

State estimation of a Quadrotor

Ramprasad Rajagopalan
MS in Mechanical Engineering
Colorado School of Mines
Golden, USA
rrajagopalan1@mymail.mines.edu

Abstract—The project deals with the state estimation of a X-configuration quadrotor using Extended Kalman Filter (EKF) and Unscented Kalman Filter (UKF) based on Gyroscopes and Global Positioning System (GPS) measurements. The quadrotor system was modelled in Simulink and the filters were programmed in MATLAB. The sensors were added with measurement noise and process noise. The filter model was tested for two different operating conditions and the robustness of each was evaluated. The results shows that the performance of EKF in estimating the staes of the quadrotor is comparatively better than UKF for the given filter model.

Index Terms—Extended Kalman filter, Unscented Kalman filter, Quadrotor, state estimation

I. INTRODUCTION

The system chosen is an X configuration quadrotor with the co-ordinates frames and labels given in Fig.1. A quadrotor has 4 rotors controlled by a flight controller and mounted on 4 arms. 2 rotors on diagonally opposite side rotates in clockwise direction while 2 other rotors rotate in anticlockwise direction. Fig.1 shows the body frame x - axis x_B and y - axis y_B and inertial frame x - axis $x_{Inertial}$ and y - axis $y_{Inertial}$. The

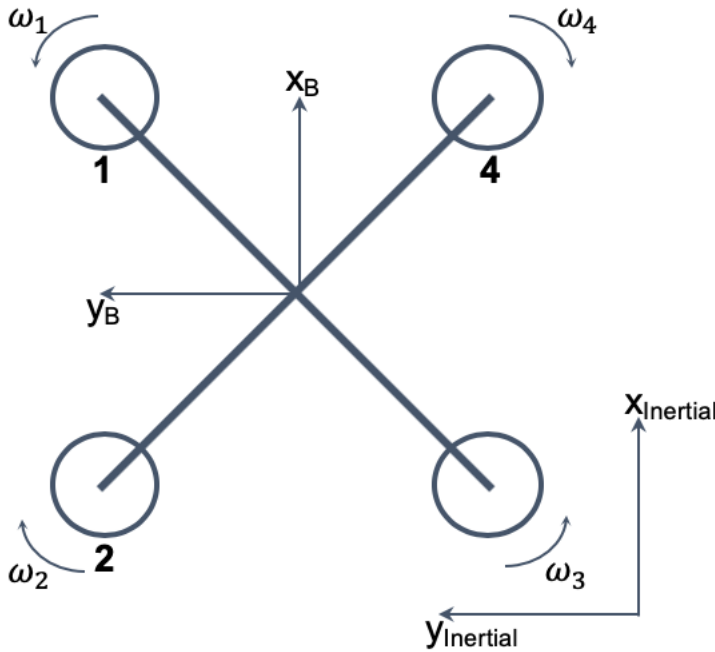


Fig. 1. Quadrotor co-ordinate frame

forward direction of the quadrotor is x_B and roll about x_B is given by ϕ . The pitch θ is rotation about y_B and yaw ψ is rotation about z_B . Each rotating propeller produces an upward force given by,

$$F_i = k_f \omega_i^2 \quad (1)$$

where F_i is the thrust generated by each propeller, k_f is the thrust factor and ω_i is the anuglar velocity of each propeller. Apart from upward thrust, a rotating propeller also produces an opposing moment given by,

$$M_i = k_m \omega_i^2 \quad (2)$$

where M_i is the moment generated by each propeller about z_B and k_m is the drag factor. The dynamic equations of the system is derived as follows. The total thrust from the 4 rotating propellers is given by,

$$F_{total} = k_f (\omega_1^2 + \omega_2^2 + \omega_3^2 + \omega_4^2) \quad (3)$$

The linear acceleration equations are obtained as,

$$\begin{bmatrix} \ddot{x} \\ \ddot{y} \\ \ddot{z} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ -g \end{bmatrix} + \frac{R}{mass} \begin{bmatrix} 0 \\ 0 \\ F_{total} \end{bmatrix} \quad (4)$$

where $R = R_z(\psi)R_y(\theta)R_x(\phi)$ is the rotation matrix from world frame to body frame obtained from corresponding rotation along the body axes. The moments along the respective axes are given by,

$$\tau_x = \frac{Lk_f}{\sqrt{2}} (\omega_1^2 + \omega_2^2 - \omega_3^2 - \omega_4^2) \quad (5)$$

$$\tau_y = \frac{Lk_f}{\sqrt{2}} (-\omega_1^2 + \omega_2^2 + \omega_3^2 - \omega_4^2) \quad (6)$$

$$\tau_z = k_m (\omega_1^2 - \omega_2^2 + \omega_3^2 - \omega_4^2) \quad (7)$$

The body frame angular accelerations are given by the Euler's equation for rigid body dynamics,

$$\begin{bmatrix} \ddot{\phi} \\ \ddot{\theta} \\ \ddot{\psi} \end{bmatrix} = I^{-1} \left(\begin{bmatrix} \tau_x \\ \tau_y \\ \tau_z \end{bmatrix} - \omega \times I \omega \right) \quad (8)$$

where $I = \begin{bmatrix} I_{xx} & 0 & 0 \\ 0 & I_{yy} & 0 \\ 0 & 0 & I_{zz} \end{bmatrix}$ and $\omega = \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix}$ in the body frame. Therefore, the equations of motion from the above relations are written as follow,

$$\ddot{x} = \frac{F_{total}(\cos(\phi)\sin(\theta)\cos(\psi) + \sin(\phi)\sin(\psi))}{mass} \quad (9)$$

$$\ddot{y} = \frac{F_{total}(\cos(\phi)\sin(\theta)\sin(\psi) - \sin(\phi)\cos(\psi))}{mass} \quad (10)$$

$$\ddot{z} = \frac{F_{total}(\cos(\phi)\cos(\theta))}{mass} - 9.81 \quad (11)$$

$$\ddot{\phi} = \frac{\tau_x + (I_{yy} - I_{zz})\dot{\theta}\dot{\psi}}{I_{xx}} \quad (12)$$

$$\ddot{\theta} = \frac{\tau_y + (I_{zz} - I_{xx})\dot{\phi}\dot{\psi}}{I_{yy}} \quad (13)$$

$$\ddot{\psi} = \frac{\tau_z + (I_{xx} - I_{yy})\dot{\theta}\dot{\phi}}{I_{zz}} \quad (14)$$

Based on the above relations, one can derive the continuous states of the quadrotor given an input angular velocity of the propeller or the thrust and moments about correspondding axes. The state of the quadrotor i.e., position and orientation can be obtained through these relation. Although the former can be measured using a laser tracking or GPS in an environment, the latter cannot be physically measured but rather needs to be estimated. It is given that the laser tracking or GPS measurement includes noise and it is not the absolute measurement.

