CSE 298 – Foundations of Robotics

Homework 4

Due: August 6 by end of day

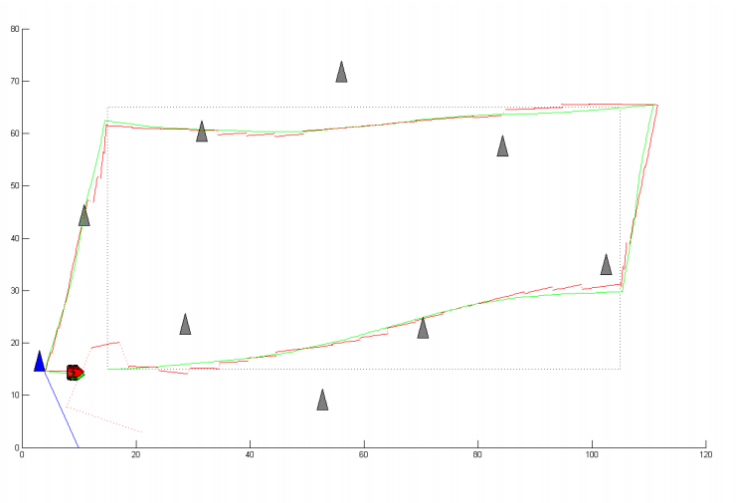

Figure 1: Sample EKF Simulation

A. Objectives: The goal of this assignment is to implement an EKF that will be used to localize a mobile robot. We assume the robot is traveling through a sensor network. Each sensor node in the network knows its own position. Furthermore, each has a camera with which they are able to detect the relative bearing of the robot. Note they are not able to estimate the distance to the robot. Your objective is to put together an EKF that can fuse the bearing measurements reported by the different sensor nodes with the robot odometry to estimate its pose (position and orientation) over time.

B. Robot Model:

1. Motion Model: Again, we are using a differential drive kinematic model. However, we again will assume that there is uncertainty in the robot motion model.
   a) Assume that both the linear and angular velocities of the robot are to independent random noise. Values of 0.1 m/s for the standard deviation are reasonable.
   b) Assume a constant linear robot velocity of 5 m/s
   c) Assume the update rate for the odometry is 10Hz.
2. Sensor Model (See Skeleton .m file for more details):
   a) Cameras are created at a position taken from mouse input via the Matlab ginput function

```
camera(i) = make_camera( x, y, max_range, 'r' );
```

   where (x,y) denote the camera locations, max_range denotes the range at which the camera becomes visible to the robot, and ‘r’ denotes the color

red in this example. What is returned is a camera struct that contains all of the relevant information needed to draw (and redraw) the camera.

b) We assume that the cameras detect the robot, and transmit its relative bearing only. We further assume that the robot knows their position in the map. Cameras must be tested to determine if they are in range via the

```
[ camera, bearing ] = test_camera( camera, robot );
```

function where the arguments are a camera struct, the robot struct, and the returned values are the bearing from the camera to the robot, and the potentially modified camera struct (e.g., if it changed color). If the robot is out of range, the camera returns [] (empty set) for bearing.

c) You will need to implement both the make_camera and test_camera functions (take a hint from how make_robot and move_robot are implemented).

d) Cameras that are within range are plotted as blue, and with a line from the camera to the robot. Cameras that are not within range of the robot are plotted as grey ([0.5 0.5 0.5]).

e) Assume that the robot can detect cameras at a distance of 20 meters (max_range). Cameras outside the range not visible. We assume the cameras are omnidirectional, so you need not worry about field of view constraints.

f) Further assume that the bearing measurements are corrupted with Gaussian noise, and $\sigma_{bearing}$ = 3-5 degrees.

g) Assume a sensor update rate of 1 Hz.

