KALMAN FILTER IMPLEMENTATION

DECEMBER 29, 2021

Submitted to: Prof. Dr. Kemalettin Erbatur

Name of Student: Moses Chuka Ebere
Student Number: 31491
Course: Autonomous Mobile Robotics (ME 525) Lab

TABLE OF CONTENTS

INTRODUCTION

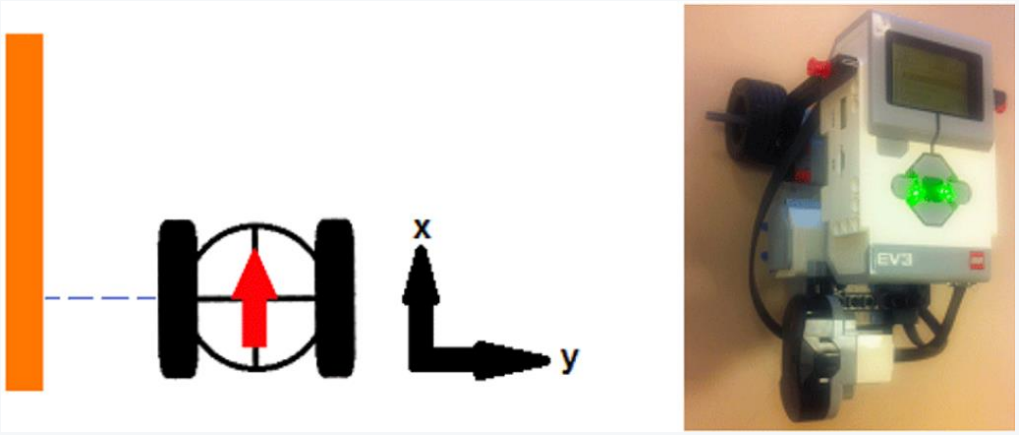
This lab involves the implementation of a Kalman filter on a two-wheeled differential drive robot equipped with an infrared sensor.

In this implementation, the robot is made to move along the x-axis while the infrared sensor (mounted at 90 degrees to the motion trajectory of the robot) is used to measure the distance in the y-axis. The aim of this experiment is to identify the drift (or deviation) along this axis and estimate it using a suitable 1D Kalman filter.

The differential drive robot is built using the Lego EV3 toolbox, and the codes are generated using MATLAB.

PROCEDURE

Firstly, the differential drive robot is assembled using the necessary parts as shown in the figure below and the camera is mounted:



The major components used are:

- Two large motors
- Two castor wheels
- Two actuated wheels
- The EV3 Brick
- An infrared sensor

MATLAB is the underpinning software for this application.

To perform the necessary tasks, a program is written in MATLAB. Prior to this, a USB connection is established between the robot and the PC on which the MATLAB code is written.

Problem Formulation

After creating a connection to the Ev3 and camera, the motors and IR sensor are properly defined. The following steps are now taken:

- The problem is modelled.
- To implement the Kalman filter, we need the model (plant/process) and the measurements. These are reflected by the following equations:

a. Process equation –

$$\mathbf{x}_k = \mathbf{A}\mathbf{x}_{k-1} + \mathbf{B}\mathbf{u}_k + \mathbf{w}_{k-1}$$

Where the process noise $\mathbf{w}$ is drawn from N(0,$\mathbf{Q}$), with covariance matrix $\mathbf{Q}$. This is obtained within the code.

b. Measurement equation –

$$\mathbf{z}_k = \mathbf{H}\mathbf{x}_k + \mathbf{v}_k$$

Where the measurement noise **v** is drawn from N(0,**R**), with covariance matrix **R**. This is outrightly defined as R = 0.001 since the variance of the measurement noise is really small.

- However, we are required to implement a 1D Kalman filter with no control input, so **A** = **H** = 1, **B** = 0, and $\mathbf{u}_k = 0$. Our equations become:

$$\text{Process model:} \quad x_k = x_{k-1} + w_{k-1}$$
$$\text{Measurement model:} \quad z_k = x_{k-1} + v_k$$

- The Kalman filter equations include:

$$1.\ \hat{x}_k^- = A\hat{x}_{k-1} + Bu_k \qquad 2.\ P_k^- = AP_{k-1}A^T + Q \qquad 3.\ K_k = P_k^- H^T\left(HP_k^- H^T + R\right)^{-1}$$

$$4.\ \hat{x}_k = \hat{x}_k^- + K_k\left(z_k - H\hat{x}_k^-\right) \qquad 5.\ P_k = \left(I - K_k H\right)P_k^-$$

- where equations 1 & 2 are the prediction steps, equation 3 is the Kalman gain, and equations 4 & 5 are the correction steps.

