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

Autonomous Mobile Robots: Lab #2

# Localization

*Instructor:*
Dr. Hadas Kress-Gazit

# Objectives

In this lab, students will localize a mobile robot using three different localization algorithms:

- Dead reckoning
- Extended Kalman filter
- Particle filter

This lab is designed to give some understanding of how localization works and how various factors can contribute to the difficulty of this task. Using various onboard sensors and a known map, the goal is to estimate the robot's pose $(x, y, \theta)$. Accurate pose estimation is critical for most mobile robot tasks, and choosing the "best" (or most appropriate) localization algorithm depends on the specific application.

# Required Code

- `feedbackLin.m`
- `readStoreSensorData.m`
- `limitCmds.m`
- `backupBump.m`
- `visitWaypoints.m`
- `integrateOdom.m`
- `GjacDiffdrive.m`
- `depthPredict.m`
- `intersectPoint.m` (if used for depthPredict)
- `hGPS.m`
- `HjacGPS.m, HjacDepth.m`
- `EKF.m`
- `PF.m`
- `motionControl.m`

# Required plots (to be shown to the TAs at the beginning of the lab)

- A plot of the truthPose, deadReck and ekfMu trajectories for a run of the simulator with GPS data (can be the same plot as in 4(d) in HW 4)
- A plot of the truthPose, deadReck and ekfMu trajectories for a run of the simulator with depth data (can be the same plot as in 5(a) in HW 4)
- A plot of the truthPose and the "best" particle trajectories for a run of the simulator with depth data (can be the same plot as in 6(d) in HW 4)

# 1 Lab Manual

## 1.1 Set-up - Remote student

(a) Join the lab Zoom session (link on Canvas).

(b) Open the Twitch stream: https://www.twitch.tv/cu_mae_amr

(c) (One remote group member) create and share a Google drive or Box folder with the rest of the group. Add subfolders for each group member to put their code in.

(d) Put all the required files in the shared folder.

(e) Put the file `lab2WallMap.mat` from Canvas in the shared folder.

## 1.2 Station Set-up - In-Person student

(a) Join the lab Zoom session (link on Canvas) on the **lab computer** and share your screen.

(b) Open the shared folder created by one of the online group members. Add your files.

(c) Create a local folder on the lab computer and copy all the files there.

(d) Open Matlab and change the working directory of Matlab to be the folder that contains your files. **Make sure to periodically save data to your online folder.**

(e) Unplug your iRobot Create. Make sure to only pick up the robot from the *bottom* and not the sensor platform. **Be careful not to hit the markers on the top of the robot.** Put it on the floor next to your lab station.

(f) Take note of the name of your robot.

(g) Turn on the robot by pressing the power button. The Raspberry Pi takes about 20-30 seconds to boot.

(h) In your MATLAB Command Window, run `Robot = CreatePiInit('robotName')` where `robotName` is the robot name from step (f). The initialization process creates the variable `Robot`, which contains the port configurations for interfacing with the Create and the added sensors and the robot name; it has five elements:

- `Robot.Name` contains the robot name.
- `Robot.OL_Client` used for getting the robot pose ground truth from the Optitrack system.
- `Robot.CreatePort` used for sending commands and reading sensors from the robot.
- `Robot.DistPort` used for getting depth information from the realsense camera.
- `Robot.TagPort` used for getting tag information from the realsense camera.

(i) Check that you have connected to the robot properly by running `BeepRoomba(Robot.CreatePort)`. You should hear your robot beep.

(j) Put your robot on the field. Make sure you can access the robot's localization information by running `[x,y,theta] = OverheadLocalizationCreate(Robot)`.

(k) Put the robot in front of a tag, at least one foot away. Run `RealSenseTag(Robot.TagPort)` and `RealSenseDist(Robot.DistPort)`. Make sure you can get tag and depth information.

(l) If any of steps i-k fail, disconnect from the robot (run `CreatePiShutdown(Robot)`), shut it down, close Matlab, restart Matlab, turn the robot on and go back to step h.

