Midterm - Extended Information Filter

Ryan Day

October 2019

1 Equations

$$\begin{pmatrix} x_t = x_{t-1} + (v_t \cos\theta_{t-1})\,dt \\ y_t = y_{t-1} + v_t \sin\theta_{t-1})\,dt \\ \theta_t = \theta_{t-1} + \omega_t dt \end{pmatrix} = g(u_t, x_{t-1})$$

$$G_t = \frac{\partial g(u_t, \mu_{t-1})}{\partial x_{t-1}} = \begin{matrix} \frac{\partial x'}{\partial \mu_{t-1,x}} & \frac{\partial x'}{\partial \mu_{t-1,y}} & \frac{\partial x'}{\partial \mu_{t-1,\theta}} \\ \frac{\partial y'}{\partial \mu_{t-1,x}} & \frac{\partial y'}{\partial \mu_{t-1,y}} & \frac{\partial y'}{\partial \mu_{t-1,\theta}} \\ \frac{\partial \theta'}{\partial \mu_{t-1,x}} & \frac{\partial \theta'}{\partial \mu_{t-1,y}} & \frac{\partial \theta'}{\partial \mu_{t-1,\theta}} \end{matrix}$$

$$\boxed{G_t = \begin{bmatrix} 1 & 0 & -v_t \sin\theta_{t-1}\Delta t \\ 0 & 1 & v_t \cos\theta_{t-1}\Delta t \\ 0 & 0 & 1 \end{bmatrix}}$$

$$V_t = \frac{\partial g(u_t, \mu_{t-1})}{\partial \mu_t} = \begin{bmatrix} \frac{\partial x'}{\partial v_t} & \frac{\partial x'}{\partial w_t} \\ \frac{\partial y'}{\partial v_t} & \frac{\partial y'}{\partial w_t} \\ \frac{\partial \theta'}{\partial v_t} & \frac{\partial \theta}{\partial w_t} \end{bmatrix}$$

$$\boxed{M_t = \begin{bmatrix} \sigma_v^2 & 0 \\ 0 & \sigma_w^2 \end{bmatrix} = \begin{bmatrix} 0.15^2 & 0 \\ 0 & 0.1^2 \end{bmatrix}}$$

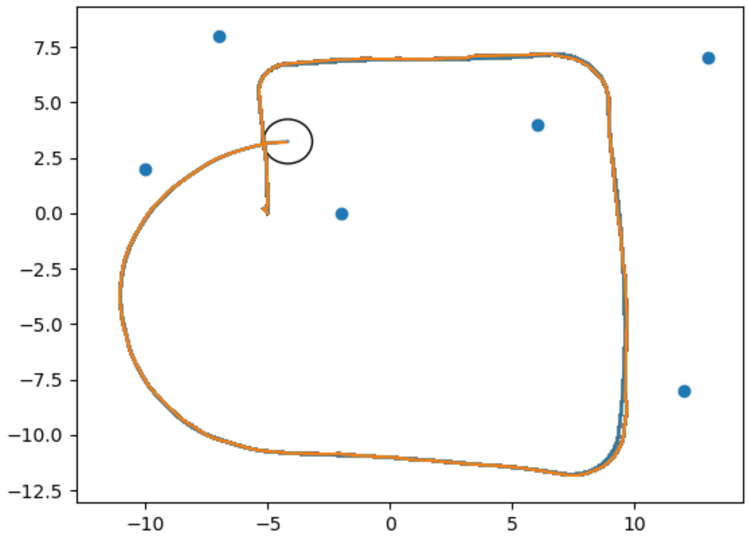
$$\boxed{V_t = \begin{bmatrix} \cos\theta_{t-1}\Delta t & 0 \\ \sin\theta_{t-1}\Delta t & 0 \\ 0 & \Delta t \end{bmatrix}}$$

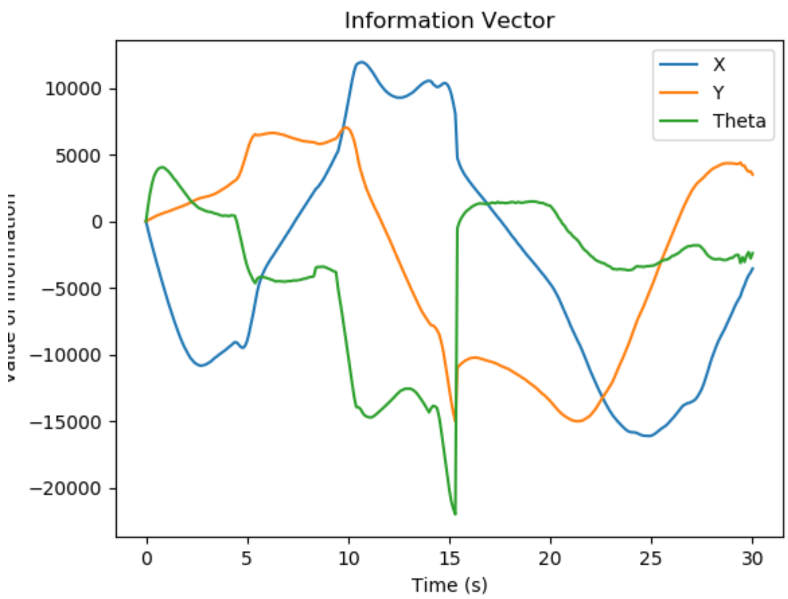
$$h(\bar{\mu}_t) = \begin{bmatrix} r_t \\ \phi_t \end{bmatrix} = \begin{pmatrix} \sqrt{(m_{j,x}-x)^2 + (m_{j,y}-y)^2} \\ \operatorname{atan2}(m_{j,y}-y,\, m_{j,x}-x) - \theta \end{pmatrix}$$