The objective of this study is to estimate position of the quadrotor by filtering the noise in GPS measurement and also to estimate the orientation value which cannot be directly measured. The quadrotor is modelled in Simulink and simulated for 2.5s with time step, T_s , 0.1s.

II. METHODOLOGY

A. State-space model of the system

The continuous time state-space model of the quadrotor is obtained by by defining the state vector of the system, X , as follows,

$$X = [x \ y \ z \ \phi \ \theta \ \psi \ \dot{x} \ \dot{y} \ \dot{z} \ \dot{\phi} \ \dot{\theta} \ \dot{\psi}]$$

$$u = [F_{total} \ \tau_x \ \tau_y \ \tau_z]$$

where u is the input vector. Then, the continuous time state-space model can be written as,

$$\dot{X} = f_x(X, u) + w$$

where $f_x(X, u)$ is the non-linear function matrix mapping the first order derivative states to the system states and w is the process noise or disturbance in input. The non-linear function matrix, $f_x(X, u)$, can be obtained from Eqn.9-14.

The process noise or input disturbance, $w \sim \mathcal{N}(\bar{x}, \sigma_w)$, is assumed to be a Gaussian distribution with mean, $\bar{x} = 0$ and variance, $\sigma_w = 1e-6$. The system variables that are assumed as given in Table.I.

TABLE I
SYSTEM VARIABLES

Parameter	Value
Mass	0.25 kg
I_{xx}	0.03 kgm^2
I_{yy}	0.03 kgm^2
I_{zz}	0.03 kgm^2
L	0.1 m
k_f	0.1
k_m	0.1

B. Sensor model

The sensors chosen for measurement are GPS and Gyroscope. The GPS measures the position of the quadrotor in the world frame. The GPS sensor is modelled as follows,

$$y_k^{GPS} = r(kT_s) + n_k^{GPS}$$

where y_k^{GPS} is the GPS measurement, $r(kT_s)$ is the actual position of quadrotor in world frame and n_k^{GPS} is the iid measurement noise. n_k^{GPS} is assumed to be Gaussian distribution i.e., $n_k^{GPS} \sim \mathcal{N}(0, \sigma_{GPS})$. The variance σ_{GPS} is assumed to be 0.5. The Gyroscope measures the angular velocities in body frame and it is modelled as follows,

$$y_k^{Gyro} = \dot{\Omega}(kT_s) + n_k^{Gyro}$$

where y_k^{Gyro} is the gyroscope measurement, $\dot{\Omega}(kT_s)$ is the actual angular velocity of quadrotor in body frame and n_k^{Gyro} is the iid measurement noise. n_k^{Gyro} is assumed to be Gaussian distribution i.e., $n_k^{Gyro} \sim \mathcal{N}(0, \sigma_{Gyro})$. The variance σ_{Gyro} is assumed to be 0.1.

C. Extended Kalman Filter model

The EKF was chosen to filter the sensor measurements and estimate the states of the quadrotor. As the GPS measurement values had noise, the noise in the measurement is needed to be filtered out. The orientation of the quadcopter cannot be directly measured, so it is the unobservable state in our system. This is estimated by EKF based on the gyroscope measurements which in turn has noise that needs to be filtered out. The EKF model is given as follows,

$$\dot{X} = f_x(X, u) + w$$

$$Y = HX + v$$

where Y is the sensor measurement given by

$$H = \begin{bmatrix} x_{GPS} & y_{GPS} & z_{GPS} & \dot{\phi}_{gyro} & \dot{\theta}_{gyro} & \dot{\psi}_{gyro} \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

v is the measurement noise with distribution $v \sim \mathcal{N}\left(\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} \sigma_{GPS} & 0 \\ 0 & \sigma_{gyro} \end{bmatrix}\right)$ The update equations

of the EKF are given as follows,

TIME UPDATE:

$$\Phi_k = \nabla_x f(\hat{x}_k^{(+)}, u_k) = \frac{\partial f_x(\hat{x}_k^{(+)}, u_k)}{\partial x} T_s + I$$

$$\hat{x}_{k+1}^{(-)} = f(\hat{x}_k^{(+)}, u_k)$$

$$P_{k+1}^{(-)} = \Phi_k P_k^{(+)} \Phi_k^T + Q$$

MEASUREMENT UPDATE:

$$K_k = P_k^{(-)} H (H P_k^{(-)} H^T + R)^{-1}$$

$$\hat{x}_k^{(+)} = \hat{x}_k^{(-)} + K_k (Y_k - H g_x(\hat{x}_k^{(-)}, u_k))$$

$$P_k^{(+)} = (I - K_k H) P_k^{(-)}$$

where, $Q = \sigma_w * eye(12)$ and $R \sim \mathbb{R}^{6 \times 6}$. The initial estimate $X_0^{(+)}$ was assumed to be zeros. The assumption made for initial covariance, $P_k^{(+)} = eye(12)$. The implementation of EKF in MATLAB is given in Appendix.A.