(m) Assume that the **Realsense offset** (the location of the sensor in the robot-fixed frame) is (0,8cm), i.e. the x-axis offset is 0 and the y-axis offset is 8 cm.

**Important:**

- If the control function exits with an error, make sure to stop the robot by typing in the command window: `SetFwdVelAngVelCreate(Robot.CreatePort, 0, 0)`
- When you are done working with the robot, or you wish to restart Matlab or the connection to the robot, first run `CreatePiShutdown(Robot)` to disconnect properly from the robot.

## 1.3 Extended Kalman Filter

(a) Load the map `lab2WallMap.mat` into the Matlab workspace. The variable `map` contains the coordinates of the walls, where each row specifies the endpoints of a wall: $[x_1, y_1, x_2, y_2]$.

(b) Configure your control program in the following way:

- Utilize deterministic controls that are independent of the robot pose (`backupBump.m` for example, NOT `visitWaypoints.m`).
- Compute a dead reckoning pose estimate.
- Compute an extended Kalman Filter pose estimate using depth sensors. How are you dealing with the delays in the depth information?

(c) Place your robot on the field: place it such that it is at least one foot away from all walls. Initialize both the dead reckoning pose (`dataStore.deadReck`) and filter estimate (`dataStore.ekfMu`) to the true robot pose (by calling overhead localization) . Set the initial extended Kalman filter covariance (`dataStore.ekfSigma`) to $[0.05, 0, 0; 0, 0.05, 0; 0, 0, 0.1]$.

(d) Set `R` (process noise covariance matrix) and `Q` (measurement noise covariance matrix) to "reasonable" values (For example, for depth, you could use the $\sigma$ you estimated for your Lab #1 post-lab assignment and assign the diagonal elements of `Q` to be $\sigma^2$)

(e) We **highly recommend** you plot the robot's pose estimate from `dataStore.ekfMu` and 1 $\sigma$ ellipse from `dataStore.ekfSigma` in real-time, to make sure your code is working as expected (although it's not required).

(f) Run your control program for 1–2 minutes. Make sure to use the `limitCmd.m` function with a reasonable value. If you are planning to do the extra credit Beacon EKF, make sure the robot sees a few beacons as it is moving.

(g) Access the sensor data in the MATLAB Command Window by typing: `global dataStore;` Make sure all the sensor data has been properly recorded, and save (make sure to save `R` and `Q` as well). (In `dataStore`, you should have non-empty fields: `truthPose`, `rsdepth`, `odometry`, `bump`, `beacon`, `deadReck`, `ekfMu` and `ekfSigma`).

(h) Repeat (b-g) for each member of your group, choose different values for `Q` (measurement noise covariance matrix) for each member.

(i) EKF with "GPS" data : re-configure your control program such that the EKF uses GPS (noisy truthPose) instead of the depth sensor. Repeat (b-g) to collect data for a run with GPS data. You only need to run this filter once.

## 1.4 Particle Filter

(a) Configure your control program in the following way:

- Utilize deterministic controls that are independent of the robot pose (`backupBump.m` for example, NOT `visitWaypoints.m`).
- Compute a dead reckoning pose estimate.
- Compute a particle filter pose estimate using depth sensors and 50 particles. How are you dealing with the delays in the depth information?

(b) Place your robot on the field: place it such that it is at least one foot away from all walls. Initialize both pose estimates (deadreckon and particle filter) to the true robot pose. Initialize every particle in the filter to the same pose.

(c) We **highly recommend** you plot the particles in real-time to make sure your code is working the way you think it should (although it's not required).

(d) Run your control program for $\sim 2$ minutes.

(e) Access the sensor data in the MATLAB Command Window by typing: `global dataStore;` Make sure at least 60 time stamps have been recorded for the particle filter (i.e. the filter ran at least 60 times), **Depending on how efficiently your particle filter runs, you may need to run the program for a longer period of time**.

