Midterm

Mark Petersen

October 31, 2019

1 Introduction

A multirotor is flying with constant altitude and receiving bearing and range measurements to several landmarks in order to localize itself. The location of the landmarks are known, but the pose of the multirotor isn't. For the midterm I will use and Extended Information Filter (EIF) in order to localize the multirotor.

2 System and Observation Model

The multirotor is modeled using a simple discrete unicycle model of the form

$$X_k = f\left(X_{k-1}, u_k\right) + g\left(X_{k-1}, \varepsilon_k\right) \tag{1}$$

$$z_k = h\left(X_k, \ell_i\right) + w_k \tag{2}$$

The subscript k denote the time index. The state vector $X \in \mathbb{R}^3$ contains the mutilator's x position, y position, and heading θ such that $X_k = \left[x_k, y_k, \theta_k\right]^\top$. The input vector $u \in \mathbb{R}^2$ contains the velocity, v, and angular rate, ω, such that $u_k = \left[v_k, \omega_k\right]^\top$. The noise vector $\varepsilon \in \mathbb{R}^2$ contains noise on velocity, ε_v and noise on angular rate, ε_ω, such $\varepsilon_k = \left[\varepsilon_{v,k}, \varepsilon_{\omega,k}\right]$. The system function $f\left(X_{k-1}, u_k\right)$ maps the previous state, X_{k-1}, and current input, u_k, to the current state X_k provided that there is no noise. The system function is described as

$$f\left(X_{k-1}, u_k\right) = \begin{bmatrix} x_{k-1} + \left(v_k \cos\theta_{k-1}\right)\delta \\ y_{k-1} + \left(v_k \sin\theta_{k-1}\right)\delta \\ \theta_{k-1} + \omega_k \delta \end{bmatrix} \tag{3}$$

where δ is the time interval between $k-1$ and k.

The noise function $g\left(X_{k-1}, \varepsilon_k\right)$ maps the previous state and current noise ε_k to process noise. It is described as

$$g\left(X_{k-1}, \varepsilon_k\right) = \begin{bmatrix} \varepsilon_v \cos\left(\theta_{k-1}\right)\delta \\ \varepsilon_v \sin\left(\theta_{k-1}\right)\delta \\ \varepsilon_\omega \delta \end{bmatrix} \tag{4}$$

The observation function $h\left(X_k, \ell_i\right)$ maps the current states, and the i^{th} landmark location ℓ_i to the relative range $r_{i,k}$ and bearing, $\phi_{i,k}$. The observation is

$$h\left(X_k, \ell_i\right) = \begin{bmatrix} \sqrt{q} \\ \arctan\left(m_{i,y} - y_k, m_{i,x} - x_k\right) - \theta_k \end{bmatrix} \tag{5}$$

where $m_{i,y}$ and $m_{i,x}$ are the x and y position of landmark ℓ_i and

$$q = \left(m_{i,x} - x_k\right)^2 + \left(m_{i,y} - y_k\right)^2 \tag{6}$$

The measurement noise $w \sim \mathcal{N}\left(0, W\right)$ and the process noise $\varepsilon \sim \mathcal{N}\left(0, \epsilon\right)$ where

$$W = \begin{bmatrix} \sigma_r^2 & 0 \\ 0 & \sigma_\phi^2 \end{bmatrix} \tag{7}$$

$$\epsilon = \begin{bmatrix} \sigma_v^2 & 0 \\ 0 & \sigma_\omega^2 \end{bmatrix} \tag{8}$$

with the values of the standard deviations being

$$\sigma_r = 0.2 \tag{9}$$
$$\sigma_\phi = 0.1 \tag{10}$$
$$\sigma_v = 0.15 \tag{11}$$
$$\sigma_\omega = 0.1 \tag{12}$$

3 Linearization

In order to use an EIF, we need to linearize the system function, noise function and observation function.

$$F_k = \frac{\partial f\left(X_{k-1}, u_k\right)}{\partial X_{x-1}} \tag{13}$$
$$= \begin{bmatrix} 1 & 0 & -v_k \sin\left(\theta_{k-1}\right)\delta \\ 0 & 1 & v_k \cos\left(\theta_{k-1}\right)\delta \\ 0 & 0 & 1 \end{bmatrix} \tag{14}$$

$$G_k = \frac{g\left(X_{k-1}, \varepsilon_k\right)}{\partial \varepsilon} \tag{15}$$
$$= \begin{bmatrix} \cos\left(\theta_{k-1}\right)\delta & 0 \\ \sin\left(\theta_{k-1}\right)\delta & 0 \\ 0 & \delta \end{bmatrix} \tag{16}$$

$$H_k = \frac{\partial h\left(X_k, \ell_i\right)}{\partial X_k} \tag{17}$$
$$= \begin{bmatrix} -\frac{(m_{i,x}-x_k)}{\sqrt{q}} & -\frac{(m_{i,y}-y_k)}{\sqrt{q}} & 0 \\ \frac{m_{i,y}-y_k}{q} & -\frac{(m_{i,x}-x_k)}{q} & -1 \end{bmatrix} \tag{18}$$