D. Unscented Kalman Filter

The UKF was implemented to compare the performance of EKF in estimating the states of the quadrotor. The UKF model is straight forward in approach and steps to estimate the states using the sensor measurements and updates are as follows,

TIME UPDATE:

$$x^{(i)} = \hat{x}^{(+)} + \tilde{x}^{(i)}$$

$$x_{k+1}^{(i)} = f(x^{(i)}, u_k)$$

$$\hat{x}_{k+1}^{(-)} = \frac{1}{2n} \sum_{i=1}^{2n} x_{k+1}^{(i)}$$

$$P_{k+1}^{(-)} = \frac{1}{2n} \sum_{i=1}^{2n} (x_{k+1}^{(i)} - \hat{x}_{k+1}^{(-)}) (x_{k+1}^{(i)} - \hat{x}_{k+1}^{(-)})^T + Q$$

MEASUREMENT UPDATE:

$$\bar{Y} = \frac{1}{2n} \sum_{i=1}^{2n} Y^{(i)}$$

$$P_{xy} = \frac{1}{2n} \sum_{i=1}^{2n} (x_k^{(i)} - \hat{x}_k^{(-)}) (Y^{(i)} - \bar{Y})^T$$

$$P_y = \frac{1}{2n} \sum_{i=1}^{2n} (Y^{(i)} - \bar{Y}) (Y^{(i)} - \bar{Y})^T + R$$

$$K = P_{xy} P_y^{-1}$$

$$x_k^{(+)} = \hat{x}_k^{(-)} + K (Y_k - \bar{Y})$$

$$P_k^{(+)} = P_k^{(-)} - K P_{xy}^T$$

The filter model was implemented using the same initial condition and measurement noise model as EKF. The implementation of UKF in MATLAB is given in Appendix.B.

III. RESULTS

The Simulink quadrotor model was simulated to make it pitch forward and move in the forward direction while climbing up. In order to make the quadrotor move in such a fashion, the following values for angular velocity of the propeller were given as input,

$$\omega_1 = 2.8 \text{ rad/s} \quad \omega_4 = 2.8 \text{ rad/s}$$

$$\omega_2 = 3.2 \text{ rad/s} \quad \omega_3 = 3.2 \text{ rad/s}$$

The plots obtained in Fig.2 and Fig.3 shows the comparison between the z estimate obtained from EKF and UKF implementation. The plots obtained in Fig.4 and Fig.5 shows the comparison between the x estimate obtained from EKF and UKF implementation. The plots obtained in Fig.6 and Fig.7 shows the comparison between the y estimate obtained from EKF and UKF implementation.

