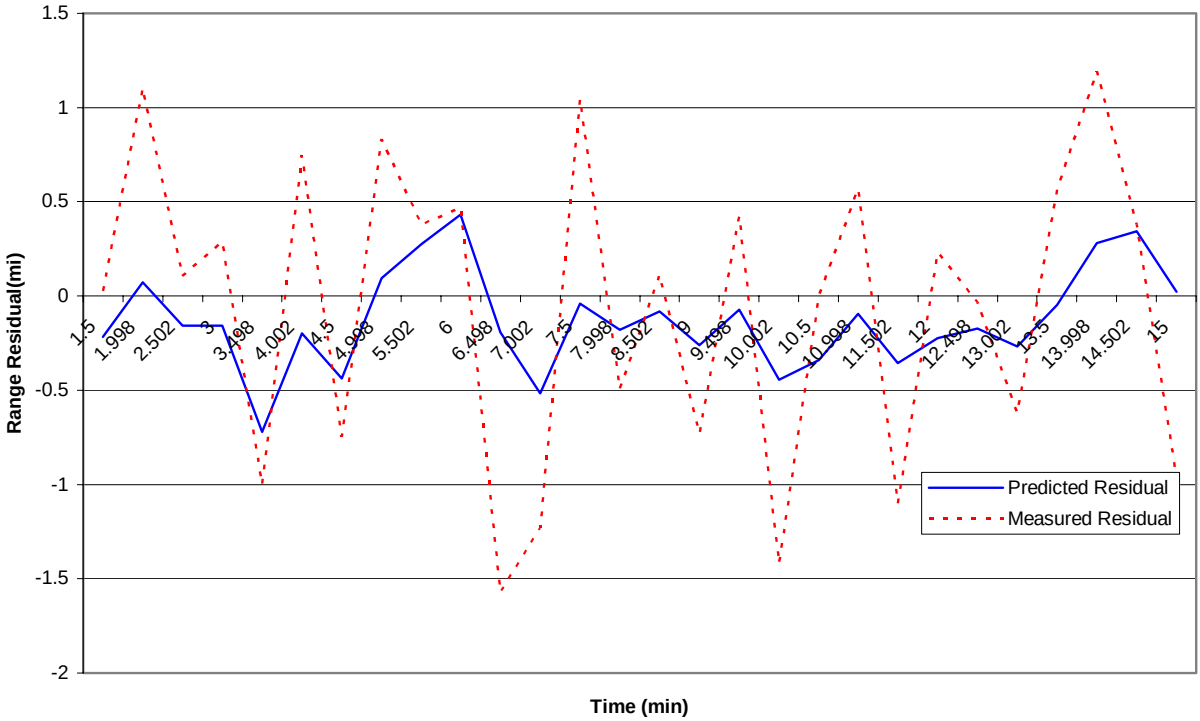

Fig. 5: Range residuals in Case 2

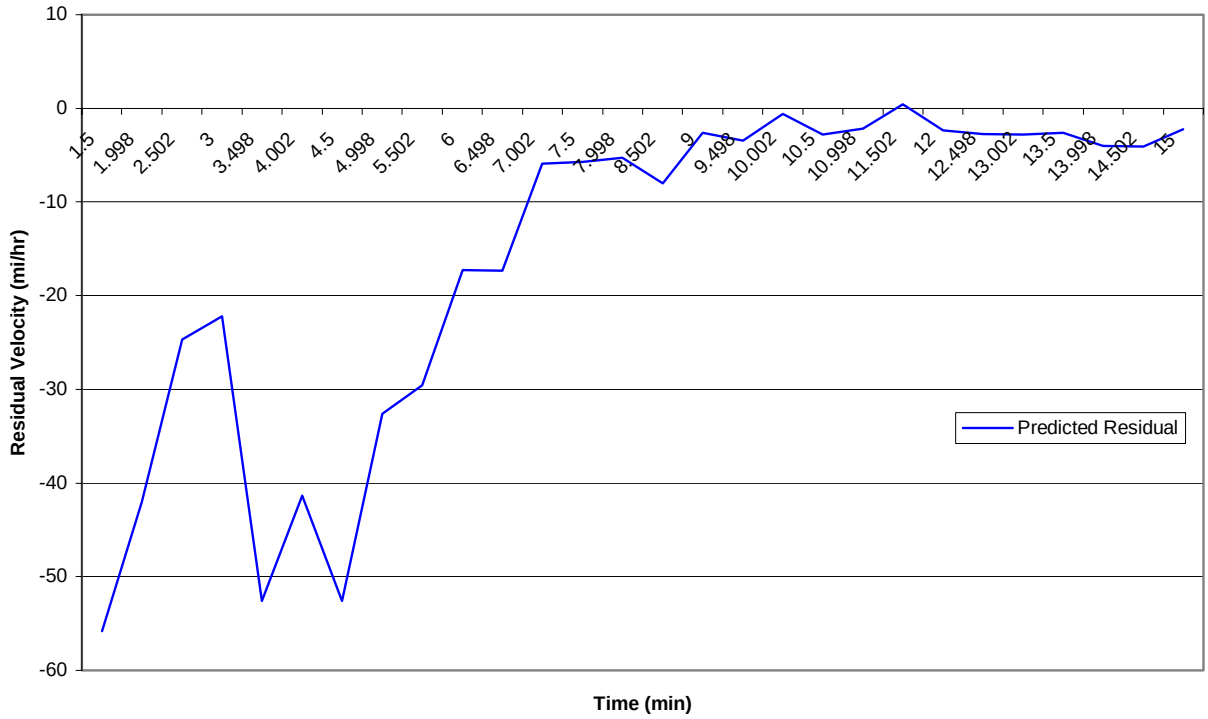

Fig. 6: Velocity residuals in Case 2

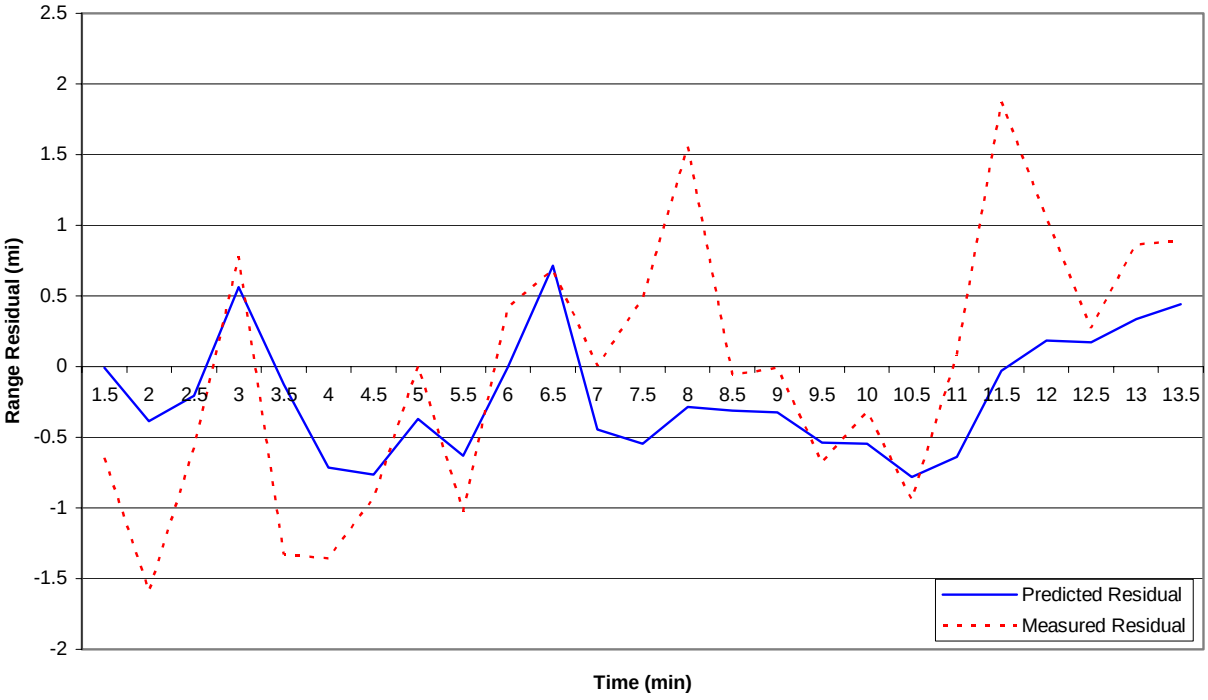

Fig. 7: Range residuals in Case 3

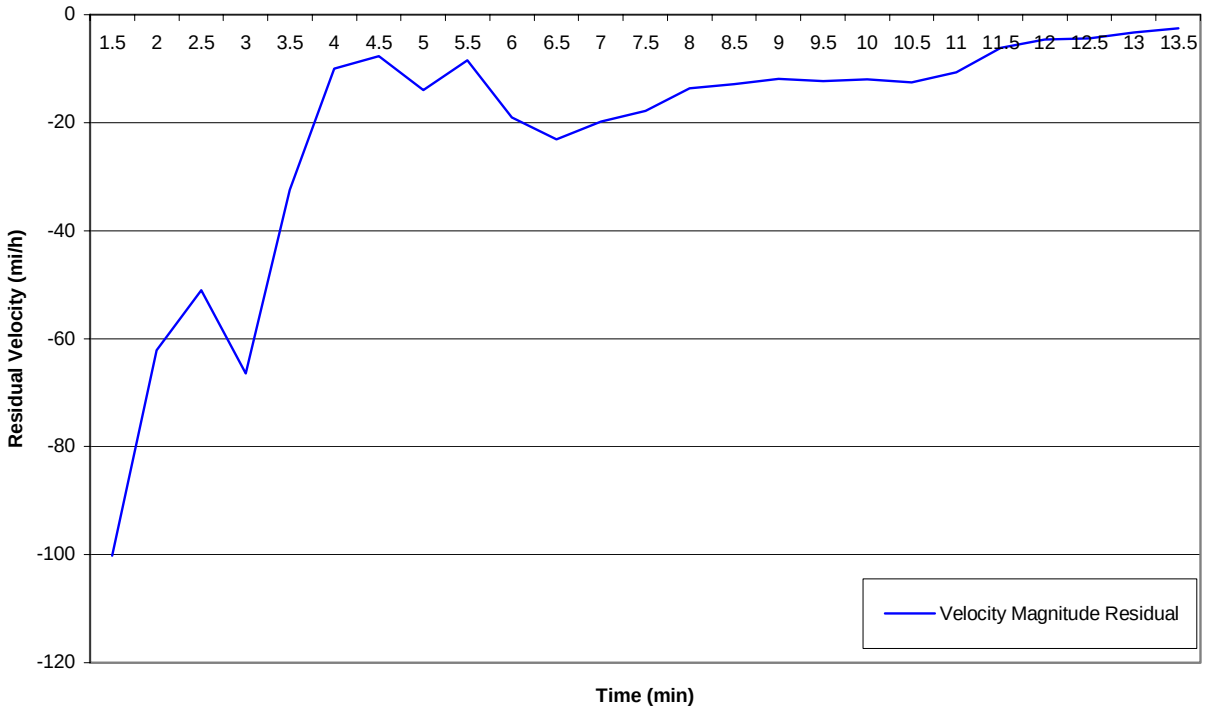

Fig. 8: Velocity residuals in Case 3

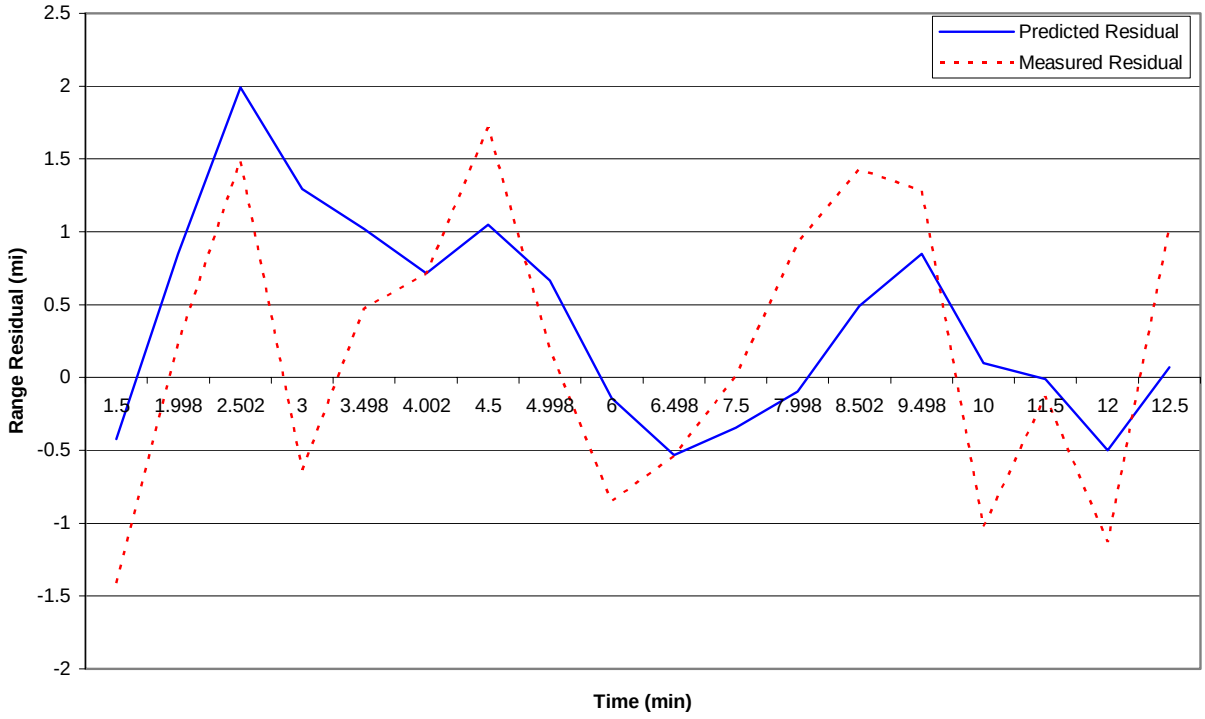

Fig. 9: Range residuals in Case 4

