MAE 4180/5180, ECE 4772/5772, CS 3758

Autonomous Mobile Robots

Homework 4 - due Fri. March 26 by 11:59PM
(available until Sun. March 28 by 11:59PM)

A document containing the answers and figures should be uploaded to Gradescope as `HW4writeup.pdf`. All the code and the files needed to run the code are to be uploaded to Canvas as a zip file named `HW4code.zip`. Note, the code zip file should contain ONLY .m files, config files and files containing data (if needed) and **not any other file** (writeup, assignment, simulator, toolboxes etc). Specific functions, as indicated below, should be uploaded to their corresponding assignments on **Canvas**.
Notes about autograded assignments:

- We **highly** recommend you develop and debug the autograded functions in Matlab. The error messages that the autograder provides are not as explicit as in Matlab
- Before submitting, we **highly** recommend running your own test case with "Run Function" as it can reveal syntax errors, missing functions, etc. when transitioning to Canvas.
- You may submit as many times as you like until the deadline
- Make sure to also include the functions in the zip file you upload to Canvas
- **Reusing code:** We encourage you to reuse your prior functions in the new functions. For the autograded assignments, if you want to use a previous function you have written (for example robot2global), simply copy-paste it below the function you are writing

Dynamics and measurement functions (40 points)

In lab 2, you will be localizing the robot using different filters and different measurements. In this section you will set up the required dynamics and measurement functions.

1. Given that the control action/inputs are the odometry information (distance traveled, angle turned), what function would you use as $g(x_{t-1}, u_t)$? Explain. (Hint: you already wrote the function in Homework 2).
2. Given that the measurement is a depth measurement, what function would you use as $h(x_t)$ to predict the measurement? Explain. (Hint: you already wrote the function in Homework 2).
3. Given that the measurement is "GPS" like information (given by the overhead localization/true pose, i.e. $[x, y, \theta]$), explain and write a function `hGPS.m` that predict the measurement (This should be very simple). Submit this function in the autograded assignment `Homework4 hGPS.m` on Canvas.
4. In order to use an EKF, one needs to calculate the Jacobian of the update and measurement functions at a given state. Write out the Jacobian $G_t = \frac{\partial g}{\partial x_{t-1}}$. Write the function `GjacDiffDrive.m` to output G_t for a particular pose x. Submit this function in the autograded assignment `Homework4 GjacDiffDrive.m` on Canvas.
5. Write the file `HjacDepth.m` to output $Hdepth_t$ at a particular x_t. Explain how you calculate this. (Hint: differentiating the function is difficult, think about using finite differences instead). Submit this function in the autograded assignment `Homework4 HjacDepth.m` on Canvas.
6. Write out the Jacobian $HGPS_t = \frac{\partial h}{\partial x}$. Write the function `HjacGPS.m` to output $HGPS_t$ at a particular x_t (Again, this should be VERY simple). Submit this function in the autograded assignment `Homework4 HjacGPS.m` on Canvas.

Extended Kalman Filter(15 points)

In the lab, you will be using the EKF with different measurements. To make switching between measurement types easy, in this section you will write a generic EKF that takes as input the measurement and update functions and as such, can be reused.

Write the file `EKF.m` to perform one prediction & update step of the EKF. To make this file very generic, the functions from the previous section should be inputs to this function and should be passed as anonymous functions (details at the end of the problem set).

What are the inputs to the function `EKF`? what are the outputs?
This function is evaluated in the autograded assignment `Homework4 testEKF.m` on Canvas, where you will have to call your filter to produce the estimated location at time t given all the information in time $t-1$ (i.e. one time step).

Particle Filter(15 points)

Same as the EKF, it is best to write a generic particle filter that can be reused later on.

Write a function `PF.m` that takes in an initial particle set, odometry and depth measurements, prediction and update functions and outputs a new particle set.

Assume there is an infinite wall at $y = 1$, the robot moved one meter forward ($d = 1, \phi = 0$), and all 9 depth measurements are 2 meters. Create an initial particle set of size 30 with uniform distribution in $x \in [-1, 1]$, $y \in [-3, 1]$ and $\theta = 0$, plot the wall, the initial particle set (x,y position only for each particle), and the particle set after one iteration of the particle filter on the same plot. Use one marker shape for the initial particles and another marker shape for the final particles.

Running the filters in the simulator (115 points)