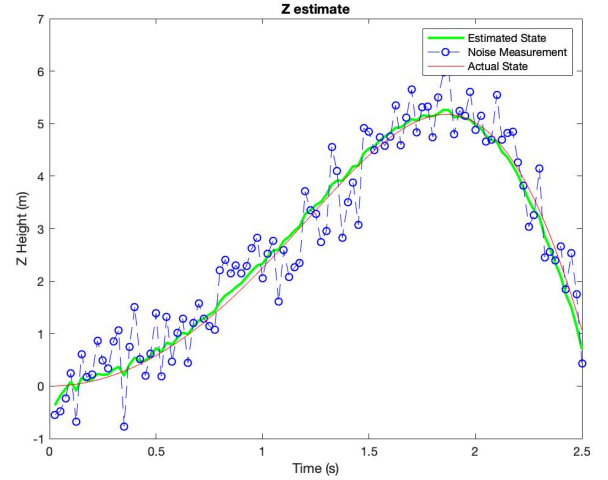


Fig. 2. z estimate using EKF

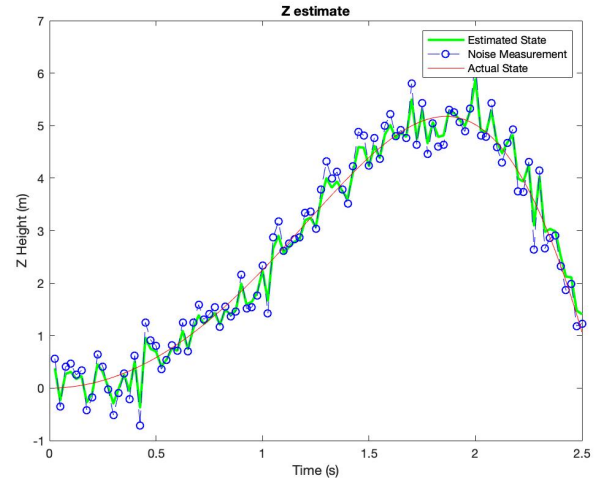


Fig. 3. z estimate using UKF

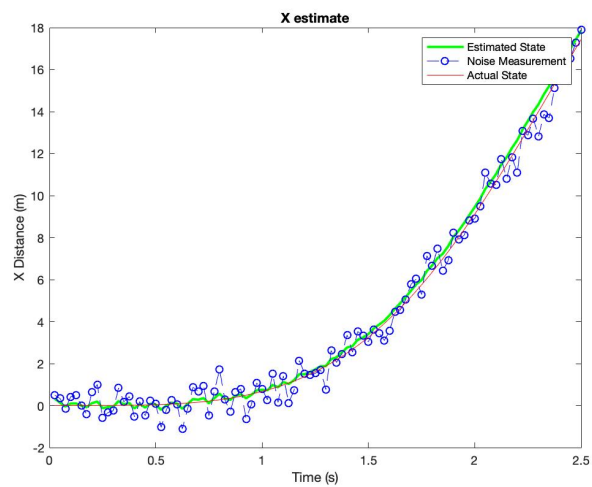


Fig. 4. x estimate using EKF

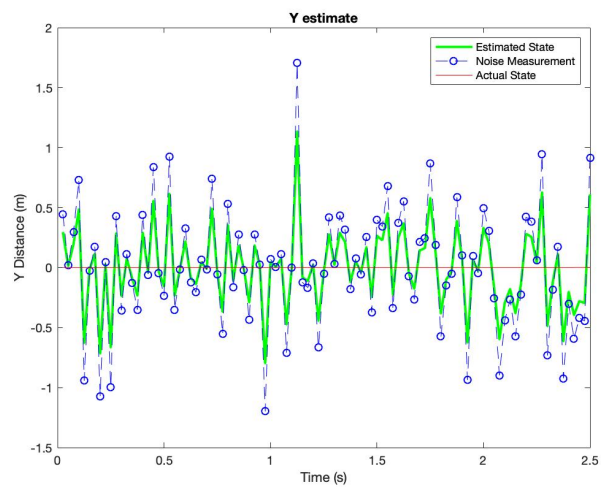


Fig. 7. y estimate using UKF

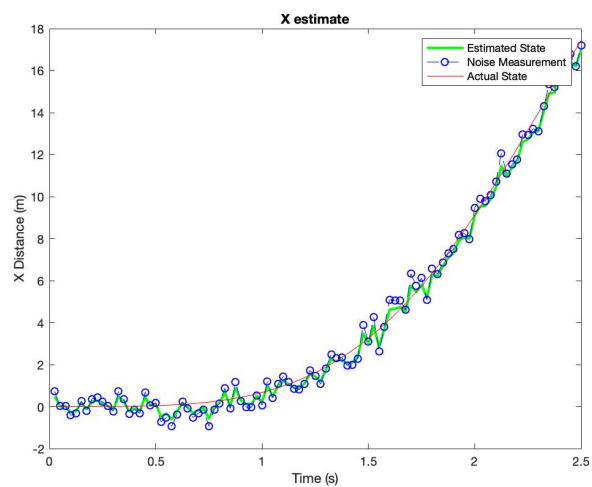


Fig. 5. x estimate using UKF

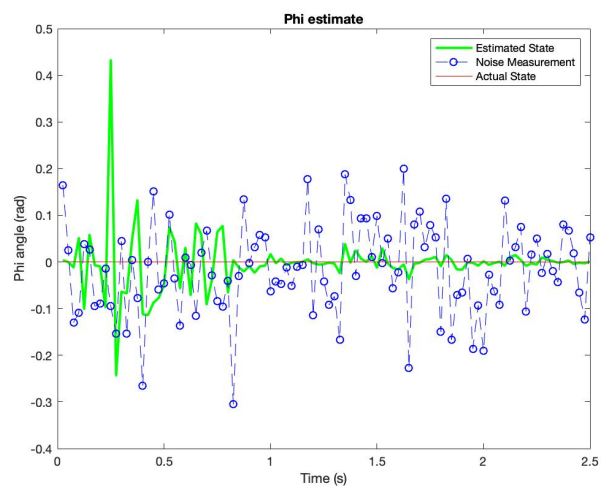


Fig. 8. ϕ estimate using EKF

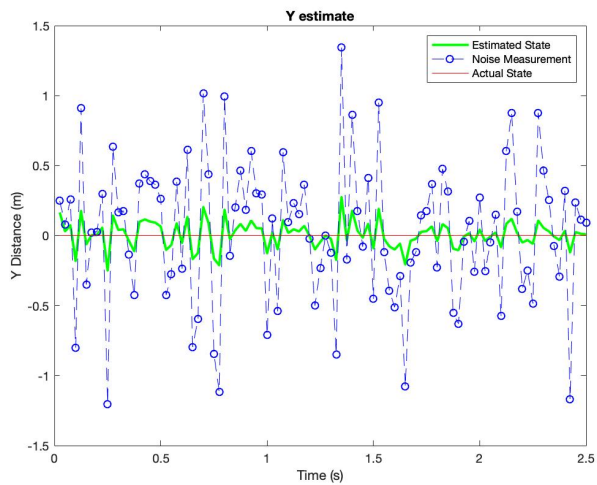


Fig. 6. y estimate using EKF

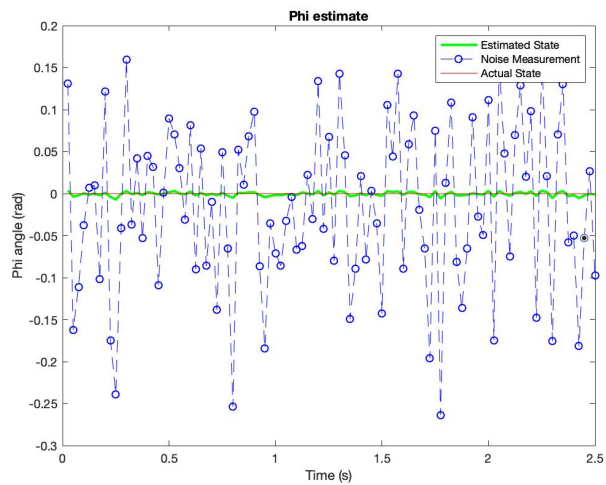


Fig. 9. ϕ estimate using UKF

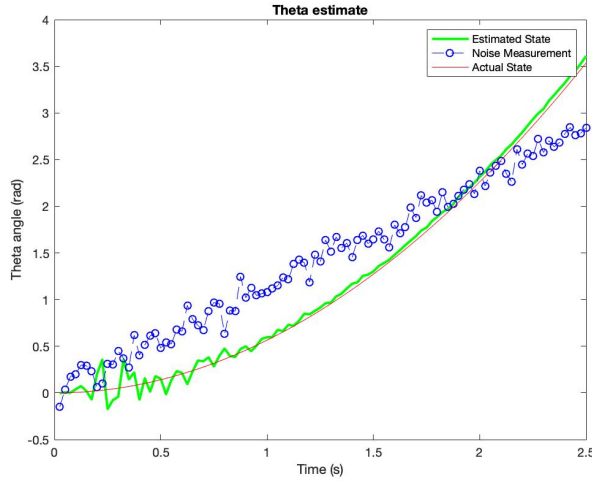


Fig. 10. θ estimate using EKF

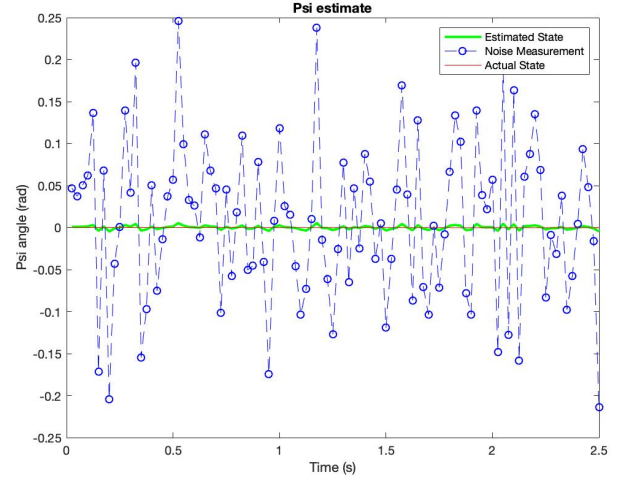


Fig. 13. ψ estimate using UKF

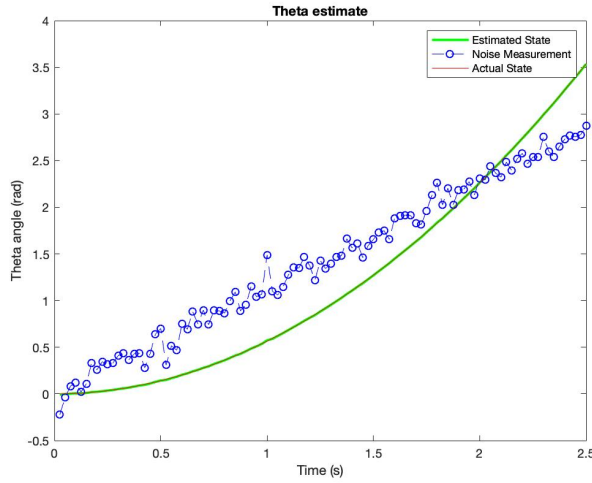


Fig. 11. θ estimate using UKF

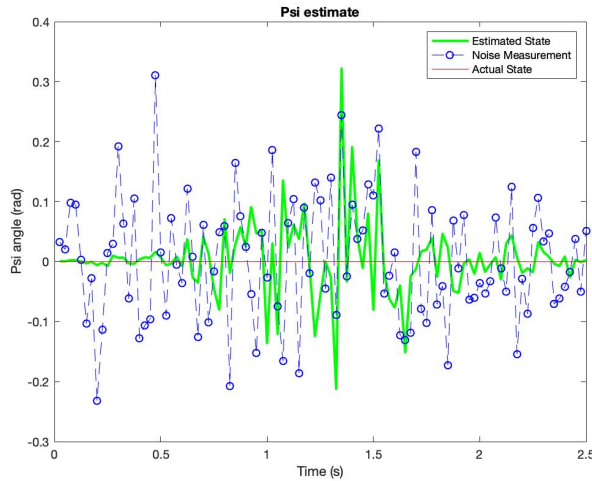


Fig. 12. ψ estimate using EKF

The plots obtained in Fig.8 and Fig.9 shows the comparison between the ϕ estimate obtained from EKF and UKF implementation. The plots obtained in Fig.10 and Fig.11 shows the comparison between the θ estimate obtained from EKF and UKF implementation. The plots obtained in Fig.12 and Fig.13 shows the comparison between the ψ estimate obtained from EKF and UKF implementation.

As from the above figures, we can see that EKF performs better in estimating the state of the quadrotor. it filters out the noise better than UKF. Although EKF performs better in estimating the position states of the quadrotor, UKF stands out while predicting the orientation states of the quadrotor. This is an interesting observation as more non linearity occurs in measuring the orientation angles. This serves to prove that UKF are better and easier to apply for more complex non-linear systems.

IV. CONCLUSION