MATLAB Code

After formulating the problem and creating a connection to the Ev3 and camera, the motors and IR sensor are properly defined.

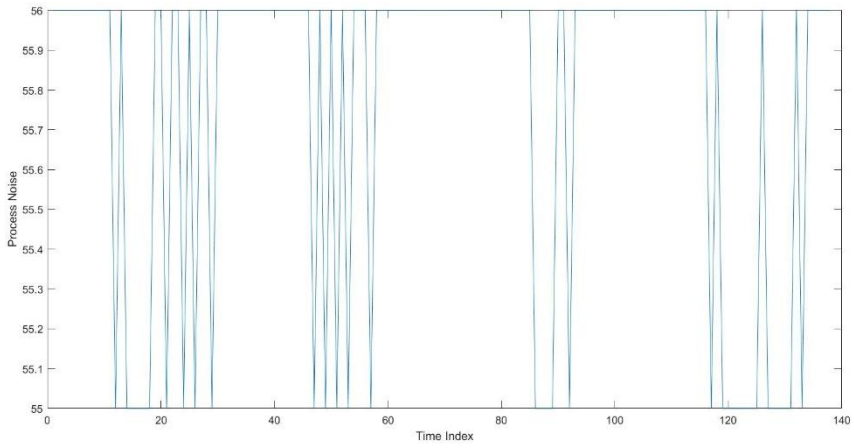
- The plant parameters like A, H, B, etc are defined alongside the noise parameter R and the initial values for the simulation (like the initial error covariance and initial distance estimation).
- Data arrays are also defined to store the process noise, estimations, measurements, error covariances, Kalman gain, etc.
- The execution time of 2 seconds is defined, and a timer function is set for the robot motion.
- The encoder rotations are reset, and the robot speed is defined.
- A nested while-loop is used to find the covariance of the process noise, **Q**. This is done by moving the robot back and forth and taking IR sensor readings which are simultaneously stored. After this, the variance is calculated, and process noise is plotted.
- A nested while-loop is now used to predict measure and correct the deviation along the y-axis using the previously defined equations.
- Finally, plots of the key parameters are obtained.

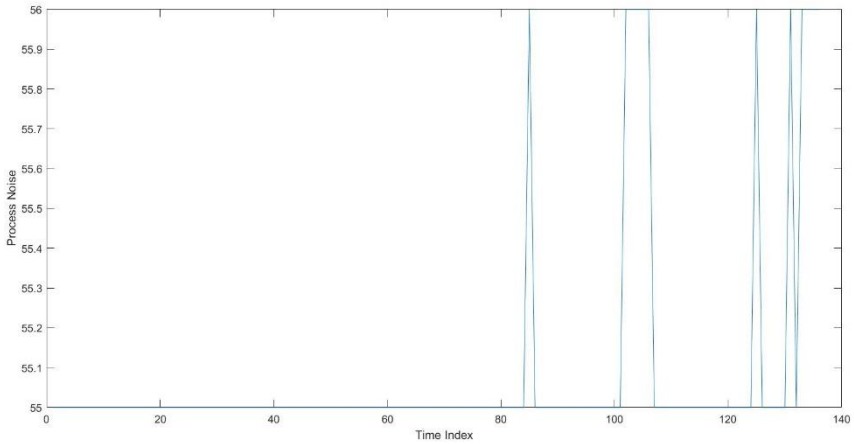
RESULTS AND COMMENTS

The following results (**for two different sets of measurements**) were obtained from the implementation of the MATLAB code on the differential drive robot equipped with the infrared sensor:

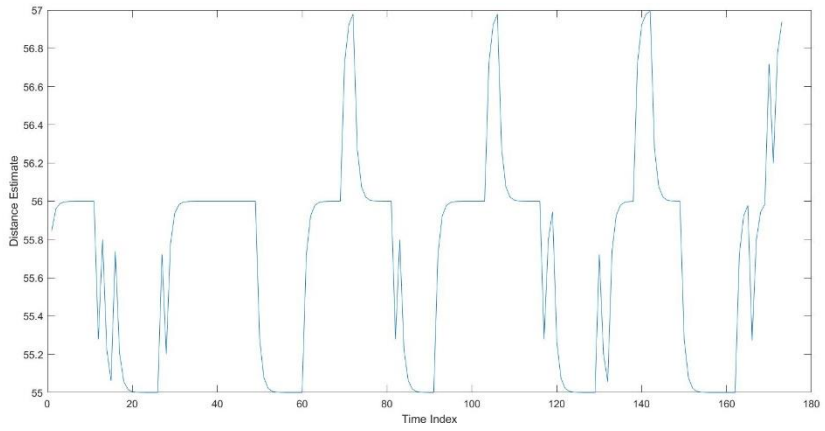
Variance Calculation and Process Noise Plot:

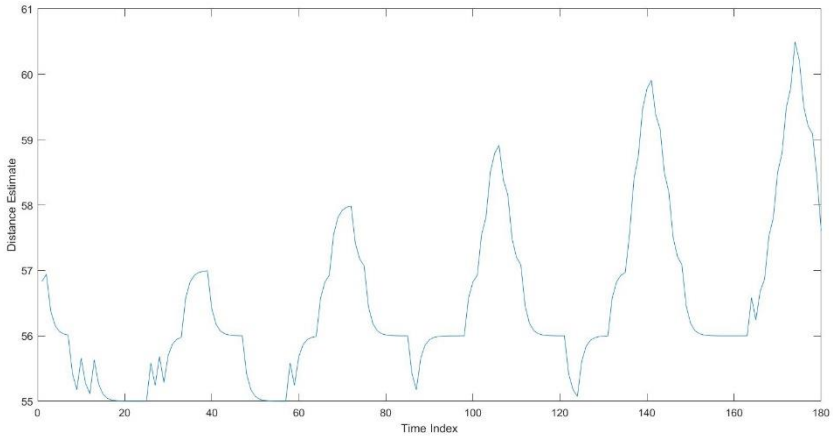
Going by the IR sensor readings, the obstacle (wall) was at a distance of about 56 (not normalized) from the robot. The readings were taken **two different times** with the following results:



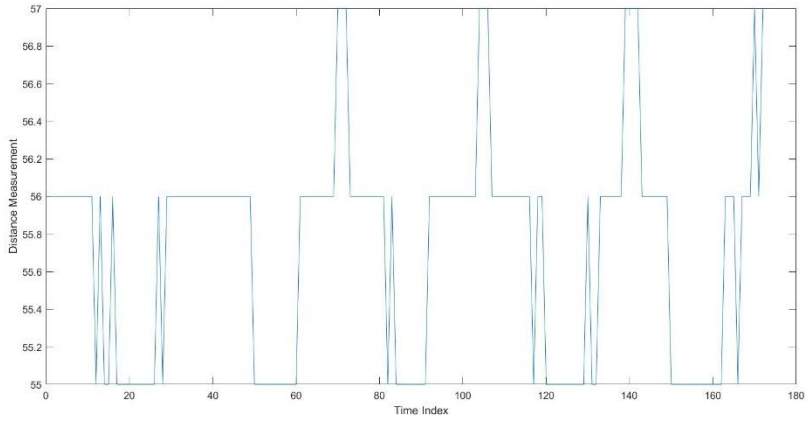


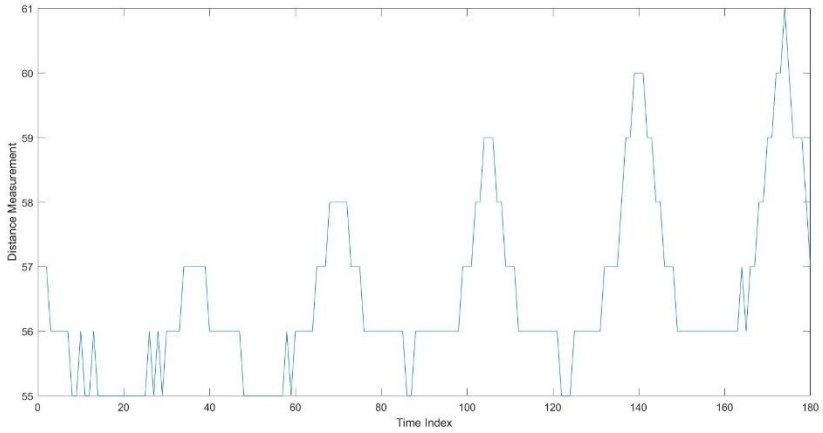
Distance Estimation (x_hat):



