ME C231B Final Project
Programming Assignment

Solved in MATLAB

Sumanyu Singh and Sharnam Shah

May 4th, 2020

1 Introduction

The task for this programming assignment is to develop a state estimator to estimate the position and heading of a bicycle. The estimator error should be minimal while being computationally inexpensive. The dynamic model, measurement datasets, and uncertainties in the system have been provided. We have tested an EKF and a UKF as state estimators in this report. Noise has been taken into account and the calibration has also been done while using these estimators. Design choices, important considerations, key equations and the results have also been shown

2 Modelling the Bicycle Dynamics

To develop a good state estimator, we need to create a good enough process and measurement model and account for the noise in both the systems. The following equations were provided in the assignment and serve as the base for our modelling.

Given 1. *The bicycle dynamics in characterized in continuous time by the following equations:*

$$\dot{x}_1(t) = v(t)\ cos(\theta(t)) \tag{1}$$
$$\dot{y}_1(t) = v(t)\ sin(\theta(t)) \tag{2}$$
$$\dot{\theta}(t) = \frac{v(t)}{B}\ tan(\gamma(t)) \tag{3}$$

where, $(x_1(t), y_1(t))$ *is the position of the rear wheel,* $v(t)$ *is the linear velocity of the bicycle,* $\gamma(t)$ *is the steering angle,* $\theta(t)$ *is the heading, and* B *is the wheel base.*

Given 2. *The idealized model of the position measurement with added noise is given by the following equation:*

$$p(k) = \begin{bmatrix} x_1(t_k) + \frac{1}{2}B\ cos(\theta(t_k)) + w_1 \\ y_1(t_k) + \frac{1}{2}B\ sin(\theta(t_k)) + w_2 \end{bmatrix} \tag{4}$$
$$h_k = p(k) \tag{5}$$

*where, $p(k)$ is the position measurement collected at discrete time t_k. h_k serves as our **measurement model** in our state estimator.*

Given 3. *The additional information required to create the estimator is provided as follows:*

$$(x(0), y(0)) \approx (0, 0) \tag{6}$$

$$\theta(0) \approx \frac{\pi}{4} \tag{7}$$

$$B = 0.8 \pm 10\% \tag{8}$$

$$r = 0.425 \pm 5\% \tag{9}$$

*where, r is the tire radius. The values of r and B are the manufacturer's nominal values with the approximate error. These equations help us model our **uncertainty**.*

Since our process model is in continuous time while our measurement model is in discrete time, the first step to model these equations is converting our process model into a discrete representation to implement the model in MATLAB. The result of discretization and adding noise is:

$$x(k+1) = x(k) + (v(k)\ cos(\theta(k)))\delta t + v_1 \tag{10}$$

$$y(k+1) = y(k) + (v(k)\ sin(\theta(k)))\delta t + v_2 \tag{11}$$

$$\theta(k+1) = \theta(k) + (\frac{v(k)}{B}\ tan(\gamma(k)))\delta t + v_3 \tag{12}$$

$$q_k = \begin{bmatrix} x(k) & y(k) & \theta(k) \end{bmatrix}' \tag{13}$$

where k represents time and δt represents the time step. Using these equations, we now have our **process model in a discrete time** represented by q_k.

Our **final model** can now be represented in the following format:

$$x(k) = q_{k-1}(x(k-1), u(k-1), v(k-1)) \tag{14}$$

$$z(k) = h_k(x(k), w(k)) \tag{15}$$

where $x(k)$ is our process model shown in equation 13. and $z(k)$ is our measurement model shown in equation 5. $v(k-1)$ is the process noise, $w(k)$ is the measurement noise.

The values for noise are not provided in the assignment and thus constitute as a design decision.

3 Design Decision - Reasoning

We first observed the model of the bicycle and inferred our initial design decisions from it. We saw that the it is non linear. Three quantities, namely x,y and θ are being used to get the next set of values. We also sought to obtain different noise values and iterate over them to get better results.

Filter and State Choice:

As three quantities are being used in the dynamic equations we chose our state X = [x, y, θ]'. From the methods we have learnt, We had a choice between Kalman filter, Extended Kalman Filter, Unscented Kalman Filter and Particle Filter.As the dynamics model is non-linear and the resultant code needed to execute within 30 seconds, we chose to implement EKF and UKF. Being

computationally very expensive, PF gets eliminated. The expected close margins of performance between UKF and EKF prompted us to test both of these methods. To understand which estimator gives a better performance between EKF and UKF we tested both and compared their results.