The quadrotor was modelled in Simulink and filters viz., EKF and UKF were implemented in MATLAB. The study compared the performance of both filters in estimating the quadrotor's position and orientation. EKF performed better at predicting the position with high noise levels. UKF was robust to sensitivities in measurement and performed better in estimating the orientation of the quadrotor. By adding, more sensor information viz., accelerometers, the states can be better estimated. The sensor model can be improved by including drift and bias in measurement to get a more accurate performance of both filter.

REFERENCES

- [1] T.Vincent, "Estimation theory and Kalman filtering".
- [2] Kurak, Sevkuthan, and Migdat Hodzic. "Control and Estimation of a Quadcopter Dynamical Model." Periodicals of Engineering and Natural Sciences (PEN) 6.1 (2018): 63-75.

APPENDIX A
EXTENDED KALMAN FILTER

```

1 %% Extended Kalman Filter
2 x0 = 0.0;
3 y0 = 0.0;
4 z0 = 0.0;
5 xDot0 = 0.0;
6 yDot0 = 0.0;
7 zDot0 = 0.0;
8 phi0 = 0.0;
9 theta0 = 0.0;
10 psi0 = 0.0;
11 phiDot0 = 0.0;
12 thetaDot0 = 0.0;
13 psiDot0 = 0.0;
14 Ts = 0.025;
15 totT = 2.5;
16 sigmaGPS = 0.5;
17 sigmaGY = 0.1;
18 e = 1e-6;
19 x_k0 = [x0;y0;z0;phi0;theta0;psi0;...
20         xDot0;yDot0;zDot0;phiDot0;thetaDot0;
21         psiDot0];
22 count = 1;
23 t = Ts:Ts:totT;
24 P_k1 = eye(12);
25 Q = diag(e*ones(1,12));
26 R = eye(6);
27 R(1,1) = sigmaGPS*R(1,1);
28 R(2,2) = sigmaGPS*R(2,2);
29 R(3,3) = sigmaGPS*R(3,3);
30 R(4,4) = sigmaGY*R(4,4);
31 R(5,5) = sigmaGY*R(5,5);
32 R(6,6) = sigmaGY*R(6,6);
33 H = [1 0 0 0 0 0 0 0 0 0 0 0;...
34      0 1 0 0 0 0 0 0 0 0 0 0;...
35      0 0 1 0 0 0 0 0 0 0 0 0;...
36      0 0 0 0 0 0 0 0 0 1 0 0;...
37      0 0 0 0 0 0 0 0 0 0 1 0;...
38      0 0 0 0 0 0 0 0 0 0 0 1];
39 simout = sim('quadcopter_control',
40             'StartTime', num2str(0), 'StopTime', ...
41             num2str(Ts), 'OutputOption', '
42             SpecifiedOutputTimes', '
43             OutputTimes', ...
44             num2str(Ts));
45 input = simout.Input.Data(end,:);
46 del_x_kp1(:,count) = simout.simout.Data(
47     end,:);
48 yMeasurement(:,count) = simout.
49     sensorMeasure.Data(end,:);
50 yMeasurement(:,count) = yMeasurement(:,
51     count) + [sigmaGPS*randn(3,1);...
52     sigmaGY*randn(3,1)];
53 %