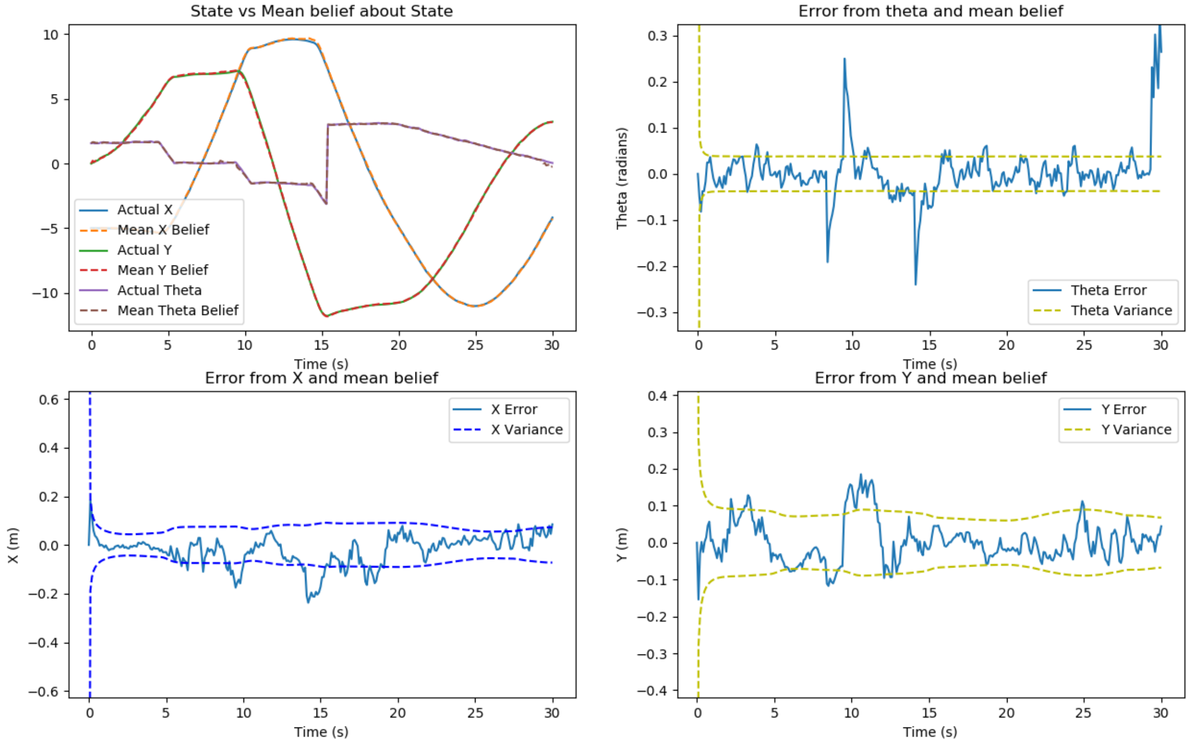
$$H_t = \begin{bmatrix} \frac{\partial r_t^i}{\partial \bar{\mu}_{t,x}} & \frac{\partial r_t^i}{\partial \bar{\mu}_{t,y}} & \frac{\partial r_t^i}{\partial \bar{\mu}_{t,\theta}} \\ \frac{\partial \phi_t^i}{\partial \bar{\mu}_{t,x}} & \frac{\partial \phi_t^i}{\partial \mu_{t,y}} & \frac{\partial \phi}{\partial \bar{\mu}_{t,\theta}} \end{bmatrix}$$

$$\boxed{H_t = \begin{bmatrix} \frac{-(m_{j,x}-\bar{\mu}_{t,x})}{\sqrt{q}} & -\frac{(m_{j,y}-\bar{\mu}_{t,y})}{\sqrt{q}} & 0 \\ \frac{m_{j,y}-\bar{\mu}_{t,y}}{q} & -\frac{(m_{j,x}-\bar{\mu}_{t,x})}{q} & -1 \end{bmatrix}}$$