4 EIF

The EIF is similar to an EKF with slight modifications. We begin by introducing the new variable

$$\zeta_k = P_k^{-1}\mu_k$$

where u_k is the estimate of X_k, P_k is the error covariance, and ζ_k is the transformed μ_k. Using the new variable, we can break up the the EIF into the predict phase and the update phase. The predict phase is

$$\mu_{k-1|j} = P_{k-1|j}\zeta_{k-1|j} \tag{19}$$
$$P_{k|j}^{-1} = \left(F_k P_{k-1|j} F_k^\top + G_k \epsilon G_k^\top\right)^{-1} \tag{20}$$
$$\zeta_{k|j} = P_{k|j}^{-1} f\left(\mu_{k-1|j}, u_k\right) \tag{21}$$

and the update phase is

$$\mu_{k|j} = f\left(\mu_{k-1|j}, u_k\right) = P_{k|j}\zeta_{k|j}$$
$$P_{k|j=k}^{-1} = P_{k|j}^{-1} + H_k^\top W_k^{-1} H_k$$
$$\zeta_{k|j=k} = \zeta_{k|j} + H_k^\top W_k^{-1}\left(z_k - h\left(\mu_{k|j}, \ell_i\right) + H_k \mu_{k|j}\right)$$

The subscript $k|j$ indicate a variable at time step k given measurements up to time step j where $j \leq k$. Also, the vector $z_k = \left[r_k, \phi_k\right]^\top$. For implementation, it can be easier to let $\Omega = P^{-1}$. It should be noted that if you have more than one measurement at a given time step, then you will run the update phase for every measurement.

