CS 8803: AI For Robotics
Final Project - Spring 2015

Team Members:
Nick Lee (nicklee@gatech.edu)
Simon Cartoon (scartoon3@gatech.edu)
Sandip Agrawal (sandipagr@gatech.edu)
Cassie Wright, Jr. (cass@gatech.edu)

Algorithm Selection

The chosen algorithm is Extended Kalman Filter (EKF). This was chosen over standard Kalman Filter for its ability to predict curvilinear motion. The team members had experience implementing EKF in a previous assignment (Runaway Robot) and were very familiar with this approach.

Unscented Kalman Filter (UKF) was considered, but we chose not to use UKF because there were fewer resources available that documented a proper UKF implementation. This made it more challenging to implement the algorithm with confidence.

The team also considered Particle Filters, however these seemed less appropriate for the task at hand due to their focus on localization rather than prediction.

Program Details

Our program first parses the input text file and creates a matrix of data points in the form of [[X1, Y1], [X2, Y2]....]. From this data, we look for the minimum and maximum X and Y coordinates and set those measurements as the boundaries along X and Y - these boundaries provide an estimate of where the walls are in the video.

Our program begins iterating through each data point in the test data matrix and feeds them into an EKF algorithm described below:

Discrete-time predict and update equations [edit]

Predict [edit]

Predicted state estimate
$$\hat{\boldsymbol{x}}_{k|k-1} = f(\hat{\boldsymbol{x}}_{k-1|k-1}, \boldsymbol{u}_{k-1})$$

Predicted covariance estimate
$$\boldsymbol{P}_{k|k-1} = \boldsymbol{F}_{k-1}\boldsymbol{P}_{k-1|k-1}\boldsymbol{F}_{k-1}^{\top} + \boldsymbol{Q}_{k-1}$$

Update [edit]

Innovation or measurement residual
$$\tilde{\boldsymbol{y}}_k = \boldsymbol{z}_k - h(\hat{\boldsymbol{x}}_{k|k-1})$$

Innovation (or residual) covariance
$$\boldsymbol{S}_k = \boldsymbol{H}_k\boldsymbol{P}_{k|k-1}\boldsymbol{H}_k^{\top} + \boldsymbol{R}_k$$

Near-optimal Kalman gain
$$\boldsymbol{K}_k = \boldsymbol{P}_{k|k-1}\boldsymbol{H}_k^{\top}\boldsymbol{S}_k^{-1}$$

Updated state estimate
$$\hat{\boldsymbol{x}}_{k|k} = \hat{\boldsymbol{x}}_{k|k-1} + \boldsymbol{K}_k\tilde{\boldsymbol{y}}_k$$

Updated covariance estimate
$$\boldsymbol{P}_{k|k} = (\boldsymbol{I} - \boldsymbol{K}_k\boldsymbol{H}_k)\boldsymbol{P}_{k|k-1}$$

where the state transition and observation matrices are defined to be the following Jacobians

$$\boldsymbol{F}_{k-1} = \left.\frac{\partial f}{\partial \boldsymbol{x}}\right|_{\hat{\boldsymbol{x}}_{k-1|k-1}, \boldsymbol{u}_{k-1}}$$

$$\boldsymbol{H}_k = \left.\frac{\partial h}{\partial \boldsymbol{x}}\right|_{\hat{\boldsymbol{x}}_{k|k-1}}$$

EKF Algorithm (source: http://en.wikipedia.org/wiki/Extended_Kalman_filter#Predict)

EKF modeling with update and predict steps:

a. Initialize initial uncertainty matrix (P) with the following assumptions:
    i. High uncertainty in robot heading (theta)
    ii. Lower uncertainty in x,y measurements (we assume centroid data is generally reliable)
b. Initialize matrices for basic Kalman filter:
    i. Identity matrix for 2-dimensional motion (I)
    ii. Measurement noise matrix (R)
c. Execute prediction step with the movement model. Here is our movement model:

*delta_x = robot_travel_distance * cos(robot_heading)*
*delta_y = robot_travel_distance * sin(robot_heading)*

next_X_coordinate = target_measurement[0] + delta_x
next_Y_coordinate = target_measurement[1] + delta_y

    i. Initialize EKF specific matrices:
        1. Jacobian of State Transition Model (F)
        2. Jacobian of Measurement Function (H)

ii. Incorporate a ‘centering affinity’ of 0.93. We assume, based on empirical tuning, that the robot will reduce it’s turning angle by ~7% toward centerline motion at each step due to drift

d. Execute measurement update, storing produced x,y coordinates in ‘observations’

Now that we have the EKF model, we go about predicting the next 60 measurements as follows:

The prediction of robot location involves using the movement model used in the EKF prediction step (see 1c above), along with a conditional tuning of the robot’s future movement using a naive ‘billiard ball’ motion. Between each iteration, we also apply robot drift which tends to reduce the turning angle by ~7%.

- The billiard ball motion is applied whenever we detect a wall collision (incorporating output of the boundary helper function created in 1c. above). Although casual observation of the video shows there are many kinds of possible collisions, we assume that a true reflection occurs (angle of incidence equals angle of reflection)
- Reflection model is tuned to each kind of wall collision (left, right, bottom, top) to ensure the correct reflection occurs
  For collisions with either side wall, the angle is (Π - Θ).
  For collisions with the top or bottom wall, the angle of reflection is (2Π - Θ).

The prediction algorithm described above is executed for 60 frames to produce the final dataset which is outputted to text file as “prediction.txt”.