EKF

The EKF was implement as taught in class according to lecture 10. The reasoning behind this was its ability to linearize non-linear components of our equation and utilize the standard Kalman Filter. The matrices A, L, H, M were calculated from our model to be the following:

$$A := \begin{bmatrix} 1 & 0 & -v(k-1)\ sin(\theta(k-1))\delta t \\ 0 & 1 & v(k-1)\ cos(\theta(k-1))\delta t \\ 0 & 0 & 1 \end{bmatrix} \tag{16}$$

$$L := I \tag{17}$$

A and L are matrices used in Prediction Update

$$H := \begin{bmatrix} 1 & 0 & -\frac{1}{2}B\ sin(\theta(t_k)) \\ 0 & 1 & \frac{1}{2}B\ cos(\theta(t_k)) \end{bmatrix} \tag{18}$$

$$M := I \tag{19}$$

H and M are matrices used in Measurement Update

where, I is the Identity Matrix.

In order to choose the best possible V matrix for EKF, we decided to consider the percentage error in equations 8 and 9 to be the standard deviation. The resultant process noise matrix was calculated based on which state the error was affected and tuned for accuracy. It is shown below:

$$Var[r] = 0.02125^2 \tag{20}$$

$$Var[\frac{r}{B}] = 0.07968^2 \tag{21}$$

$$V := \delta t \begin{bmatrix} Var[r] & 0 & 0 \\ 0 & Var[r] & 0 \\ 0 & 0 & Var[r] + Var[\frac{r}{B}] \end{bmatrix} \tag{22}$$

The measurement noise (W) was calculated from run0 calibration files and is explained in detail in the following UKF section as it is the same value used in EKF as well.

Other design choices for EKF resulted in the following noise values after significant tuning to get the best result:

$$P_m(0) = 0.001\ I \tag{23}$$

$$E[x_m(0)] = 0 \tag{24}$$

$$E[v(k-1)] = 0 \tag{25}$$

$$E[w(k)] = 0 \tag{26}$$

UKF

A UKF with additive linear noise model was chosen. It seemed to be a reasonable assumption to choose a simple possibility if linear process noise over other options as there was minimal information. Other than this, the general UKF equations with sigma points was implemented

Noise estimation choices

The measurement noise W was chosen to be calculated from run0 shown in A.1 as this was indicated as a calibration run when the bike is stationary in the assignment problem. The P matrix was initialized by selecting a general value. For process noise two approaches were followed. In one, it was modelled using the uncertainties in r,B and the dynamics equations. The other approach was to iteratively choose different values of process noise to optimize the error. Although it seems like the process noise of the position depends on that of theta, hence the noise matrix has non zero diagnal terms, we assumed a diagnal process noise matrix for simplicity.

Estimates of noise matrices

$$\triangle r = 0.05 \tag{27}$$
$$\triangle B = 0.1 \tag{28}$$
$$\triangle v = 5\ \omega \triangle r \tag{29}$$
$$\triangle \theta = \frac{\triangle v}{B} tan(\gamma) - v\ tan(\gamma) \frac{\triangle B}{B^2} \tag{30}$$
$$\triangle x = \triangle v\ cos(\theta) - v \triangle \theta\ sin(\theta) \tag{31}$$
$$\triangle y = \triangle v\ sin(\theta) + v \triangle \theta\ cos(\theta) \tag{32}$$

these equations result in the process noise shown below.

$$V = \begin{bmatrix} \triangle x^2 & \triangle x \triangle y & \triangle x \triangle \theta \\ \triangle y \triangle x & \triangle y^2 & \triangle y \triangle \theta \\ \triangle \theta \triangle x & \triangle \theta \triangle y & \triangle \theta^2 \end{bmatrix} \tag{33}$$

where $\triangle$ represents error.