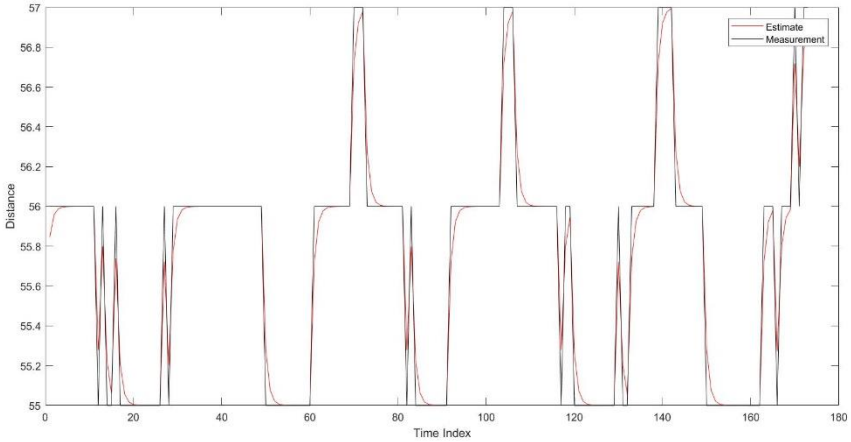


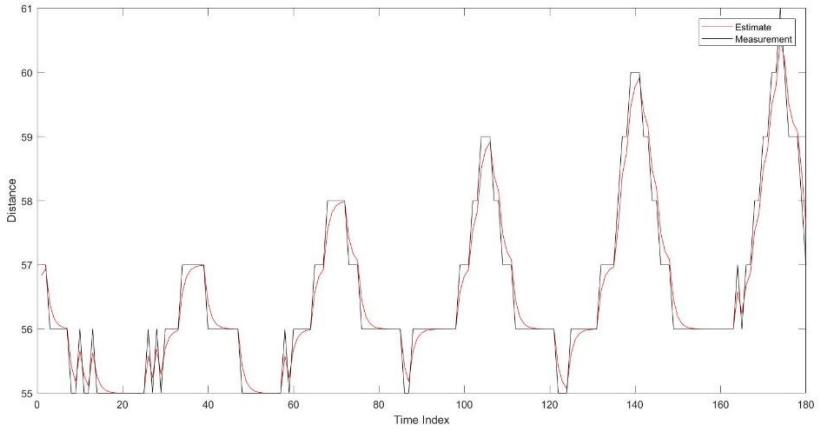
Measurements (z_k):



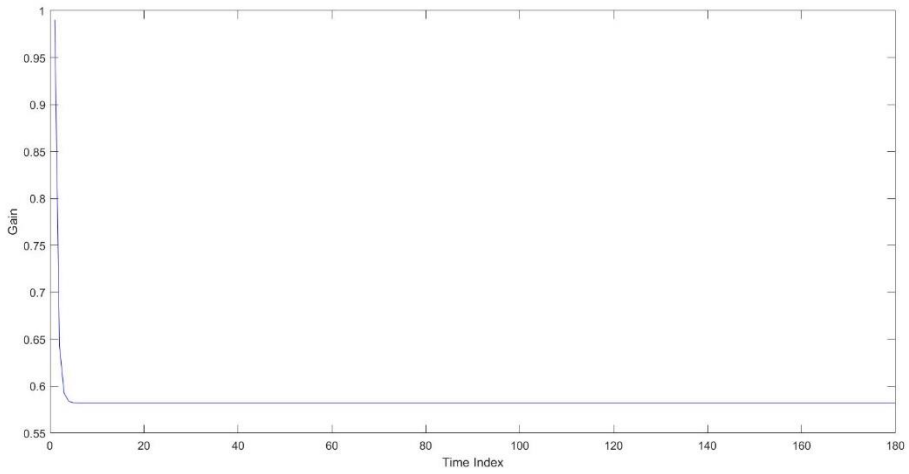


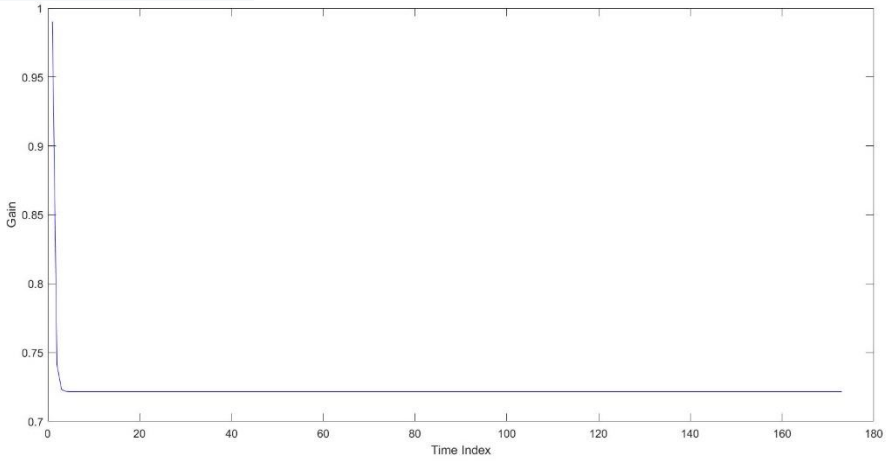
x_hat and z_k together:



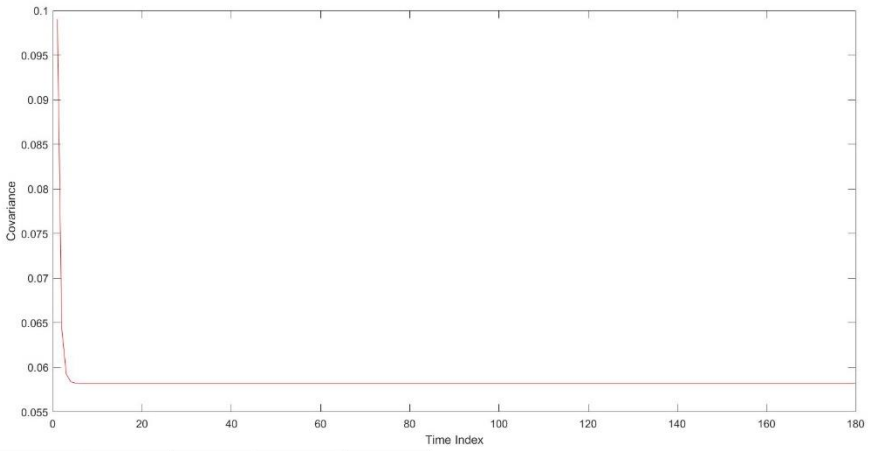


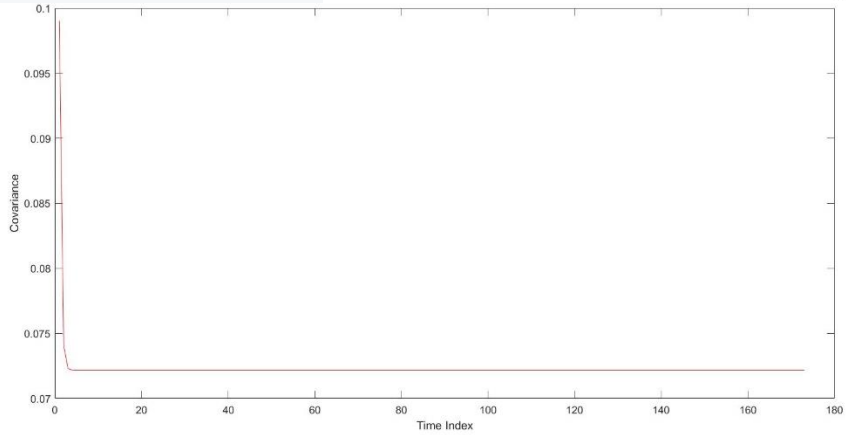
Kalman Gain:





Error Covariance:





DISCUSSION

Q1: The plot for this is included in the previous section.

```
while  ~readButton(mylego, 'up')
    %Forward Motion
    t = timer('TimerFcn', 'stat=false', 'StartDelay', EXE_TIME);
    start(t); stat = true;
    Rightmotor.Speed = speed;
    Leftmotor.Speed = speed;
    while stat == true && ~readButton(mylego, 'up')
        w_k_list = [w_k_list; double(readProximity(myirsensor))];
        pause(deltaT);
    end

    % Backward Motion
    t = timer('TimerFcn', 'stat=false', 'StartDelay', EXE_TIME);
    start(t); stat = true;
    Rightmotor.Speed = -speed;
    Leftmotor.Speed = -speed;
    while stat == true && ~readButton(mylego, 'up')
        w_k_list = [w_k_list; double(readProximity(myirsensor))];
        pause(deltaT);
    end
end
stop(Rightmotor);
stop(Leftmotor);
```