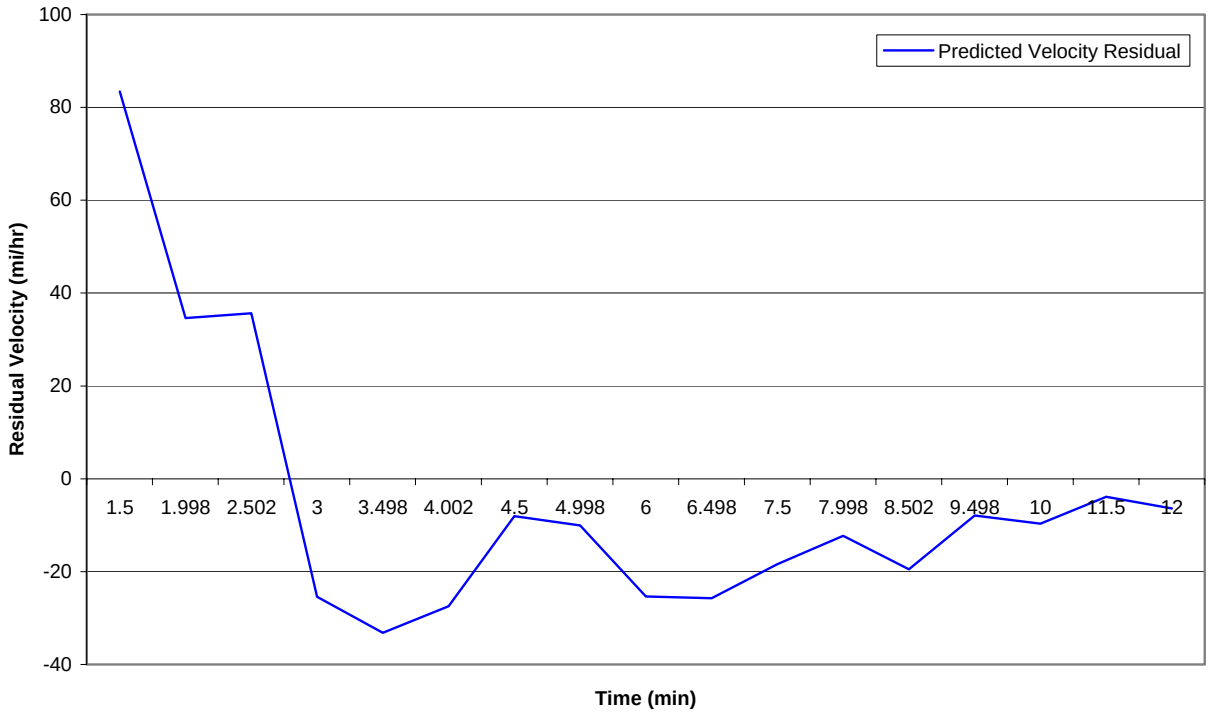

Fig. 10: Velocity residuals in Case 4

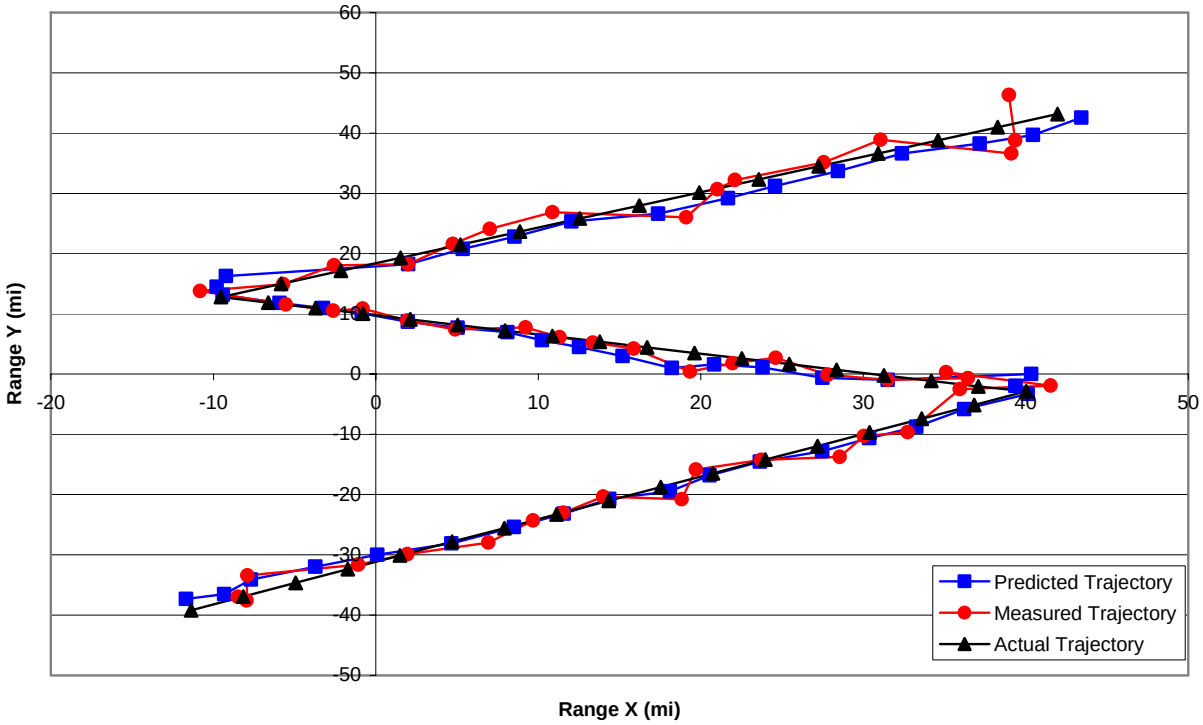

Fig. 11: Trajectory in Case 5

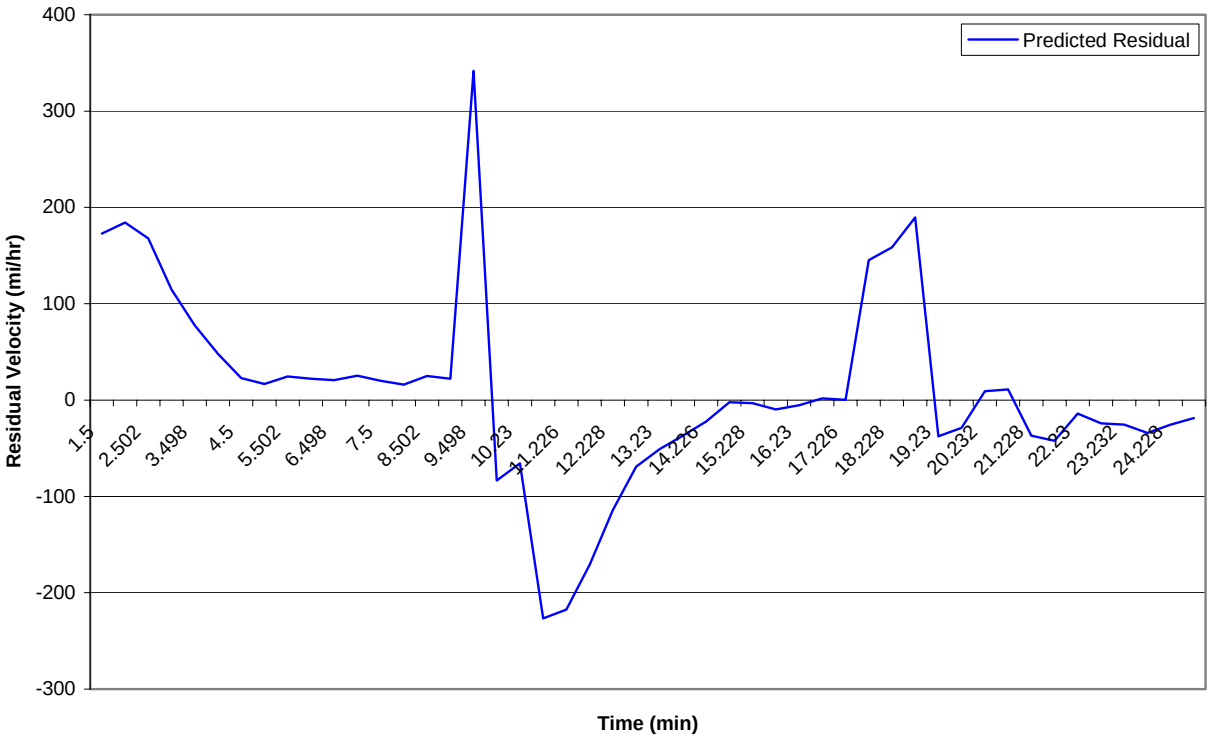

Fig. 12: Velocity residual in Case 5

CONCLUSION

The Kalman filter uses linear algebra to predict the position and velocity of a target. Based on the predictions and the radar measurements, the filter is able to