The measurement noise for both EKF and UKF is calculated to be the following:

$$W = \begin{bmatrix} 1.0893 & 1.5333 \\ 1.5333 & 2.9880 \end{bmatrix} \tag{34}$$

Other design choices for UKF resulted in the following noise values after significant tuning to get the best result:

$$P_m(0) = 0.001\ I \tag{35}$$
$$E[x_m(0)] = 0 \tag{36}$$
$$E[v(k-1)] = 0 \tag{37}$$
$$E[w(k)] = 0 \tag{38}$$

4 Estimator Performance - Discussion and Evaluation

EKF - (Code shown in A.4)

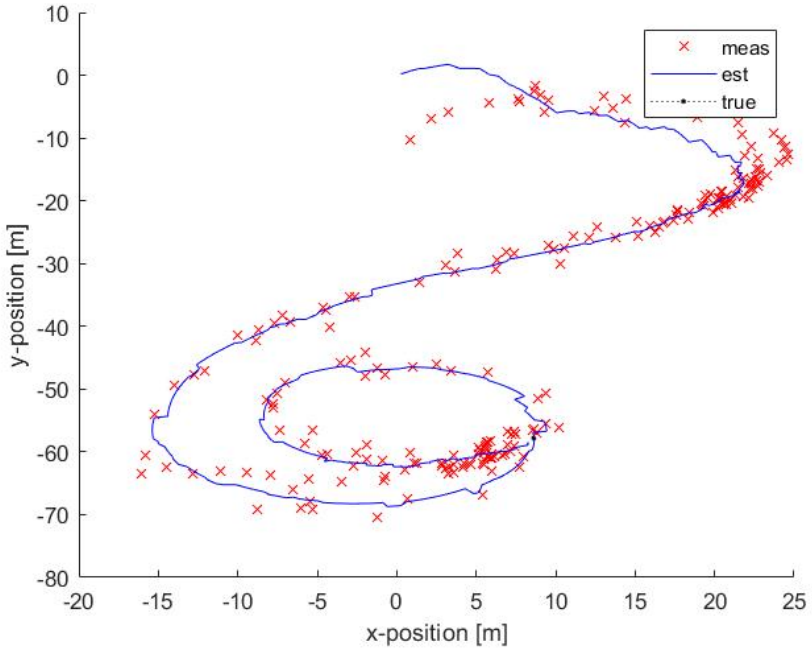

Figure 1: EKF tracking measured and expected run 1

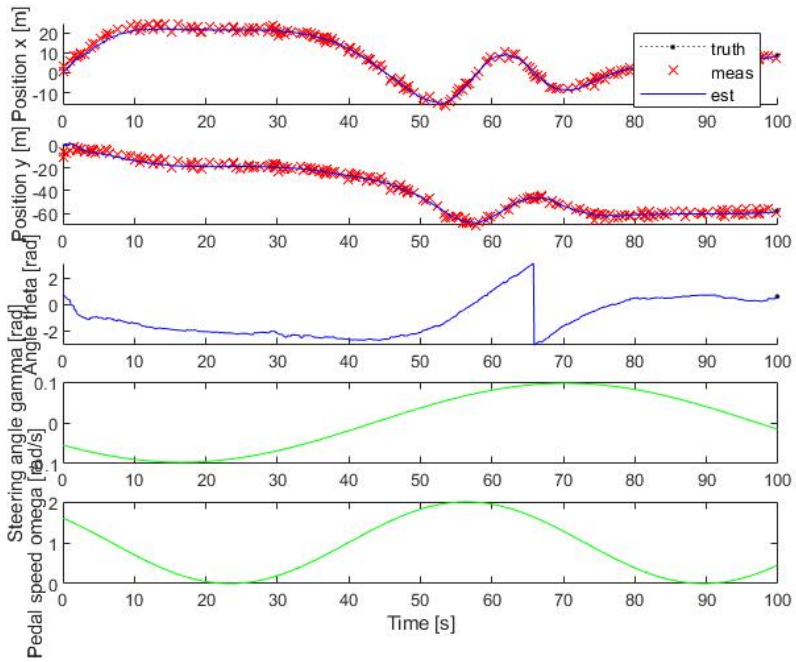

Figure 2: EKF tracking different quantities run 1

The EKF gave the following error values for run 1

$$pos\ x = -0.32128\ m$$
$$pos\ y = -0.68106\ m$$
$$angle = -0.0172\ rad$$

UKF - (Code shown in A.2)