1. Write a function `motionControl.m`. This function should drive the robot in different directions while reacting to the bump sensor (so that the robot does not try to drive through a wall). The control should be independent of the location and deterministic. Furthermore, this function should:
   (a) Call the sensor information (using the function `readStoreSensorData.m` provided in HW 1)
   (b) Call `integrateOdom.m` and store the dead reckoning information in `dataStore.deadReck`
   (c) Have a **clearly identifiable** and **well commented** section describing how to switch between the three methods described below (EKF with GPS data, EKF with depth data, PF with depth data). You will need to be able to switch between these three methods for lab 2. See Questions 4, 5, and 6 for more details.
2. In `motionControl.m`, create new fields `datastore.ekfMu` and `datastore.ekfSigma` (robot pose mean and covariance, respectively).
3. Initialize `dataStore.deadReck` and `datastore.ekfMu` to the true initial pose (from `dataStore.truthPose`) and `datastore.ekfSigma` to $[2, 0, 0; 0, 2, 0; 0, 0, 0.1]$. Define a new variable `R` (process noise covariance matrix) and set it to 0.01I. Define a new variable `Q` (measurement noise covariance matrix) and set it to 0.001I. What are the dimensions of `R` and `Q`?
4. **EKF with GPS data**:
   (a) Create noisy GPS measurement by adding gaussian noise to the true pose. What should the noise distribution be? Record this data in `dataStore.GPS`.
   (b) In the simulator, load `cornerMap` and place the robot somewhere near $[-4; 2; 0]$. Create and load a config file with noise on the depth measurements and the odomerty. What values did you choose for the noise? Make sure the values make sense with respect to `R` and `Q`.
   (c) In `motionControl.m`, call the EKF with the **noisy GPS data** and the appropriate functions (dynamics and measurement).

(d) Run `motionControl.m` in the simulator. On the same figure, plot the map and three final trajectories (x,y): `dataStore.truthPose` (the data from overhead localization), `dataStore.deadReck` (from integrating the odometry only), and `datastore.ekfMu` (estimate from the EKF). Also plot the 1-Σ ellipses from `datastore.ekfSigma`. Note, you only need to plot the x,y position and not the orientation.(You may use the provided function `plotCovEllipse.m` or any other function that plots ellipses. If you do, make sure to cite the source).

(e) Place the robot somewhere near $[-4; 2; 0]$ again. Repeat 4(b–d) using a different initialization for the filter (we want to see what happens to the estimate if the initial belief is different): $\mu_0 = [-2; -1.5; \pi/2]$ and $\Sigma_0 = [4, 0, 0; 0, 4, 0; 0, 0, 0.02]$. (Initialize `datastore.deadreckon` to the same initial pose estimate μ_0 here in (e).)

(f) Compare the results of 4(d) and 4(e) - How does the initial estimate affect the robot's estimate?

5. **EKF with depth data**: In `motionControl.m`, call the EKF with the **depth data** and the appropriate functions (dynamics and measurement). Make sure to **write comments** in `motionControl.m` on how to switch between using the GPS and depth measurements.

   (a) Repeat 3 and 4(b,d). Make sure the dimension of `R,Q` are correct.

   (b) Repeat 5(a) using `R`= 0.001I and `Q`= 0.01I.

   (c) Compare the results from 5(a) and 5(b). How do the process and measurement noise covariances affect the robot's estimate?

   (d) Does the filter behave as you'd expect? Why or why not?

   (e) What are some of the problems with using an EKF with depth measurements?

   (f) What errors do you expect to see when you run the EKF on the physical robot in the lab?

6. **PF with depth data**: In `motionControl.m`, create new field `datastore.particles`. Generate an initial particle set of size 20, with x-location sampled from a uniform distribution between $[-5, 0]$, y-location sampled from a uniform distribution between $[-5, 5]$, and θ sampled from a uniform distribution between $[-0.2, 0.2]$. Initialize the particle weights to be uniform. Call the PF with the **depth data** and the appropriate functions (dynamics and measurement). Make sure to **write comments** in `motionControl.m` on how to switch between using the EKF and using the PF.

   (a) In the simulator, load `cornerMap` and place the robot somewhere near $[-4; 2; 0]$. Load your noisy config file and run `motionControl.m`.

   (b) Plot the initial particles on the map (plot the x,y positions as well as headings).

   (c) Plot the final particles on the map (x,y position only).

   (d) For each time step, choose the particle with the highest weight and plot it. You will get the "best" trajectory. Do this for 5 additional particles and plot the 5 trajectories on the same map. Also plot the true trajectories (from `datastore.truthPose`).

   (e) Repeat 6(a–d) using a particle set of size 500.

   (f) Does the filter behave as you'd expect? Why or why not?

   (g) How does the size of the particle set affect the robot's estimate?

   (h) How else could you represent the robot's estimate (other than just looking at the highest weight particle)? Plot the trajectory of that estimate. Is it closer to the truth?

Comparing Filters (30 points)

Fill out the following table:

Filter	Assumptions about the system	This filter is appropriate when	Advantages	Disadvantages
Kalman filter				
Extended Kalman filter				
Particle filter				

Using functions as inputs

As previously mentioned, for re-usability, we can send the dynamics, measurement and Jacobian functions as parameters to our filter. There are two ways to do this, but only one that is accepted with Autograder on Canvas. Below is a simple example and link for extra information (assume `robot2global` is defined elsewhere):
Anonymous functions -

```
r2g = @(pose, p_xy) robot2global(pose, p_xy);
MyComplicatedFilter(r2g)

function MyComplicatedFilter(converter)
    % some calculations to get my_pose, my_xyR
    y = converter(my_pose, my_xyR);
    return y
```

More information: `https://www.mathworks.com/help/matlab/matlab_prog/anonymous-functions.html`