correct errors. The filter's internal correction methods can be used to adapt the filter to figure out when a target is maneuvering by resetting the parameters and beginning the predictions from a new point, as was seen in Case 5. Utilizing a powerful matrix library, the relatively simple functions of the Kalman filter were easily adapted to create a Visual Basic .NET application.

Because radar alone provides abysmal estimates of position and cannot directly measure velocity at all, a method for accurately determining both is needed. Based on our results, the Kalman filter is adept at both. By assuming a normal distribution of errors, it is able to quickly and efficiently correct measurement errors and generate velocities without storing large amounts of data and performing lengthy computations.

REFERENCES

[1] [IEEE] Institute of Electrical and Electronics Engineers. 2003 Jan 23. Rudolf E. Kalman. IEEE history center. <http://www.ieee.org/organizations/history_center/legacies/kalman.html> Accessed 2005 July 21.

[2] Absolute astronomy. 2005 July 17. Kalman filter. <http://www.absoluteastronomy.com/encyclopedia/k/ka/kalman_filter.htm> Accessed 2005 July 18.

[3] Simon, Dan. 2001 June 1. Kalman filtering. <http://www.embedded.com/shared/printableArticle.jhtml?articleID=9900168> Accessed 2005 July 18.

[4] Blackman, Samuel S. 1986. Multiple-target tracking with radar applications. Artech House, Inc. Norwood, MA.

[5] Anas SA. 2003 Jan 18. Matrix operations library .NET. <http://www.planet-source-code.com/vb/scripts/ShowCode.asp?txtCodeId=907&lngWId=10> Accessed 2005 July 21.