5 EIF Code

The EIF was operated using object oriented programming in MATLAB. The class is included in this section.

```
classdef EIF < handle

    properties
        mu = zeros(3,1);      % Estimated state
        W  = zeros(2,2);      % Measurement noise covariance
        ep = zeros(2,2);      % Process noise covariance
        zeta = zeros(3,1);    % Transformed estimated state
        P = zeros(3,3);       % Error covariance
        Omega = zeros(3,3);   % Inverse error covariance

        mu_history = [];
        zeta_history = [];
        P_history = [];

    end

    methods
        function obj = EIF(mu0,P)
            % Initialize EIF parameters
            obj.mu = mu0;
            obj.P = P;
            obj.Omega = inv(obj.P);
            obj.zeta = obj.Omega*obj.mu;

            % Measurement noise covariance
            obj.W = [0.2^2,  0;...
                     0,      0.1^2];

            % Process noise covariance
            obj.ep = [0.15^2,  0;...
                      0,       0.1^2];

            obj.UpdateHistory();

        end

        function Predict(obj,Ts,u)
            % Predict step for EIF.
            % Ts is the time step
            % u: is the input vector u=[v,w]'

            % Compute jacobians
            Fk = obj.F(Ts,u);
            Gk = obj.G(Ts);

            % Predict inverse error covariance and estimated transformed state
            obj.Omega = inv(Fk/obj.Omega*Fk'+Gk*obj.ep*Gk');
            obj.zeta = obj.Omega*obj.f(Ts,u);

            % Update mu and error covariance
            obj.mu = obj.Omega\obj.zeta;
```

```
        obj.P = inv(obj.Omega);

    end

    function Update(obj,r,phi,ell)
        % The update phase of the EIF.
        % r: is the measured range to the target
        % phi: is the relative angle to the target
        % ell: the location of the target. ell = [mx,my]
        z = [r;phi];

        % Compute jacobian
        Hk = obj.H(ell);

        % Update inverse error covariance and estimated transformed state
        obj.Omega = obj.Omega + Hk'*inv(obj.W)*Hk;
        er = z-obj.h(ell);

        % Wrap the error
        if (er(2) > pi)
            er(2) = er(2)-2*pi;
        elseif(er(2) < -pi)
            er(2) = er(2)+2*pi;
        end

        obj.zeta  = obj.zeta + Hk'*inv(obj.W)*(er+Hk*obj.mu);

        % Update error covariance and estimated state
        obj.P = inv(obj.Omega);
        obj.mu = obj.Omega\obj.zeta;

    end


    function X=f(obj,Ts,u)
        % The system function
        % Ts: time step
        % u:  input to the system
        x = obj.mu(1);      % x position of UAV
        y = obj.mu(2);      % y position of UAV
        th = obj.mu(3);     % heading of UAV
        v = u(1);           % velocity
        w = u(2);           % angular rate

        % Construct the updated state
        X=[x + v*cos(th)*Ts;...
           y + v*sin(th)*Ts;...
           th + w*Ts];
    end

    function z=h(obj,ell)
        % Observation funciton. It computes the range and relative
        % bearing to the landmark ell
        % ell: position of landmark
```

```
        mx = ell(1);    % x position of landmark
        my = ell(2);    % y position of landmark
        x = obj.mu(1);      % x position of UAV
        y = obj.mu(2);      % y position of UAV
        th = obj.mu(3);     % heading of UAV

        % Wrap theta
        if (th > pi)
            th = th-2*pi;
        elseif(th < -pi)
            th = th+2*pi;
        end

        % Construct the estimated observation
        q = (mx-x)^2 + (my-y)^2;
        z = [sqrt(q);...
            atan2(my-y,mx-x)-th];

        % Wrap the heading measurement
        if (z(2) > pi)
            z(2) = z(2)-2*pi;
        elseif(z(2) < -pi)
            z(2) = z(2)+2*pi;
        end

    end

    function F_out = F(obj,Ts,u)
        % Returns the jacobian of the system funciton
        % The system function
        % Ts: time step
        % u:  input to the system
        th = obj.mu(3);     % heading of UAV
        v = u(1);           % velocity

        F_out = [1, 0, -v*sin(th)*Ts;...
                 0, 1,  v*cos(th)*Ts;...
                 0, 0,   1];
    end

    function G_out = G(obj,Ts)
        % Returns the jacobian of the process noise function
        % Ts: time step
        th = obj.mu(3);     % heading of UAV

        G_out = [cos(th)*Ts, 0;...
                 sin(th)*Ts, 0;...
                      0,       Ts];

    end

    function H_out = H(obj,ell)
        % Return the jacobian of the observation function
        % ell: position of landmark
```

```
            mx = ell(1);    % x position of landmark
            my = ell(2);    % y position of landmark
            x = obj.mu(1);      % x position of UAV
            y = obj.mu(2);      % y position of UAV

            q = (mx-x)^2 + (my-y)^2;

            H_out = [-(mx-x)/sqrt(q),  -(my-y)/sqrt(q),  0;...
                      (my-y)/q,         -(mx-x)/q,       -1];
        end

        function UpdateHistory(obj)
            obj.mu_history = [obj.mu_history,obj.mu];
            obj.P_history =  cat(3,obj.P_history,obj.P);
            obj.zeta_history = [obj.zeta_history,obj.zeta];
        end
    end
end
```

6 Results

In this section we present the results of the simulation.

The figures in this section show that the estimated state converged to the true state using the EIF described in the note.