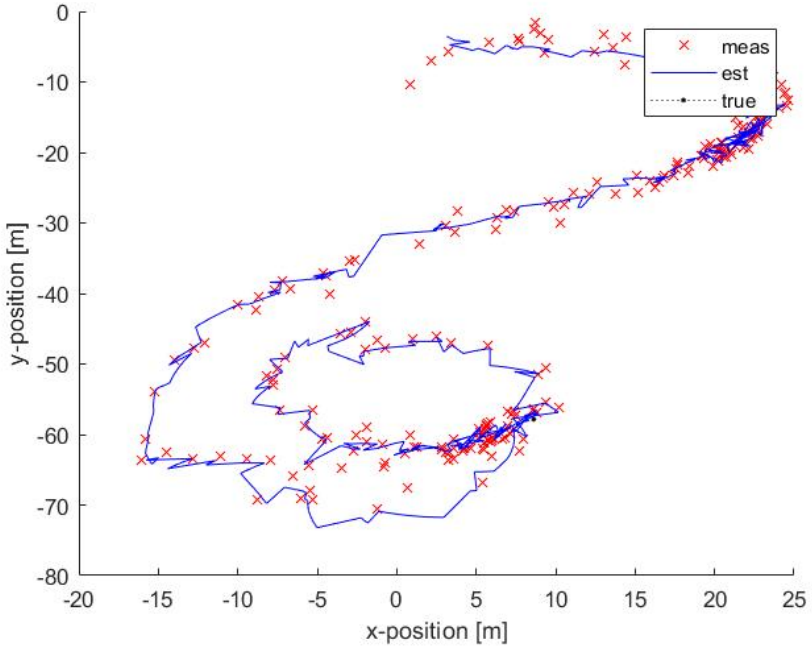

Figure 3: UKF tracking measured and expected run 1

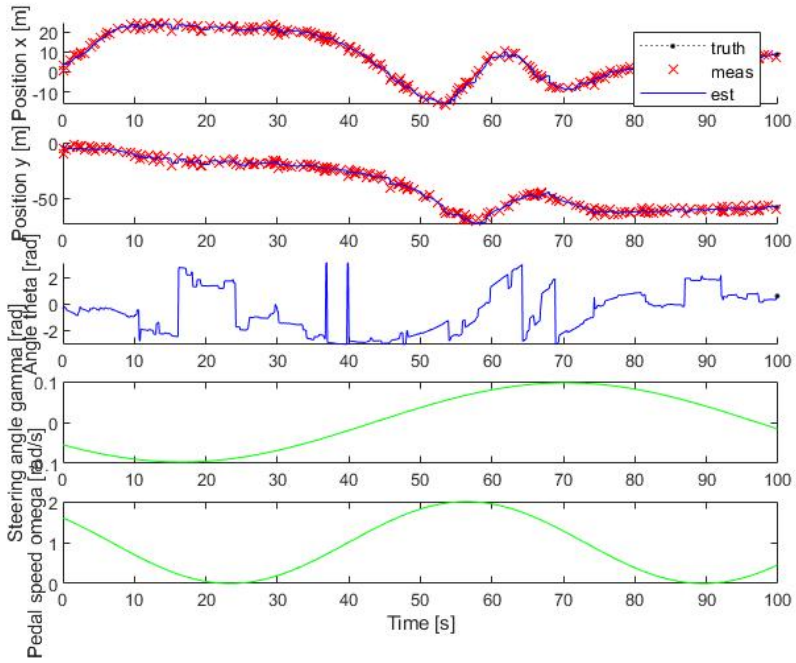

Figure 4: UKF tracking different quantities run 1

The result for run 1 is obtained after using the below optimal value of V

$$V = \begin{bmatrix} 0.002 & 0 & 0 \\ 0 & 0.002 & 0 \\ 0 & 0 & 0.002 \end{bmatrix} \tag{39}$$

The UKF gave the following error values for run 1

$$pos\ x = -1.1046\ m$$
$$pos\ y = -0.80397\ m$$
$$angle = -0.28908\ rad$$

While we are comparing the results of run 1 in this report, we concluded after multiple experimental runs from the data files that our EKF model outperforms our UKF model. This is why we chose to submit our final code from EKF model. The following table shows the results from our EKF code from run 1 to 5:

EKF Results			
Run #	x [m]	y [m]	θ [rad]
1	-0.32128	-0.68106	-0.0172
2	0.0067402	0.37486	0.18439
3	0.07269	0.60745	0.11847
4	0.030378	0.78311	-0.17363
5	-0.45988	-1.341	-0.16563

5 Conclusion

In this report, we described the problem assigned to us and our approach in designing a state estimator to track the position and heading of a bicycle as it moves. We first modelled our bicycle system to represent it in a format on which we can readily apply techniques learned in class. Then, using the course material, we designed an Extended Kalman Filter and an Unscented Kalman Filter to evaluate performance for the provided data sets. Observing that the Extended Kalman Filter performed better after tuned values of our design variables, we finalized on using an Extended Kalman Filter as it tracked the position and heading with minimal error with quick computation when compared to our UKF state estimator.

A Appendices

A.1 *Code to Calculate W*

```
clear all
%%Code to calculate the Measurement variance for part 0 as the set 0 is
%%meant for calibration
experimentalRun = 0;
fprintf(['Loading the data file #' num2str(experimentalRun) ' \n']);
filename = ['data/run_' num2str(experimentalRun,'%03d') '.csv'];
experimentalData = csvread(filename);
numDataPoints = size(experimentalData,1);
count = 0;
N = 0;

for i = 1:numDataPoints
    if ~isnan(experimentalData(i,4)) & ~isnan(experimentalData(i,5))
        % have a valid measurement
        N=N+1;
    end
end

wx = zeros(N,1)
wy = zeros(N,1);
theta = 0;
ux = 0;
uy = 0;

for i = 1:numDataPoints
    if ~isnan(experimentalData(i,4)) & ~isnan(experimentalData(i,5))
        ux = experimentalData(i,4)+ux;
        uy = experimentalData(i,5)+uy;
    end
end
ux = ux/N;
uy = uy/N;

for i = 1:numDataPoints
    if ~isnan(experimentalData(i,4)) & ~isnan(experimentalData(i,5))
        % have a valid measurement
        count=count+1;
        wx(count) = experimentalData(i,4)-ux;
        wy(count) = experimentalData(i,5)-uy;
    end
end

fprintf('Mean of wx')
mean(wx)
fprintf('Mean of wy')
mean(wy)
```

```
fprintf('Variance of wxx')
var(wx)
fprintf('Variance of wyy')
var(wy)
fprintf('Covariance matrix')
cov(wx,wy)
```

