Linear Project Report

Matthew Simiele & Nevin Nedumthakady

This report will outline each function and script required to complete the linear algebra final project. All referenced figures are at the end of the document.

Part I: Dynamic Systems with Disturbances

Gen_state.m: This function used to create all the states that will be used later in the project. Inputs to the function are an initial state vector (initial_state), the number of state vectors (K), the time between iterations (T), the random input multipliers to the x and y velocity (x_velocity and y_velocity). The dynamic system modeled in this project relies on having the previous states available to be used so the next state can be calculated. The next state (q+1) is equal to the current state (q) times a state transition matrix (A) plus a B matrix that forms N linear combinations of the inputs and adds one of them to the current state times the column vector that holds the random σ values to be applied to the x and y velocity. The state transition matrix A is equal to: $\begin{bmatrix} 1 & t & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & t \\ 0 & 0 & 0 & 1 \end{bmatrix}$

The matrix B is equal to: $\begin{bmatrix} 1 & x_v * t * (m-1) & 0 & 0 \\ 0 & x_v & 0 & 0 \\ 0 & 0 & 1 & y_v * t * (m-1) \\ 0 & 0 & 0 & y_v \end{bmatrix}$

I used two different while loops to compute the separate components of Q (the output matrix containing all state vectors) the state transition matrix with no outside forces acting on it and the second computes the variation in the states. The final line of the function adds the state matrix with no variance to the matrix that contains only the variance thus giving you the matrix of the dynamic system 5 with variance the output of the function Q_o.

Project_1.m: This is a script file used to provide inputs to the Gen_state function and take the output from that function in order to plot the x and y-positions of the system. To test that both parts of the above function worked I used this script and modified the inputs to test certain cases. To test that the case where no variation is applied to the system I set x_velocity and y_velocity to zero (these are the variances not the actual velocities that are stored in the initial state vector) (Figure 1). After I knew that the first part of the function worked I could just use a disturbance value of anything to create variance in the dynamic system. To check that the positions of x and y were reasonable I used the plot command to check that it generally followed the path of x=y (Figure 2).

Part II: Using Sensors to Measure States

Measure_state.m: This function's main purpose is to take an inputted set of state vectors and measure them based on the user inputted C matrix (sensor matrix) and a vector containing the values for a diagonal matrix, D, to create a sensor measurement variance for a non-ideal sensor. One of the parameters of this function is that the values for the diagonal matrix (D) have to be positive. So the fist check is to create a while loop to parse through all the values in the user provided vector d that contains all the values for the diagonal matrix D to check and make sure they are positive and to output an error otherwise. The measurement contains two parts to its final set of measurement vectors much like the time-varying state vector described in part 1. The measurement vectors two parts are as follows the first being user inputted sensor matrix which controls how the current positions and velocities are measured times the current position and velocity. The second component takes the diagonal matrix created by the user specified vector and multiplies it by a series of randomly generated column vectors W. The while loop creates the Y matrix with a series of measured states by computing both components of the measured position and adding them together.

Project_2.m: This script was used to test the functionality of the newly created measure_state function in use with the function created in part 1 (gen_state). To test that the first part of the function was working correctly I overrode the condition that all the diagonal matrix values had to be greater than zero so that no variance was present in the system at all (Figure 3). The plot consists of the generated position and the position measured by the sensors. After the test that the measure state code worked I applied the suggested variance to the system (Figure 4).

<u>Part III: Exploiting the Dynamic Model to Improve Sensor Measurement</u>

RLS_Estimation.m: The main goal of this function is to create an accurate measurement of the position of an object based on the measured positions and velocities, the state model, and a matrix with forgetting factors. This function uses the recursive least squares algorithm (RLS) to compute the estimated positions. The RLS algorithm is defined as "an adaptive filter algorithm that recursively finds the coefficients that minimize a weighted linear least squares function relating to the input signals." In each iteration of the algorithm 3 values are computed in each iteration: the Kalman Gain (K), Covariance Matrix (P), and the estimated states (q_hat). There is one more step before you can start iterating through the states and that is to initialize the first covariance matrix to the identity matrix. The value of q_hat(-1) is initialized to a zero vector in the case of my program. Every time the while loop iterates it creates a new Kalman Gain Matrix (K(n)), a new Covariance Matrix (P(n)) and computes a new value of an estimated state (q_hat(n)). In the function I wrote I keep all the previous values of the Kalman Gain Matrix and the Covariance Matrix which creates a 3-d matrix of depth the number of iterations + 2.