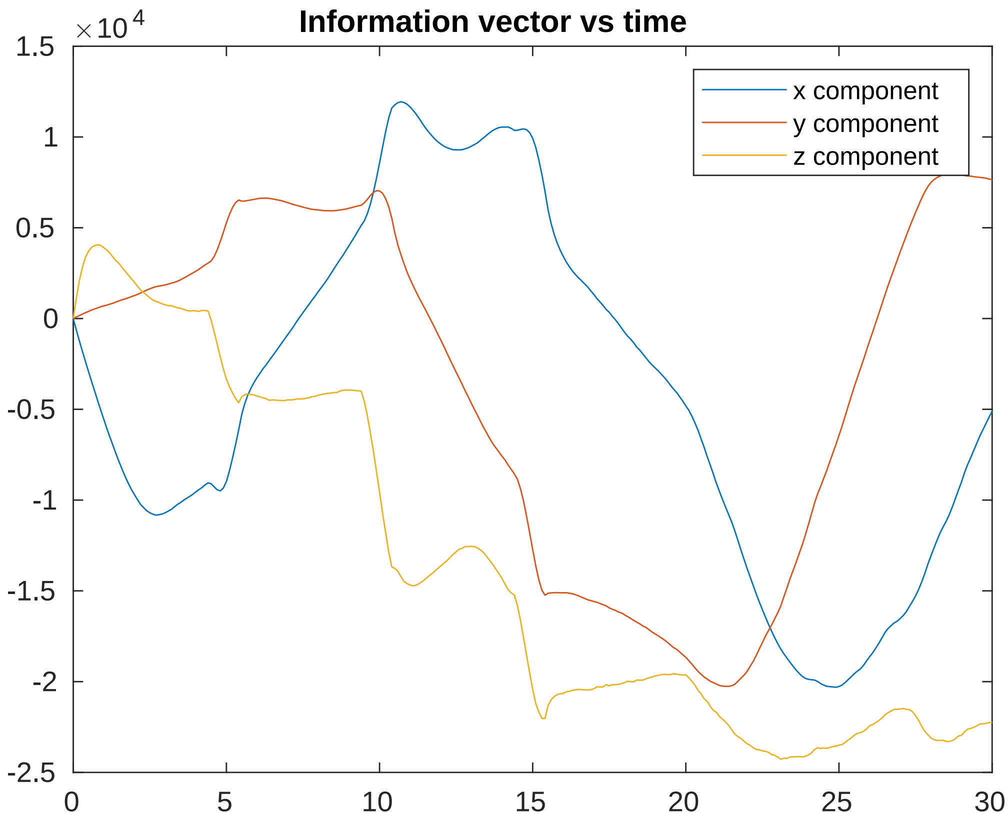

Figure 1: A plot of the components of the information vector as a function of time.

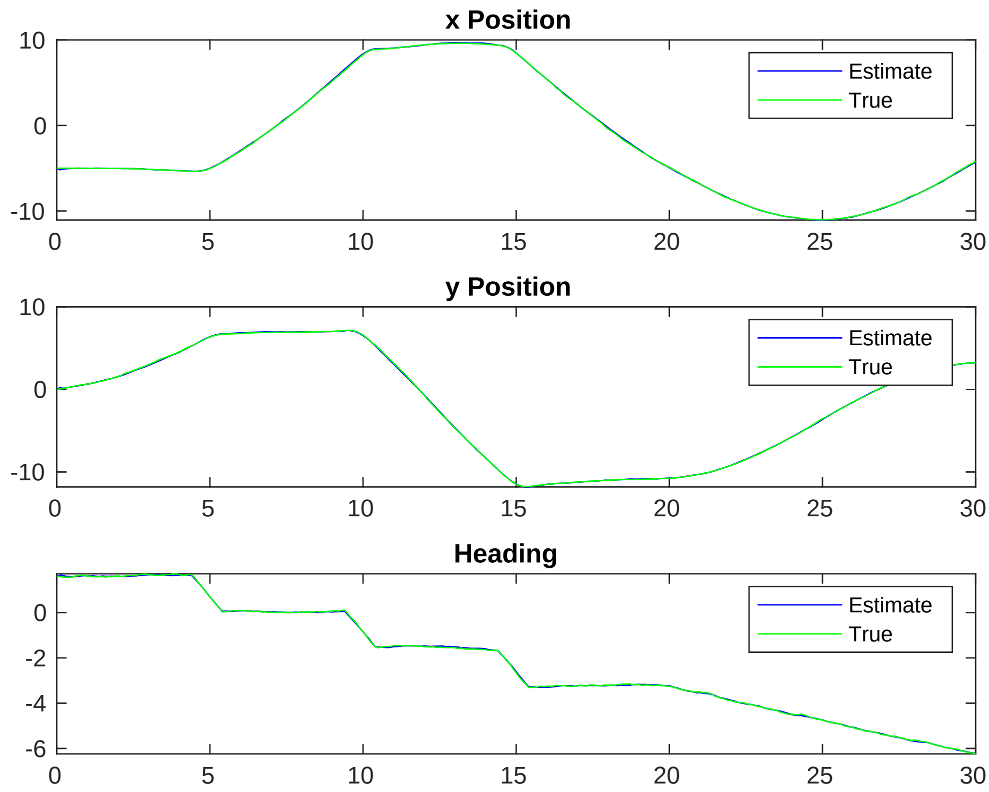

Figure 2: Plots of the estimated state and true states versus time.