```

```

47 deltaF = zeros(12); % Derivative of the
58     non-linear function matrix
59 %
60 F = zeros(12,1); % Non-linear function
61     matrix
62 %
63 deltaF(1,7) = 1.0; % xDot
64 deltaF(2,8) = 1.0; % yDot
65 deltaF(3,9) = 1.0; % zDot
66 deltaF(4,10) = 1.0; % phiDot
67 deltaF(5,11) = 1.0; % thetaDot
68 deltaF(6,12) = 1.0; % psiDot
69 %
70 deltaF(7,4) = (input(1)/mass)*((-sin(phi0)
71     )*sin(theta0)*cos(psi0))+(cos(phi0)*
72     sin(psi0));
73 deltaF(7,5) = (input(1)/mass)*(cos(phi0)*
74     cos(theta0)*cos(psi0));
75 deltaF(7,6) = (input(1)/mass)*((-cos(phi0)
76     )*sin(theta0)*sin(psi0))+(sin(phi0)*
77     cos(psi0));
78 %
79 deltaF(8,4) = (input(1)/mass)*((-sin(phi0)
80     )*sin(theta0)*sin(psi0))-(cos(phi0)*
81     cos(psi0));
82 deltaF(8,5) = (input(1)/mass)*(cos(phi0)*
83     cos(theta0)*sin(psi0));
84 deltaF(8,6) = (input(1)/mass)*((cos(phi0)
85     )*sin(theta0)*cos(psi0))+(sin(phi0)*sin
86     (psi0));
87 %
88 deltaF(9,4) = (input(1)/mass)*(-sin(phi0)
89     )*cos(theta0);
90 deltaF(9,5) = (input(1)/mass)*(-cos(phi0)
91     )*sin(theta0);
92 %
93 deltaF(10,11) = ((iY-iZ)/iX)*psiDot0;
94 deltaF(10,12) = ((iY-iZ)/iX)*thetaDot0;
95 %
96 deltaF(11,10) = ((iZ-iX)/iY)*psiDot0;
97 deltaF(11,12) = ((iZ-iX)/iY)*phiDot0;
98 %
99 deltaF(12,10) = ((iX-iY)/iZ)*thetaDot0;
100 deltaF(12,11) = ((iX-iY)/iZ)*phiDot0;
101 %
102 phi_k = eye(12) + deltaF*Ts;
103 % EKF
104 % Time update
105 xCap_kMinus = phi_k*(del_x_kp1(:,count));
106 P_kMinus = phi_k*P_k1*phi_k' + Q;
107 % Measurement update
108 K_k = (P_kMinus*H')*inv(H*P_kMinus*H'+R);
109 %
110 xCap_k = xCap_kMinus + K_k*(yMeasurement
111     - H*xCap_kMinus);
112 %

```



```

88 P_k = (eye(12) - K_k*H)*P_kMinus;
89 %
90 x_k = xCap_k;
91 P_k1 = P_k;
92 %
93 stateEstimates(:,count) = x_k(1:6);
94 %
95 x_k0 = del_x_kp1(:,count);
96 %
97 t = Ts:Ts:totT;
98 %
99 for k = 1:(length(t)-1)
100 %
101 x0 = x_k0(1) + e*(-1^count);
102 y0 = x_k0(2) + (e*(-1^(count+1)));
103 z0 = x_k0(3) + (e*(-1^count));
104 xDot0 = x_k0(7) + (e*(-1^(count+1)));
105 yDot0 = x_k0(8) + (e*(-1^count));
106 zDot0 = x_k0(9) + (e*(-1^(count+1)));
107 phi0 = x_k0(4) + (e*(-1^count));
108 theta0 = x_k0(5) + (e*(-1^(count+1)))
109 ;
110 psi0 = x_k0(6) + (e*(-1^count));
111 phiDot0 = x_k0(10) + (e*(-1^(count+1)
112 ));
113 thetaDot0 = x_k0(11) + (e*(-1^count))
114 ;
115 psiDot0 = x_k0(12) + (e*(-1^(count+1)
116 ));
117 %
118 simout = sim('quadcopter_control','
119 StartTime',num2str(0),'StopTime'
120 ,...
121 num2str(Ts),'OutputOption','
122 SpecifiedOutputTimes','
123 OutputTimes',...
124 num2str(Ts));
125 input = simout.Input.Data(end,:);
126 %
127 del_x_kp1(:,count+1) = simout.simout.
128 Data(end,:);
129 yMeasurement(:,count+1) = simout.
130 sensorMeasure.Data(end,:);
131 yMeasurement(:,count+1) =
132 yMeasurement(:,count+1) + [
133 sigmaGPS*randn(3,1);...
134 sigmaGY*randn(3,1)];
135 %
136 x = x_k(1);
137 y = x_k(2);
138 z = x_k(3);
139 xDot = x_k(7);
140 yDot = x_k(8);
141 zDot = x_k(9);
142 phi = x_k(4);
143 theta = x_k(5);
144
145 psi = x_k(6);
146 phiDot = x_k(10);
147 thetaDot = x_k(11);
148 psiDot = x_k(12);
149 % EKF
150 deltaF = zeros(12); % Derivative of
151 the non-linear function matrix
152 F = zeros(12);
153 %
154 deltaF(1,7) = 1.0; % xDot
155 deltaF(2,8) = 1.0; % yDot
156 deltaF(3,9) = 1.0; % zDot
157 deltaF(4,10) = 1.0; % phiDot
158 deltaF(5,11) = 1.0; % thetaDot
159 deltaF(6,12) = 1.0; % psiDot
160 %
161 deltaF(7,4) = (input(1)/mass)*((-sin(
162 phi)*sin(theta)*cos(psi))+(cos(phi)
163 )*sin(psi));
164 deltaF(7,5) = (input(1)/mass)*(cos(
165 phi)*cos(theta)*cos(psi));
166 deltaF(7,6) = (input(1)/mass)*((-cos(
167 phi)*sin(theta)*sin(psi))+(sin(phi)
168 )*cos(psi));
169 %
170 deltaF(8,4) = (input(1)/mass)*((-sin(
171 phi)*sin(theta)*sin(psi))-cos(phi)
172 )*cos(psi));
173 deltaF(8,5) = (input(1)/mass)*(cos(
174 phi)*cos(theta)*sin(psi));
175 deltaF(8,6) = (input(1)/mass)*((cos(
176 phi)*sin(theta)*cos(psi))+(sin(phi)
177 )*sin(psi));
178 %
179 deltaF(9,4) = (input(1)/mass)*(-sin(
180 phi)*cos(theta));
181 deltaF(9,5) = (input(1)/mass)*(-cos(
182 phi)*sin(theta));
183 %
184 deltaF(10,11) = ((iY-iZ)/iX)*psiDot;
185 deltaF(10,12) = ((iY-iZ)/iX)*thetaDot
186 ;
187 %
188 deltaF(11,10) = ((iZ-iX)/iY)*psiDot;
189 deltaF(11,12) = ((iZ-iX)/iY)*phiDot;
190 %
191 deltaF(12,10) = ((iX-iY)/iZ)*thetaDot
192 ;
193 deltaF(12,11) = ((iX-iY)/iZ)*phiDot;
194 %
195 phi_k = eye(12) + deltaF*Ts;
196 % EKF
197 % Time update
198 xCap_kMinus = phi_k*(del_x_kp1(:,
199 count+1));
200 P_kMinus = phi_k*P_k1*phi_k' + Q;