A.2 *UKF Code - estRun*

```
function [x,y,theta,internalStateOut] = estRun(time, dt,...
    internalStateIn, steeringAngle, pedalSpeed, measurement)

x = internalStateIn.x;
y = internalStateIn.y;
theta = internalStateIn.theta;
Pp = internalStateIn.P;

%%%%%%%%%%%%%%%%% UKF %%%%%%%%%%%%%%%%%%%%%%
% Variables for modelling
r = 0.425;
B = 0.8;
v = 5*r*pedalSpeed;
s = zeros(3,6);

%% Noise settings
W = [1.0893,1.5333;1.5333,2.9880];
%%Defining variables
T = chol(Pp);
X = [x;y;theta];
%V = zeros(3,3);
sigmaV = 0.1;
Xm = zeros(3,1);
Pm = zeros(3,1);
delB = 0.1;
delr = 0.05;
delv = 5*pedalSpeed*delr;
deltheta = (delv*tan(steeringAngle)/B)-(v*tan(steeringAngle)*delB/(B*B));
delx = (delv*cos(theta))-(v*sin(theta)*deltheta);
dely = (delv*sin(theta))+(v*cos(theta)*deltheta);
vn = [delx;dely;deltheta];
%%Using different V matrices
%V = [delx*delx, delx*dely, delx*deltheta;
%    dely*delx, dely*dely, dely*deltheta;
%    delx*deltheta, deltheta*dely, deltheta*deltheta];
%V = [delx,0,0;0,dely,0;0,0,deltheta];
%vn
%V = zeros(3)
%sigmaV = 0.02125^2;
%sigmaB = 0.07968^2;
%V =dt*[sigmaV, 0, 0;
%    0, sigmaV, 0;
```

```
%     0,  0,  sigmaV + sigmaB];

b = 0.002;
V = [b,0,0;0,b,0;0,0,b];
%V = [delx,0,0;0,dely,0;0,0,deltheta];
%V = zeros(3);

%%Initializing Sigma Points
s1 = X+(sqrt(3)*T(:,1));
s4 = X-(sqrt(3)*T(:,1));
s2 = X+(sqrt(3)*T(:,2));
s5 = X-(sqrt(3)*T(:,2));
s3 = X+(sqrt(3)*T(:,3));
s6 = X-(sqrt(3)*T(:,3));
s = [s1,s2,s3,s4,s5,s6];
sx = zeros(6,1);
sy = zeros(6,1);
stheta = zeros(6,1);

%Sending the points through the model
for i=1:6
    sx(i) = s(1,i) + (v*cos(theta))*dt;
    sy(i) = s(2,i) + (v*sin(theta))*dt;
    stheta(i) = s(3,i) + ((v*tan(steeringAngle))*dt/B);
end

usx = mean(sx);
usy = mean(sy);
utheta = mean(stheta);
usm = [usx;usy;utheta];
%sVr = [var(sx);var(sy);var(stheta)];
sus = [sx,sy,stheta];

for j = 1:6
    Pp = Pp + ((sus(j,:)')-usm)*transpose((sus(j,:)')-usm);
end
Pp = Pp/6;
Pp = Pp+V;

xp = usx;
yp = usy;
thetap = utheta;
szx = zeros(6,1);
szy = zeros(6,1);
sz = zeros(6,2);
Xp = [xp;yp;thetap];

%%Checking for measurement and if measurement is available then updating
if ~isnan(measurement(1)) & ~isnan(measurement(2))
    measurex = measurement(1);
    measurey = measurement(2);
```

```
    % Measurement passing sigma points through the z model
    for i = 1:1:6
        szx(i)= sx(i)+ 0.5*B*cos(stheta(i));
        szy(i) = sy(i) + 0.5*B*sin(stheta(i));
    end
    sz = [szx,szy];
    zex = [mean(szx);mean(szy)];
    Pzz = zeros(2,2);
    Pxz = zeros(3,2);
    Pxs = zeros(3,2);
    Pzs = zeros(2,2);

    % Calculating Pzz
    for j = 1:6
        Pzz = Pzs + (((sz(j,:)')-zex)*transpose(((sz(j,:)')-zex)));
        Pzs = Pzz;
        Pxz = Pxz + ((([sx(j);sy(j);stheta(j)])-Xp)*transpose(((sz(j,:)')-zex)));
        Pxs = Pxz;
    end
    Pzz = ((1/6)*(Pzz))+W;
    Pxz = ((1/6)*Pxz);
    K = zeros(3,2);
    K = Pxz*inv(Pzz);
    Z = zeros(2,1);
    Z = [measurex;measurey];
    Xm = Xp+(K*(Z-zex));
    Pm = Pp - (K*Pzz*(transpose(K)));
else
    Pm = Pp;
    Xm = Xp;
end

x = Xm(1);
y = Xm(2);
theta = Xm(3);


%% OUTPUTS %%
% Update the internal state (will be passed as an argument to the function
% at next run), must obviously be compatible with the format of
internalStateOut.x = x;
internalStateOut.y = y;
internalStateOut.theta = theta;
internalStateOut.P = Pm;

end
```

A.3 *Submitted Code: EKF - estInitialize*

```
function [internalState, studentNames, estimatorType] = estInitialize
```

```
% Fill in whatever initialization you'd like here. This function
% generates the internal state of the estimator at time 0. You may do
% whatever you like here, but you must return something that is in the
% format as may be used by your run() function as the first
% returned variable.
%
% The second returned variable must be a list of student names.
%
% The third return variable must be a string with the estimator type

% we make the internal state a structure, with the first three elements the
% positions x, y; the angle theta; and our favorite colour.

% note that there is *absolutely no prescribed format* for this internal
% state.
% You can put in it whatever you like. Probably, you'll want to keep the
% position and angle, and probably you'll remove the color.

%%%%%%%%%%%%%%%%%% Assumed initialization values %%%%%%%%%%%%%%%%%%%%%%%%%%
internalState.x = 0;
internalState.y = 0;
internalState.theta = pi/4;
internalState.P = 0.001*eye(3);

% replace these names with yours.
% Delete the second name if you are working alone.
studentNames = ['Sumanyu Singh','\tSharnam Shah']

% replace this with the estimator type. Use one of the following options:
%  'EKF' for Extended Kalman Filter
%  'UKF' for Unscented Kalman Filter
%  'PF' for Particle Filter
%  'OTHER: XXX' if you're using something else, in which case please
%                replace "XXX" with a (very short) description
estimatorType = 'EKF'

end
```