Project_3.m: This script is used to test the use of the RLS_estimation function in tandem with the functions written in the two previous parts of the lab (gen_state and measure_state). To test the above function, I used the same method as the previous parts by eliminating the measurement error and the variance in the path I should be able to create an RLS estimation of the same line generated by the gen_state and measure_state functions. After this was true I generated a plot using the suggested values for the script and saw that the estimated positions almost followed a linear pattern which they should so I concluded that my functions were working correctly (Figure 5).

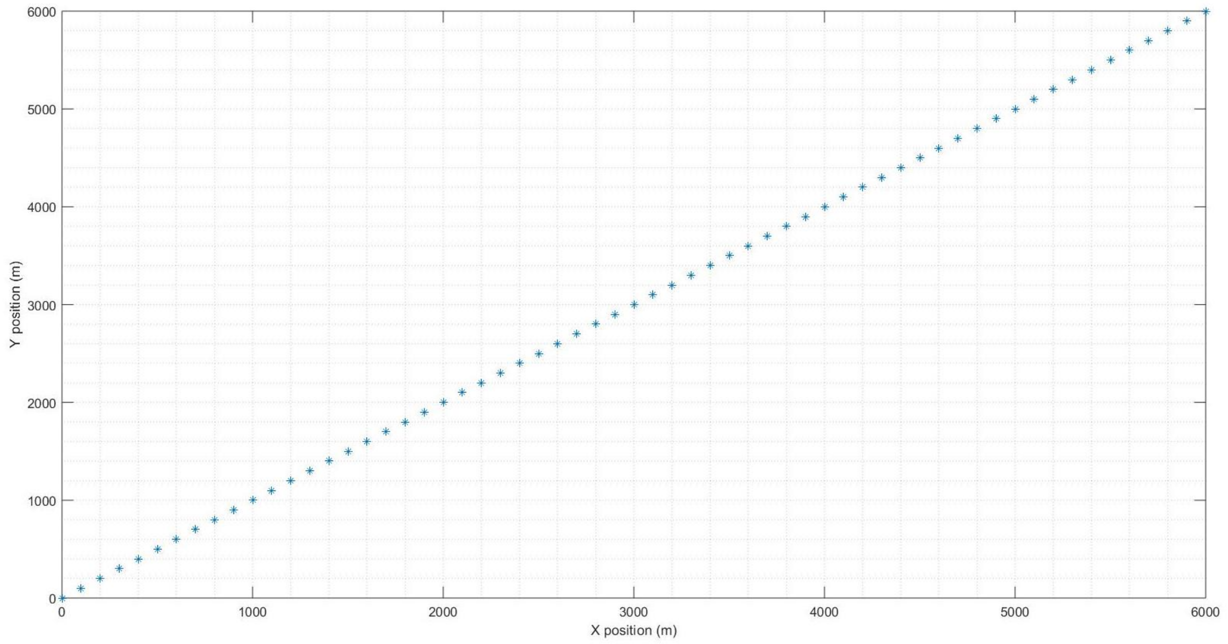

Figure 1: Gen_state w/o variance

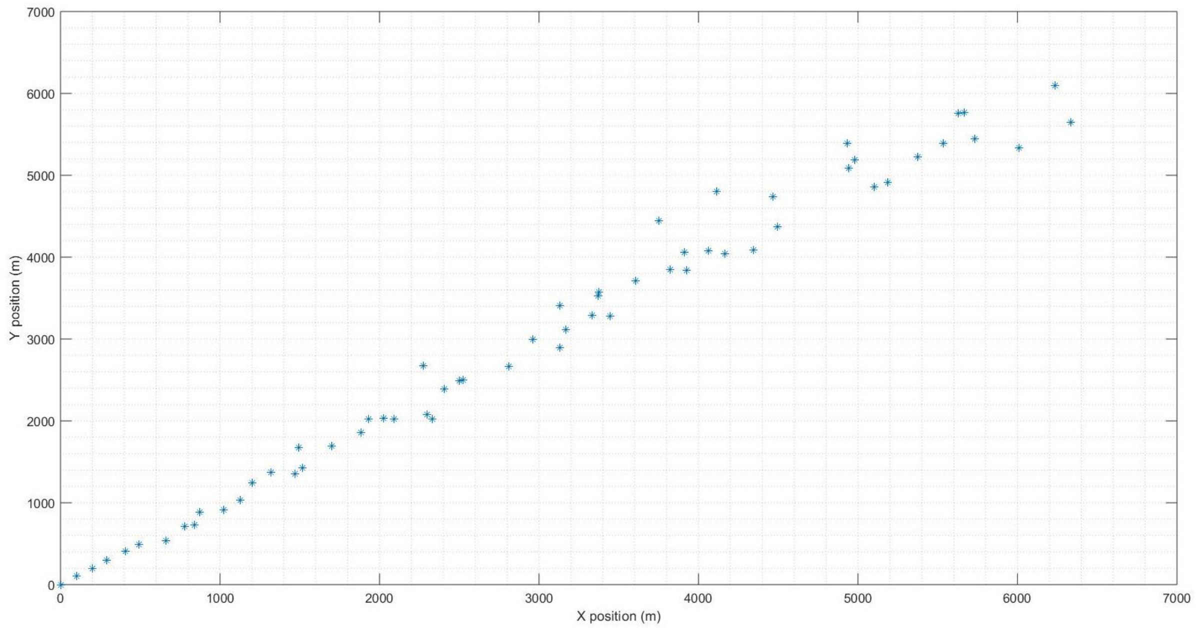