2 Figures







3 Code

3.1 Filter

```
from math import cos, sin, atan2, exp, radians

import numpy as np
from tools.wrap import wrap

class EIF:
    def __init__(self, all_features):
        self.mean_belief = np.vstack((-5, 0, radians(90)))
        self.covariance_belief = np.eye(3)
        self.info_matrix = np.linalg.inv(self.covariance_belief)
        self.info_vector = self.info_matrix @ self.mean_belief
        self.Qt = np.diag((0.2**2, 0.1**2))
        self.Mt = np.diag((0.15**2, 0.1**2))
        self.all_features = all_features

    def prediction_step(self, vc, wc, change_t):
        prev_mean_belief = np.linalg.inv(self.info_matrix) @ self.info_vector
        prev_mean_belief = wrap(prev_mean_belief, index=2)
        theta = prev_mean_belief[2]
        # Jacobian of ut at xt-1
        Gt = np.array([
            [1, 0, -vc*sin(theta)*change_t],
```

```
            [0, 1, vc*cos(theta)*change_t],
            [0, 0, 1]])
        # Jacobian to map noise in control space to state space
        Vt = np.array([
        [cos(theta)*change_t, 0],
        [sin(theta)*change_t, 0],
        [0, change_t]])

        mean_belief = prev_mean_belief + np.array([
        [vc*cos(theta)*change_t],
        [vc*sin(theta)*change_t],
        [wc*change_t]
        ])
        mean_belief = wrap(mean_belief, index=2)

        Mt = self.Mt
        self.info_matrix = np.linalg.inv(
            Gt @ np.linalg.inv(self.info_matrix) @ Gt.T + Vt @ Mt @ Vt.T
            )
        self.info_vector = self.info_matrix @ mean_belief

    def measurement_step(self, feature_measurements):
        Qt = self.Qt
        for index, feature in enumerate(self.all_features):
            mean_belief = np.linalg.inv(self.info_matrix) @ self.info_vector
            mean_belief = wrap(mean_belief, index=2)
            f_x = feature[0]
            f_y = feature[1]
            mean_x = mean_belief[0]
            mean_y = mean_belief[1]
            mean_theta = mean_belief[2]
            # Range and bearing from mean belief
            q = (f_x - mean_x)**2 + (f_y - mean_y)**2
            h = np.array([
                [np.sqrt(q)],
                [np.arctan2((f_y - mean_y), (f_x - mean_x)) - mean_theta]]).reshape((2,1))
            h = wrap(h, index=1)

            measurement = feature_measurements[:, index].reshape((2,1))
            measurement = wrap(measurement, index=1)

            Ht = np.array([
                [-(f_x - mean_x)/np.sqrt(q), -(f_y - mean_y)/np.sqrt(q), np.array([0])],
                [(f_y - mean_y)/q, -(f_x - mean_x)/q, np.array([-1])]]).reshape((2,3))

            self.info_matrix = self.info_matrix + Ht.T @ np.linalg.inv(Qt) @ Ht
            self.info_vector = (
                self.info_vector +
                Ht.T @ np.linalg.inv(Qt) @
                (measurement - h + (Ht @ mean_belief))
            )
        self.covariance_belief = np.linalg.inv(self.info_matrix)
        self.mean_belief = self.covariance_belief @ self.info_vector
        self.mean_belief = wrap(self.mean_belief, index=2)
```

3.2 Simulation runner

```
from math import cos, sin, atan2, sqrt, pi, exp

import numpy as np
from matplotlib import pyplot as plt
from scipy.io import loadmat

from midterm.extended_information_filter import EIF
from heading_range_robot.robot_plotter_midterm import RobotPlotter
from tools.wrap import wrap


def main():
    data = loadmat('midterm/midterm_data.mat')
    # Unpack data
    true_state = data['X_tr']
    true_state = wrap(data['X_tr'], index=2)
    landmarks = data['m']
    w_c = data['om_c'][0]
    w = data['om'][0]
    t = data['t'][0]
    v = data['v'][0]
    v_c = data['v_c'][0]
    true_bearing = data['bearing_tr']
    true_range = data['range_tr']

    eif = EIF(landmarks.T)
    # Initialize plots
    robot_plotter = RobotPlotter()
    robot_plotter.init_plot(true_state, eif.mean_belief, landmarks.T)

    # Initialize history for plotting
    all_mean_belief = [np.copy(eif.mean_belief)]
    all_covariance_belief = [np.copy(eif.covariance_belief)]
    all_information_vector = [np.copy(eif.info_vector)]
    # Go through data
    for time_step in range(1, len(t)):
        t_curr = t[time_step]
        change_t = t[time_step] - t[time_step-1]

        eif.prediction_step(v_c[time_step], w_c[time_step], change_t)
        eif.measurement_step(np.vstack((true_range[time_step], true_bearing[time_step])))

        robot_plotter.update_plot(true_state[:, time_step], eif.mean_belief)

        all_mean_belief.append(np.copy(wrap(eif.mean_belief, index=2)))
        all_covariance_belief.append(np.copy(eif.covariance_belief))
        all_information_vector.append(np.copy(eif.info_vector))
    # Plot summary
    plot_summary(true_state, all_mean_belief, all_covariance_belief, t, all_information_vector)
```