A.4 *Submitted Code: EKF - estRun*

```
function [x,y,theta,internalStateOut] = estRun(time,...
    dt, internalStateIn, steeringAngle, pedalSpeed, measurement)

% In this function you implement your estimator. The function arguments
% are:
%  time: current time in [s]
%  dt: current time step [s]
%  internalStateIn: the estimator internal state, definition up to you.
%  steeringAngle: the steering angle of the bike, gamma, [rad]
%  pedalSpeed: the rotational speed of the pedal, omega, [rad/s]
%  measurement: the position measurement valid at the current time step
```

```
%
% Note: the measurement is a 2D vector, of x-y position measurement.
%  The measurement sensor may fail to return data, in which case the
%  measurement is given as NaN (not a number).
%
% The function has four outputs:
%  est_x: your current best estimate for the bicycle's x-position
%  est_y: your current best estimate for the bicycle's y-position
%  est_theta: your current best estimate for the bicycle's rotation theta
%  internalState: the estimator's internal state, in a format that can...
% be understood by the next call to this function

% Assigning variables from previous time step
x = internalStateIn.x;
y = internalStateIn.y;
theta = internalStateIn.theta;

%%%%%%%%%%%%%%%%%% EKF %%%%%%%%%%%%%%%%%%%%%%%%

% Variables for modelling
r = 0.425;
B = 0.8;
v = 5*r*pedalSpeed;

%%%%%%%%%%%%%%%%%% Noise %%%%%%%%%%%%%%%%%%%%%%%

% xm(0)
Pm = internalStateIn.P; %This value is intialized in estInitialize

% Process Noise [E[v] = 0]
varR = 0.02125^2;
varB = 0.07968^2;
V = dt*[varR, 0, 0;
     0, varR, 0;
     0, 0, varR + varB];


% Measurement Noise [E[w] = 0]
W =[1.0893, 1.5333;
     1.5333, 2.9880];

%%%%%%%%%%%%%% Prediction Update %%%%%%%%%%%

% Prediction Model is not needed as A and L are calculated by hand.
% Linearized Matrices
A =[1, 0, -(v*sin(theta))*dt;
     0, 1, (v*cos(theta))*dt;
     0, 0, 1];

L = eye(3);
```

```
% Prediction Step
xp = x + (v*cos(theta))*dt;
yp = y + (v*sin(theta))*dt;
thetap = theta + (v*tan(steeringAngle))*dt/B;
Xp = [xp; yp; thetap];

Pp = A*Pm*A' + L*V*L';

%%%%%%%%%%%% Measurement Update %%%%%%%%%%

% Measurement Model
h =[xp + 0.5*B*cos(thetap);
    yp + 0.5*B*sin(thetap)];

% Linearized Matrices
H =[1, 0, -0.5*B*sin(thetap);
    0, 1, 0.5*B*cos(thetap)];

M = eye(2);

% Measurement Step
K = Pp*H'*inv(H*Pp*H' + M*W*M'); %Gain

if ~isnan(measurement)
    xm = Xp + K*(measurement' - h);
    %       Pm = (eye(3) - K*H)*Pp; %This method is less accurate
    Pm = (eye(3) - K*H)*Pp*(eye(3) - K*H)' + K*W*K';
else
    xm = Xp;
    Pm = Pp;
end

% Storing new values before returning the function
x = xm(1);
y = xm(2);
theta = xm(3);


%% OUTPUTS %%
% Update the internal state (will be passed as an argument to the function
% at next run), must obviously be compatible with the format of
% internalStateIn:

internalStateOut.x = x;
internalStateOut.y = y;
internalStateOut.theta = theta;
internalStateOut.P = Pm;

end
```