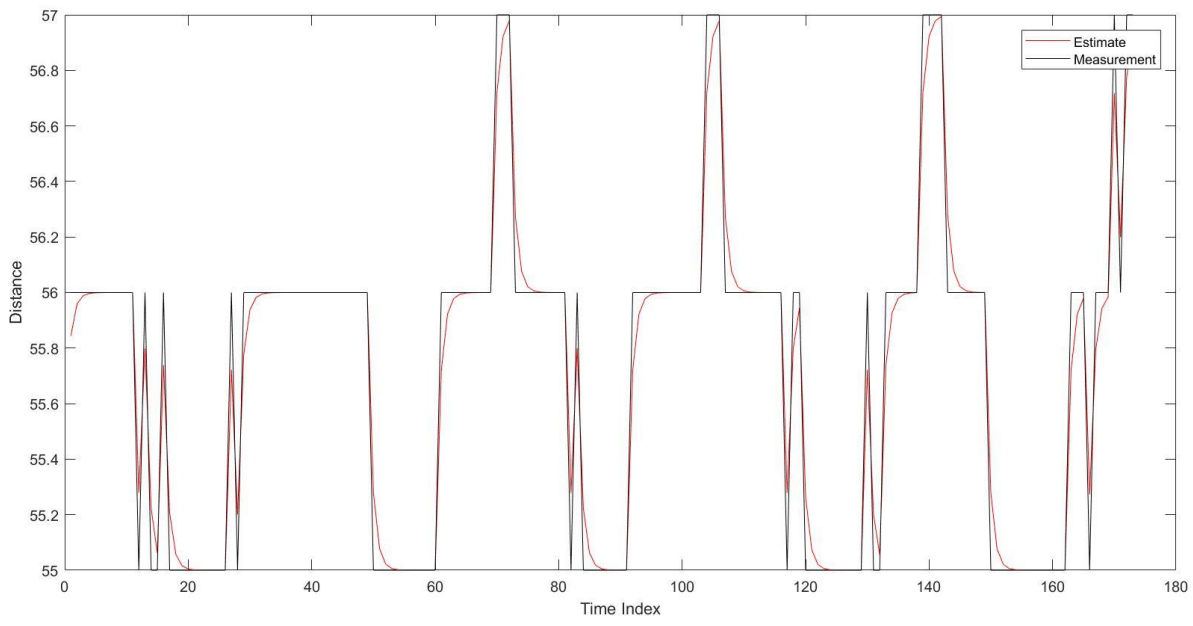
As seen in the screenshot on the left, I created a while-loop controlled by the 'up' button on the EV3 brick. Within this loop, I created separate loops for the forward and backward motion controlled by the timer status and the 'up' button. Within each loop, I measured the distance using the IR sensor and recorded the result in the w_k_list array which was previously defined. Each motion loop is designed to run for two seconds. After this, Q is calculated using variance function, var() in MATLAB.

```
Q = var(w_k_list);
```

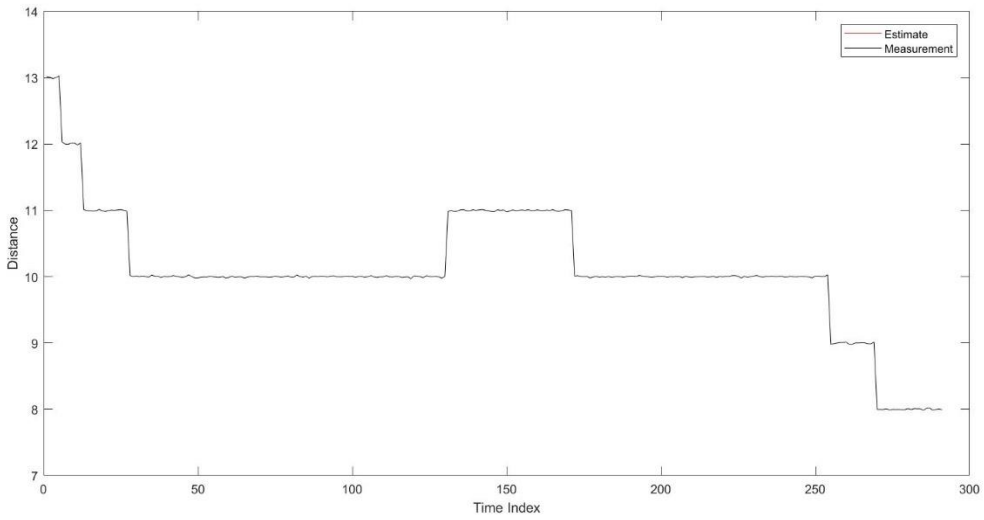
Q2: a. The plots are in the previous section.

b.. I started the process with an initial Error covariance (P) = 10. As we can observe from the plots in the previous section, the Kalman Gain (K) and Error covariance (P) gradually **converge** to a specific value. They both follow an **exponential curve** that becomes asymptotic to a specific value with time.

c..