```

```

172 % Measurement update
173 K_k = (P_kMinus*H')*inv(H*P_kMinus*H
    '+R);
174 %
175 xCap_k = xCap_kMinus + K_k*(
    yMeasurement(:,count+1) - H*
    xCap_kMinus);
176 %
177 P_k = (eye(12) - K_k*H)*P_kMinus;
178 %
179 x_k = xCap_k;
180 P_k1 = P_k;
181 %
182 count = count + 1;
183 stateEstimates(:,count) = x_k(1:6);
184 %
185 x_k0 = del_x_kp1(:,count);
186 end
187 %% Plot the results
188 %
189 figure
190 p = plot(t, stateEstimates(3,:), 'g', t,
    yMeasurement(3,:), 'b—o', t, del_x_kp1
    (3,:), 'r');
191 p(1).LineWidth = 2;
192 title('Z estimate')
193 xlabel('Time (s)')
194 ylabel('Z Height (m)')
195 legend('Estimated State', 'Noise
    Measurement', 'Actual State')
196 %
197 figure
198 p = plot(t, stateEstimates(1,:), 'g', t,
    yMeasurement(1,:), 'b—o', t, del_x_kp1
    (1,:), 'r');
199 p(1).LineWidth = 2;
200 title('X estimate')
201 xlabel('Time (s)')
202 ylabel('X Distance (m)')
203 legend('Estimated State', 'Noise
    Measurement', 'Actual State')
204 %
205 figure
206 p = plot(t, stateEstimates(2,:), 'g', t,
    yMeasurement(2,:), 'b—o', t, del_x_kp1
    (2,:), 'r');
207 p(1).LineWidth = 2;
208 title('Y estimate')
209 xlabel('Time (s)')
210 ylabel('Y Distance (m)')
211 legend('Estimated State', 'Noise
    Measurement', 'Actual State')
212 %
213 figure
214 p = plot(t, stateEstimates(4,:), 'g', t,
    yMeasurement(4,:), 'b—o', t, del_x_kp1

```

```

    (4,:), 'r');
215 p(1).LineWidth = 2;
216 title('Phi estimate')
217 xlabel('Time (s)')
218 ylabel('Phi angle (rad)')
219 legend('Estimated State', 'Noise
    Measurement', 'Actual State')
220 %
221 figure
222 p = plot(t, stateEstimates(5,:), 'g', t,
    yMeasurement(5,:), 'b—o', t, del_x_kp1
    (5,:), 'r');
223 p(1).LineWidth = 2;
224 title('Theta estimate')
225 xlabel('Time (s)')
226 ylabel('Theta angle (rad)')
227 legend('Estimated State', 'Noise
    Measurement', 'Actual State')
228 %
229 figure
230 p = plot(t, stateEstimates(6,:), 'g', t,
    yMeasurement(6,:), 'b—o', t, del_x_kp1
    (6,:), 'r');
231 p(1).LineWidth = 2;
232 title('Psi estimate')
233 xlabel('Time (s)')
234 ylabel('Psi angle (rad)')
235 legend('Estimated State', 'Noise
    Measurement', 'Actual State')
236 %

```

APPENDIX B UNSCENTED KALMAN FILTER

```

1 %% Unscented Kalman Filter
2 %
3 Ts = 0.025;
4 totT = 2.5;
5 sigmaGPS = 0.25;
6 sigmaGY = 0.1;
7 e = 1e-6;
8 %
9 x0 = 0.0;
10 y0 = 0.0;
11 z0 = 0.0;
12 xDot0 = 0.0;
13 yDot0 = 0.0;
14 zDot0 = 0.0;
15 phi0 = 0.0;
16 theta0 = 0.0;
17 psi0 = 0.0;
18 phiDot0 = 0.0;
19 thetaDot0 = 0.0;
20 psiDot0 = 0.0;
21 %
22 count = 1;
23 %

```



```

24 t = Ts:Ts:totT;
25 %
26 P_k1 = eye(12);
27 %
28 Q = diag(e*ones(1,12));
29 %
30 R = eye(6);
31 R(1,1) = sigmaGPS*R(1,1);
32 R(2,2) = sigmaGPS*R(2,2);
33 R(3,3) = sigmaGPS*R(3,3);
34 R(4,4) = sigmaGY*R(4,4);
35 R(5,5) = sigmaGY*R(5,5);
36 R(6,6) = sigmaGY*R(6,6);
37 %
38 stateEstimates = zeros(6,24);
39 %
40 x_k0 = [x0;y0;z0;phi0;theta0;psi0;...
41         xDot0;yDot0;zDot0;phiDot0;thetaDot0;
42         psiDot0];
43 %
44 n = length(x_k0);
45 %
46 for k = 1:length(t)
47     %
48     x0 = x_k0(1)+e*(-1^(count+1));
49     y0 = x_k0(2)+e*(-1^count);
50     z0 = x_k0(3)+e*(-1^(count+1));
51     xDot0 = x_k0(7)+e*(-1^count);
52     yDot0 = x_k0(8)+e*(-1^(count+1));
53     zDot0 = x_k0(9)+e*(-1^count);
54     phi0 = x_k0(4)+e*(-1^(count+1));
55     theta0 = x_k0(5)+e*(-1^count);
56     psi0 = x_k0(6)+e*(-1^(count+1));
57     phiDot0 = x_k0(10)+e*(-1^count);
58     thetaDot0 = x_k0(11)+e*(-1^(count+1));
59     ;
60     psiDot0 = x_k0(12)+e*(-1^count);
61     %
62     simout = sim('quadcopter_control','
63                 StartTime',num2str(0),'StopTime'
64                 ,...
65                 num2str(Ts),'OutputOption','
66                 SpecifiedOutputTimes','
67                 OutputTimes',...
68                 num2str(Ts));
69     %
70     input = simout.Input.Data(end,:);
71     %
72     del_x_kp1(:,count) = simout.simout.
73         Data(end,:);
74     yMeasurement(:,count) = simout.
75         sensorMeasure.Data(end,:);
76     yMeasurement(:,count) = yMeasurement
77         (:,count) + [sigmaGPS*randn(3,1)
78         ;...
79         sigmaGY*randn(3,1)];