C. The Assignment:

1. To understand this assignment, up-front you must realize that there are two "robots." The actual robot (`robot`), and the robot's belief (`robot_hat`). The simulator knows the actual robot's position, orientation, and velocities perfectly. The robot does not, so it tries to estimate these with the EKF (its belief). You must clearly understand this distinction to do this assignment. If there is any confusion, see me with questions before starting.
2. There is a skeleton .m file that everyone MUST use as the basis for their homeworks. Exceptions will not be accepted. Download this from the assignment repository.
3. Initial values for the sensor/motion model parameters are included in this .m file. You may modify these as you deem necessary.
4. The robot is initially placed at position 15, 15 with orientation = 0.
5. The robot's belief in its initial position is corrupted as reflected in the .m file.
6. The true robot is plotted as green (what the simulator knows).
7. The robot's belief (from the EKF) is plotted in red.

8. At all times in addition to the robot's belief, you must also plot an ellipse around the robot expected position that reflects 3 standard deviations in positional uncertainty (don't worry about its orientation). You can obtain this from the covariance matrix ***P***. To do this, you will write a function

   ```
   h = plot_cov( mu, P )
   ```

   which takes as arguments the mean position and covariance matrix P and plots the appropriate ellipse while returning the graphics handle. HINT: You can use the eig function in Matlab to obtain a decomposition of the covariance matrix that will yield the ellipse's size as well as its orientation.
9. During the measurement update rate, cameras look for the robot. Those that detect the robot (within range) will be used serially in the `MeasurementUpdate` phase. So, if three cameras can see the robot, there will be three calls to `MeasurementUpdate`.
10. Since we are measuring bearing to the robot only, your measurement equation for the EKF should be of the form.
    `bearing = atan2( robot.y-camera.y, robot.x-camera.x );`
    Note this assumes a perfect measurement. You will need to corrupt this with noise when returning a bearing measurement from the camera.
11. Assume that a camera that cannot view the robot responds with a bearing of [].
12. During each call to `MeasurementUpdate`, we not only calculate the new mean and covariance for the robot, we also replot the corresponding robot pose and uncertainty ellipse.
13. The time update reflects the motion portion of the EKF. During each step, you recalculate the robot's belief AND actual robot position. NOTE: The actual robot's motion is corrupted by noise and the simulator knows these values. However, the robot's belief (robot_hat) is NOT corrupted by noise (as the noise values are not known to the robot – only to the simulator).
14. Take a look at the call to TimeUpdate. You will notice that there are 2 additional arguments `v_leg` and `omega_leg`. These correspond to the current robot linear and angular velocities, respectively.

```
[robot, robot_hat, P] = TimeUpdate(robot, robot_hat, P, v_leg, omega_leg);
```

15. The goal of your robot is NOT to travel around aimlessly. Instead, it attempts to follow the rectangular path outlined in Figure 1. There are 8 "legs" to this path. They are:
    a) Travel from (15,15,0) to (105,15,0)
    b) Travel from (105,15,0) to (105,15,π/2)
    c) Travel from (105,15,π/2) to (105,65,π/2)
    d) Travel from (105,65,π/2) to (105,65,π)
    e) Travel from (105,65,π) to (15,65,π)

f) Travel from (15,65,π) to (15,65,3π/2)
g) Travel from (15,65,3π/2) to (15,15,3π/2)
h) Travel from (15,15,3π/2) to (15,15,0)

16. At the conclusion of traveling these legs the robot would have returned to its initial position in an ideal world. From Figure 1, we see the world is not ideal.
    a) The robot travels each leg of the path open loop.
    b) Linear and angular velocities for each of the legs are (5,0) and (0,π/4) depending on whether the robot is translating or rotating, respectively.
    c) Only when it is within some tolerance of the objective pose for each leg will it transition to the new leg.
    d) NOTE: Transitions are based on the robot's BELIEF of its pose – NOT the actual pose as this is unknown to the robot.
17. You are to plot the paths that the actual robot takes, as well as the robot's belief.

D. Submission: You are to submit:

1. A video of your simulation working.
2. Your Matlab source code.