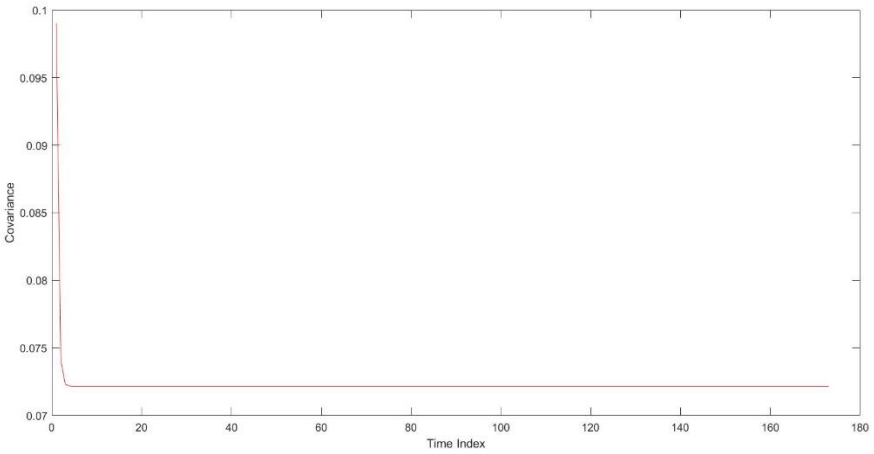


From the above plot, we see that the estimates are very close to the actual measurements. The deviations are found to occur when the measurement changes abruptly. I also observed that the result is heavily dependent on the noise parameter, R. When R is really small (e.g., 0.0001), the estimates and actual measurements are found to be almost exactly the same as in the following case:



In the above, R = 0.0001, so there's no noticeable difference between the estimates and the measurements.

d.. The variance P_k is found to converge to about 0.073 from an initial value of 10 (this was set as the initial value to start the computation). 0.073 is ~ 0.



CONCLUSION

This lab includes a condensed study on the implementation of a one-dimensional Kalman Filter in the estimation of the drift experienced by a moving different drive robot. The robot used was equipped with an IR sensor for the actual distance measurement aspect. As opposed to arbitrarily selecting noise parameters for the process, measurements were taken with the IR sensor to calculate the variance of the process noise. It was found that carefully aligning the castor wheels of the robot directly reduces variation in measurements. Using the Kalman filter equation, the desired parameters were predicted, measured, and corrected over 2-second forward and 2-second backward movements of the robot. The results show that a repetition of the predict-correct-measure cycle brings the estimations closer to the actual measurements while the Kalman gain converges, and the variance also converges to zero. Throughout the entire experiment, low motor speeds (10 and 20) were used to reduces the adverse effect of vibration on the process.