```

```

70 % Time update
71 M = chol(P_k1,'upper');
72 %
73 xBar_i_p = sqrt(n)*[M -M];
74 %
75 xI_p = x_k0 + xBar_i_p;
76 %
77 for i = 1:(2*n)
78     %
79     x0 = xI_p(1,i);
80     y0 = xI_p(2,i);
81     z0 = xI_p(3,i);
82     phi0 = xI_p(4,i);
83     theta0 = xI_p(5,i);
84     psi0 = xI_p(6,i);
85     xDot0 = xI_p(7,i);
86     yDot0 = xI_p(8,i);
87     zDot0 = xI_p(9,i);
88     phiDot0 = xI_p(10,i);
89     thetaDot0 = xI_p(11,i);
90     psiDot0 = xI_p(12,i);
91     %
92     simout_Temp = sim('
93                 quadcopter_control','StartTime
94                 ',num2str(0),'StopTime',...
95                 num2str(Ts),'OutputOption','
96                 SpecifiedOutputTimes','
97                 OutputTimes',...
98                 num2str(Ts));
99     %
100     x_ki(:,i) = simout_Temp.simout.
101         Data(end,:);
102 end
103 %
104 xCap_Minus = (1/(2*n))*sum(x_ki,2);
105 %
106 PMinus = zeros(12);
107 %
108 for i = 1:(2*n)
109     PMinus = PMinus + (x_ki(:,i)-
110         xCap_Minus)*(x_ki(:,i)-
111         xCap_Minus)' + Q;
112 end
113 %
114 PMinus = (1/(2*n)).*PMinus;
115 %
116 % Measurement update
117 M = chol(PMinus,'upper');
118 %
119 xBar_i = sqrt(n)*[M -M];
120 %
121 xI_p = xCap_Minus + xBar_i;
122 %
123 for i = 1:(2*n)
124     %
125     x0 = xI_p(1,i);

```

```

119     y0 = xI_p(2,i);
120     z0 = xI_p(3,i);
121     phi0 = xI_p(4,i);
122     theta0 = xI_p(5,i);
123     psi0 = xI_p(6,i);
124     xDot0 = xI_p(7,i);
125     yDot0 = xI_p(8,i);
126     zDot0 = xI_p(9,i);
127     phiDot0 = xI_p(10,i);
128     thetaDot0 = xI_p(11,i);
129     psiDot0 = xI_p(12,i);
130     %
131     simout_Temp = sim('
        quadcopter_control','StartTime
        ',num2str(0),'StopTime',...
132         num2str(Ts),'OutputOption','
        SpecifiedOutputTimes','
        OutputTimes',...
133         num2str(Ts));
134     %
135     yMeasure(:,i) = simout_Temp.
        sensorMeasure.Data(end,:);
136
137     end
138     %
139     yBar = (1/(2*n))*sum(yMeasure,2);
140     %
141     Pxy = zeros(12,6);
142     Py = zeros(6);
143     %
144     for i = 1:(2*n)
145         Pxy = Pxy + (xI_p(:,i)-xCap_Minus)
            *(yMeasure(:,i)-yBar)';
146         Py = Py + (yMeasure(:,i)-yBar)*(
            yMeasure(:,i)-yBar)' + R;
147     end
148     %
149     Pxy = (1/(2*n)).*Pxy;
150     Py = (1/(2*n)).*Py;
151     %
152     K = Pxy*inv(Py);
153     %
154     xCap = xCap_Minus + K*(yMeasurement
        (:,count) - yBar);
155     %
156     P_k = PMinus - K*Pxy;
157     %
158     stateEstimates(:,count) = xCap(1:6);
159     %
160     x_k0 = del_x_kp1(:,count);
161     %
162     count = count + 1;
163 end
164 %% Plot the results
165 %
166 figure

```

```

167 p = plot(t, stateEstimates(3,:), 'g', t,
        yMeasurement(3,:), 'b—o', t, del_x_kp1
        (3,:), 'r');
168 p(1).LineWidth = 2;
169 title('Z estimate')
170 xlabel('Time (s)')
171 ylabel('Z Height (m)')
172 legend('Estimated State','Noise
        Measurement','Actual State')
173 %
174 figure
175 p = plot(t, stateEstimates(1,:), 'g', t,
        yMeasurement(1,:), 'b—o', t, del_x_kp1
        (1,:), 'r');
176 p(1).LineWidth = 2;
177 title('X estimate')
178 xlabel('Time (s)')
179 ylabel('X Distance (m)')
180 legend('Estimated State','Noise
        Measurement','Actual State')
181 %
182 figure
183 p = plot(t, stateEstimates(2,:), 'g', t,
        yMeasurement(2,:), 'b—o', t, del_x_kp1
        (2,:), 'r');
184 p(1).LineWidth = 2;
185 title('Y estimate')
186 xlabel('Time (s)')
187 ylabel('Y Distance (m)')
188 legend('Estimated State','Noise
        Measurement','Actual State')
189 %
190 figure
191 p = plot(t, stateEstimates(4,:), 'g', t,
        yMeasurement(4,:), 'b—o', t, del_x_kp1
        (4,:), 'r');
192 p(1).LineWidth = 2;
193 title('Phi estimate')
194 xlabel('Time (s)')
195 ylabel('Phi angle (rad)')
196 legend('Estimated State','Noise
        Measurement','Actual State')
197 %
198 figure
199 p = plot(t, stateEstimates(5,:), 'g', t,
        yMeasurement(5,:), 'b—o', t, del_x_kp1
        (5,:), 'r');
200 p(1).LineWidth = 2;
201 title('Theta estimate')
202 xlabel('Time (s)')
203 ylabel('Theta angle (rad)')
204 legend('Estimated State','Noise
        Measurement','Actual State')
205 %
206 figure
207 p = plot(t, stateEstimates(6,:), 'g', t,

```

```
        yMeasurement(6,:), 'b—o', t, del_x_kp1
        (6,:), 'r');
208 p(1).LineWidth = 2;
209 title('Psi estimate')
210 xlabel('Time (s)')
211 ylabel('Psi angle (rad)')
212 legend('Estimated State', 'Noise
        Measurement', 'Actual State')
213 %
```