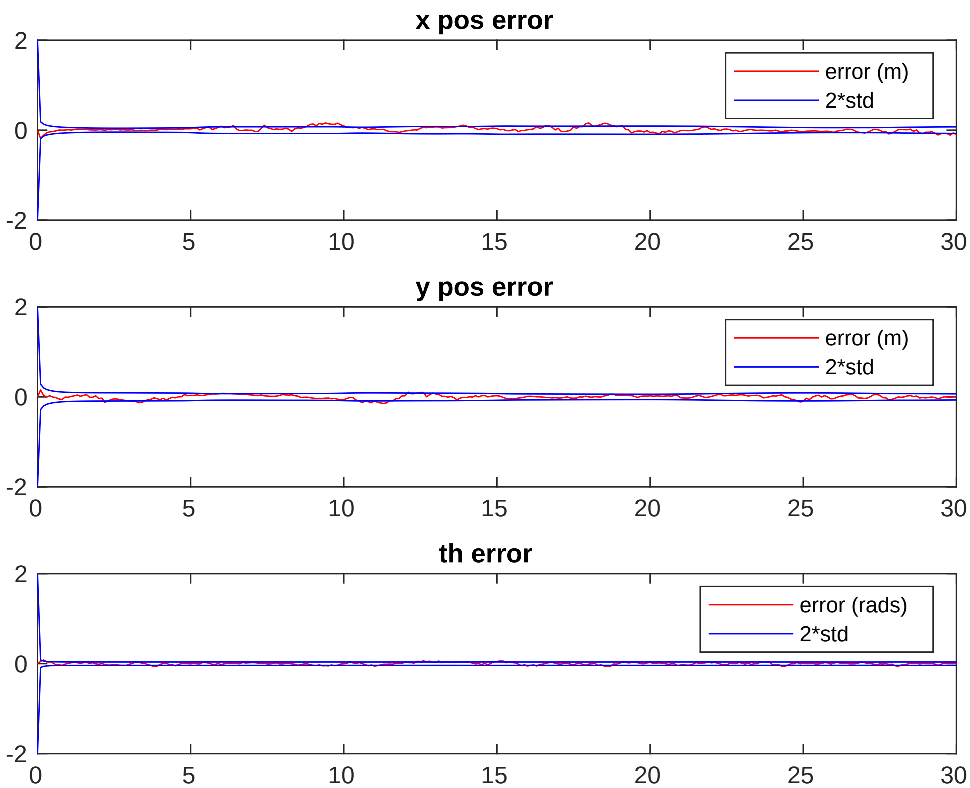

Figure 3: A plots showing the error bounded by the 95 percentile of the error covariancs.

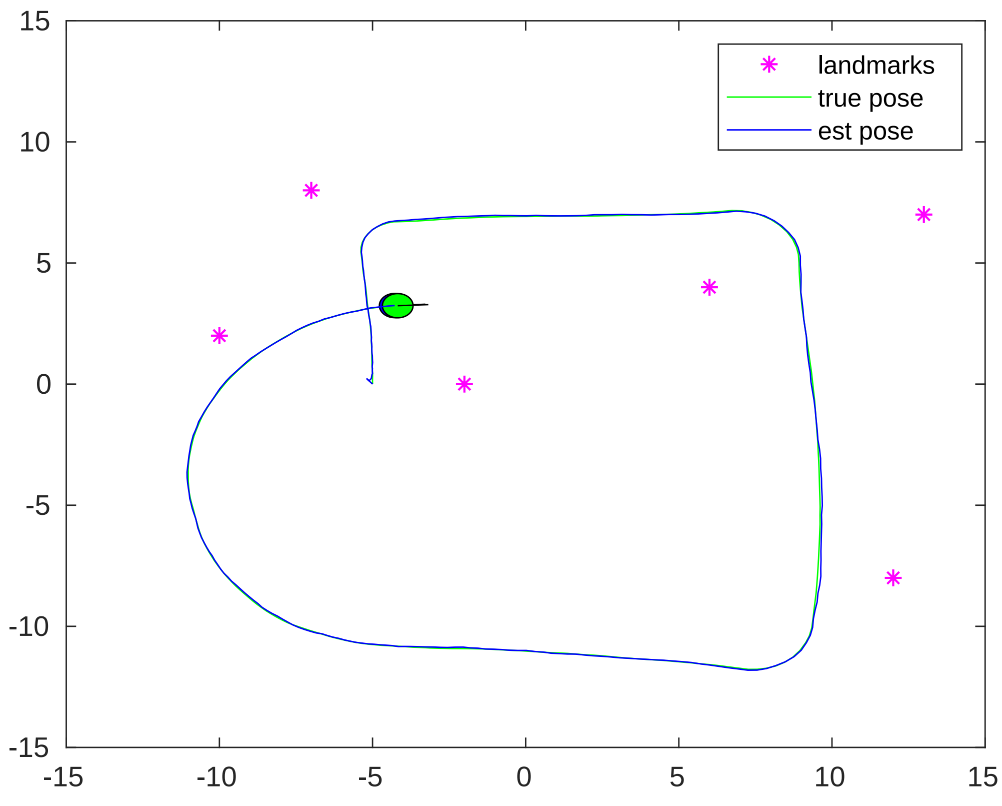

Figure 4: The image shows the landmark locations and the trajectory of the multirotor true and estimated pose.