APPENDIX

```
clear all; close all; clc

mylego = legoev3('usb');

% Set up infrared sensor on port 3
myirsensor = irSensor(mylego, 3);

Rightmotor = motor(mylego,'C');
Leftmotor = motor(mylego,'B');

resetRotation(Rightmotor);
resetRotation(Leftmotor);

speed = 20;
Rightmotor.Speed = speed;
Leftmotor.Speed = speed;

start(Rightmotor);
start(Leftmotor);

% Plant Parameters
A = 1; H = 1; B = 0;
R = 0.1; % Noise Parameters

% Inital Values
u_k = 0;
P_k_minus_1 = 10;
x_initial = 0;
x = x_initial;
x_hat_k_minus_1 = 40;
K_k = P_k_minus_1*H'*inv(H*P_k_minus_1*H' + R);

% Record Arrays
x_list = []; x_hat_list = []; z_k_list = [];
K_k_list = []; P_k_list = []; w_k_list = [];

deltaT = 0.1; EXE_TIME = 2; PERIOD = 0.1;
t = timer('TimerFcn', 'stat=false', 'StartDelay',
EXE_TIME);

while  ~readButton(mylego, 'up')
```

```
    %Forward Motion
    t = timer('TimerFcn', 'stat=false', 'StartDelay',
EXE_TIME);
    start(t); stat = true;
    Rightmotor.Speed = speed;
    Leftmotor.Speed = speed;
    while stat == true && ~readButton(mylego, 'up')
        w_k_list = [w_k_list;
double(readProximity(myirsensor))];
        pause(deltaT);
    end

    % Backward Motion
    t = timer('TimerFcn', 'stat=false', 'StartDelay',
EXE_TIME);
    start(t); stat = true;
    Rightmotor.Speed = -speed;
    Leftmotor.Speed = -speed;
    while stat == true && ~readButton(mylego, 'up')
        w_k_list = [w_k_list;
double(readProximity(myirsensor))];
        pause(deltaT);
    end
end
stop(Rightmotor);
stop(Leftmotor);

Q = var(w_k_list); % Variance of process noise

figure %1
plot(w_k_list)
xlabel('Time Index')
ylabel('Process Noise')

% This is just to prepare the robot for the Kalman filter
implementation.
while  ~readButton(mylego, 'down')
end

Rightmotor.Speed = speed;
Leftmotor.Speed = speed;
```

```
start(Rightmotor);
start(Leftmotor);

i_a = []; % Counter Array
while  ~readButton(mylego, 'up')
    % Forward Motion
    t = timer('TimerFcn', 'stat=false', 'StartDelay',
EXE_TIME);
    start(t); stat = true;
    Rightmotor.Speed = speed;
    Leftmotor.Speed = speed;
    while stat == true && ~readButton(mylego, 'up')
        % Prediction Update
        x_hat_minus_k = A * x_hat_k_minus_1 + B * u_k;
        P_minus_k = A * P_k_minus_1 * A + Q;
        % Measurement Update
            % Process Simulation Equation
            x = A*x + B*u_k + sqrt(Q) * randn; % Unused
            % Measurement Simulation Equation
            z_k = double(readProximity(myirsensor)); % +
sqrt(R) * randn;
            % Computation of K
            K_k = P_minus_k * H /(H * P_minus_k * H + R);
            % State Estimate Correction
            x_hat_k = x_hat_minus_k + K_k * (z_k - H *
x_hat_minus_k);
            % Error Covariance Update
            P_k = (1 - K_k * H) * P_minus_k;
        % Variables for the next cycle
        P_k_minus_1 = P_k;
        x_hat_k_minus_1 = x_hat_k;
        % Array Recording
        x_list = [x_list; x];
        x_hat_list = [x_hat_list; x_hat_k];
        z_k_list = [z_k_list; z_k];
        K_k_list = [K_k_list; K_k];
        P_k_list = [P_k_list; P_k];

        i = i+1; i_a = [i_a; i];
        pause(deltaT);
    end
```

```
    % Backward Motion
    t = timer('TimerFcn', 'stat=false', 'StartDelay',
EXE_TIME);
    start(t); stat = true;
    Rightmotor.Speed = -speed;
    Leftmotor.Speed = -speed;
    while stat == true && ~readButton(mylego, 'up')
        % Prediction Update
        x_hat_minus_k = A * x_hat_k_minus_1 + B * u_k;
        P_minus_k = A * P_k_minus_1 * A + Q;
        % Measurement Update
            % Process Simulation Equation
            x = A*x + B*u_k + sqrt(Q) * randn; % Unused
            % Measurement Simulation Equation
            z_k = double(readProximity(myirsensor)); % +
sqrt(R) * randn;
            % Computation of K
            K_k = P_minus_k * H /(H * P_minus_k * H + R);
            % State Estimate Correction
            x_hat_k = x_hat_minus_k + K_k * (z_k - H *
x_hat_minus_k);
            % Error Covariance Update
            P_k = (1 - K_k * H) * P_minus_k;
        % Variables for the next cycle
        P_k_minus_1 = P_k;
        x_hat_k_minus_1 = x_hat_k;
        % Array Recording
        x_list = [x_list; x];
        x_hat_list = [x_hat_list; x_hat_k];
        z_k_list = [z_k_list; z_k];
        K_k_list = [K_k_list; K_k];
        P_k_list = [P_k_list; P_k];

        i = i+1; i_a = [i_a; i];
        pause(deltaT);
    end
end
stop(Rightmotor);
stop(Leftmotor);
```

```
figure %2
plot(i_a, x_hat_list)
xlabel('Time Index')
ylabel('Distance Estimate')

figure %3
plot(i_a, z_k_list)
xlabel('Time Index')
ylabel('Distance Measurement')

figure %4
plot(i_a, x_hat_list, 'r')
hold
plot(i_a, z_k_list, 'k')
legend('Estimate', 'Measurement')
xlabel('Time Index')
ylabel('Distance')

figure %5
plot(i_a, K_k_list, 'b')
xlabel('Time Index')
ylabel('Gain')

figure %6
plot(i_a, P_k_list, 'r')
xlabel('Time Index')
ylabel('Covariance')
```