Figure 2: Gen_state w/ variance

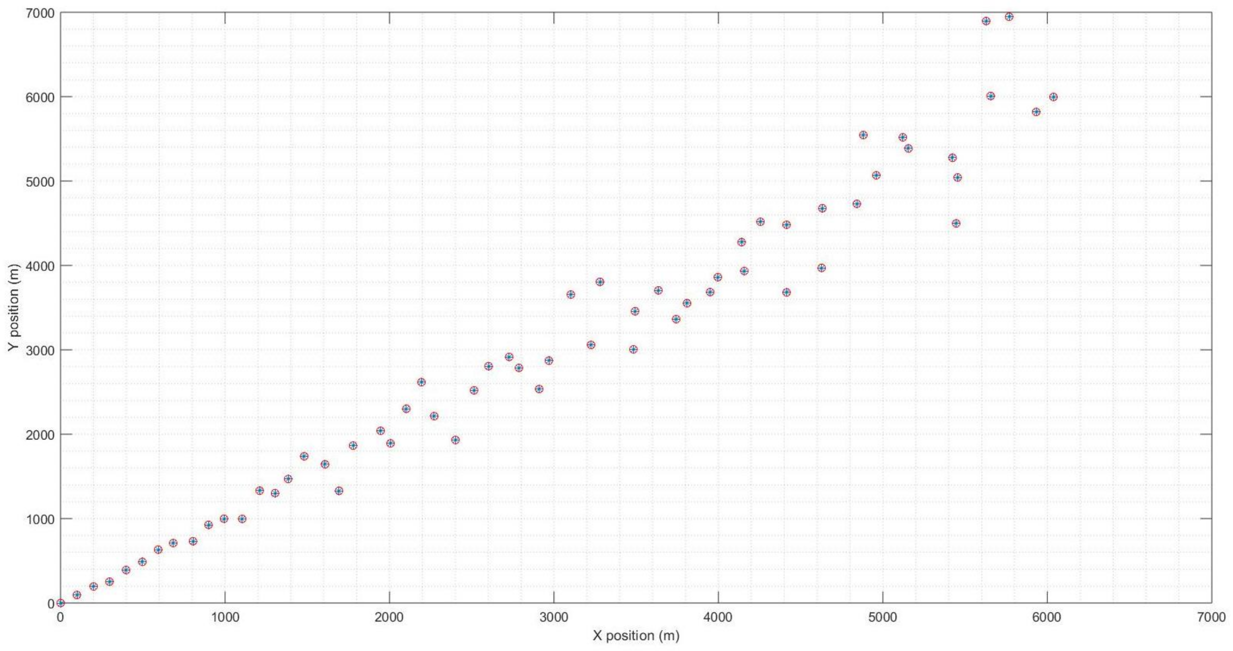

Figure 3: No measurement standard deviation

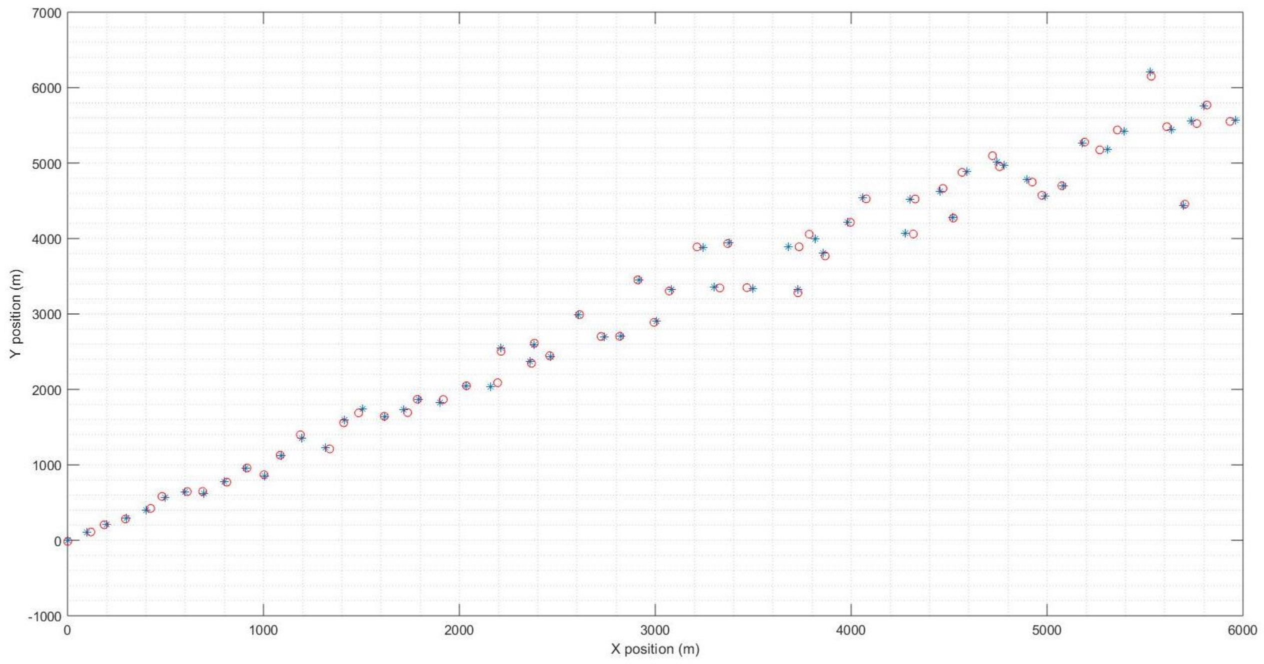

Figure 4: Measurement Standard Deviation set to Suggested Values
(X and Y position set to 25 and X and Y velocity set to 10)

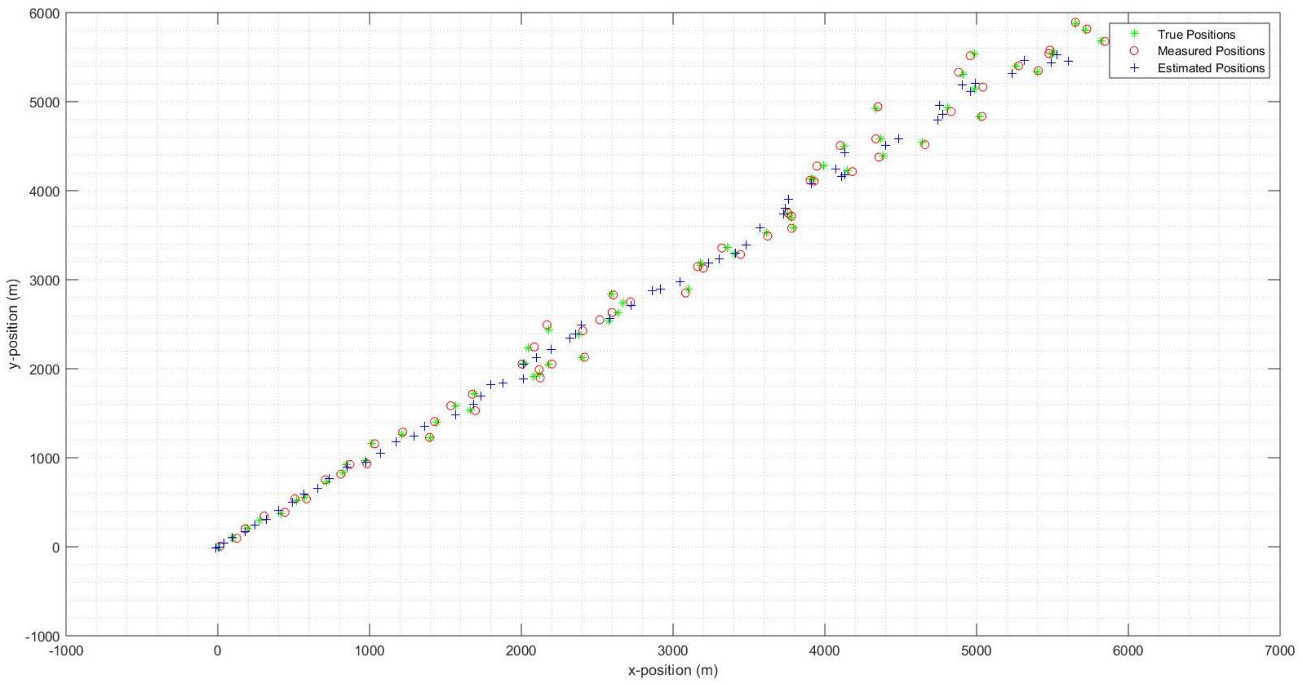

Figure 5: Part 3 RLS Estimation with True and Measured Positions