(f) Make sure all the sensor data has been properly recorded, and save to file (be sure to save `R` and `Q` as well). In `dataStore`, you should have non-empty fields `truthPose`, `rsdepth`, `odometry`, `bump`, `deadReck` and `particles`.

(g) Repeat for each member of your group with a different particle set size (e.g. 100, 200, 500).

## 1.5 Navigate while Localizing - Optional

In the final competition, you will need to localize while moving. If you have time at the end of the lab, try running one of the filters while trying to reach points in the workspace space.

(a) Configure your control program in the following way:

- Change your control function to `visitWaypoints.m`).
- Decide on two waypoints the robot should reach.
- Compute a localization solution based on one of the previous sections (i.e. EKF with depth or PF with depth. You can also try EKF with GPS but that information will not be available at the final competition).
- Make sure your calculation of the control is using the localization solution from the filter and NOT truthPose!

(b) Place your robot on the field: place it such that it is at least one foot away from all walls. Initialize both the dead reckoning pose (`dataStore.deadReck`) and filter estimates. Choose appropriate noise parameters.

(c) We **highly recommend** you plot the robot's pose estimate in real-time, to make sure your code is working as expected.

(d) Run your control program until you reach the waypoints, or your robot looks hopelessly lost. Make sure to use the `limitCmd.m` function with a reasonable value.

## 1.6 Clean up

When you are done collecting all the data and after you make sure you have everything that you need for the lab report:

(a) Run `CreatePiShutdown(Robot)` to disconnect from the robot.

(b) Turn off the robot, return it to the lab station and plug it in.

(c) Make sure to leave your lab station clean, without any papers or other items. Follow COVID-19 protocols for sanitizing your station.

---

# 2 Post-Lab Assignment

Remember to submit your assignment as a group on **Gradescope**. To form a group:

1. **One** individual from the group submits the PDF on Gradescope.
2. When you click on your submitted assignment, there will be a section denoted by "Group" below the assignment name. Click on "View or edit group."
3. Add student(s) by selecting their name(s) from the drop-down menu under "Add Student."

## 2.1 Time delays (10 points)

1. In the lab, what was the longest delay you encountered with the depth information?
2. In the following sections you will plot and analyze your localization solutions. How are you taking the delay in depth information into account when performing pose estimation?

## 2.2 Extended Kalman Filter (80 points)

(a) Plot each run from Part 1.3 (one run for GPS, one run per group member for depth, each run in a different figure). On the same plot display the environment walls, the true robot pose (from `dataStore.truthPose`), the integrated robot pose (from `dataStore.deadReck`), and the estimated pose (from `dataStore.ekfMu`). Also plot the 1 $\sigma$ ellipse (for x/y pose only) from `dataStore.ekfSigma` – you can plot this at only a few points (rather than every point) so it is easier to visualize.

(b) For the runs using depth information what did you observe? Was the EKF able to maintain a reasonable pose estimate? Why or why not?

(c) For the runs using GPS what did you observe? Was the EKF able to maintain a reasonable pose estimate? Why or why not?

(d) Choose one run with the depth information. Using the same initialization, R and Q, for each team member estimate the pose of the robot over time using their EKF code (you should run the same data with the different EKF implementations). Plot the estimated trajectories and the truth pose on the same figure. Were all trajectories the same? why or why not?

(e) Choose one run with the depth information. Initialize the EKF estimate to an incorrect initial pose. Run the EKF again with the recorded data. What do you observe? Was the robot ever able to recover its true pose? Why or why not?

(f) For the run with the GPS information, initialize the EKF estimate to an incorrect initial pose. Run the EKF again with the recorded data. What do you observe? Was the robot ever able to recover its true pose? Why or why not?

(g) How did your choice of `Q` affect your pose estimate? Was this expected?

(h) How does the EKF estimate compare to dead reckoning? Is this what you expected?

(i) What can you say about the strengths and weaknesses of dead reckoning? Under what circumstances might dead reckoning be the best choice for localization?

(j) What can you say about the strengths and weaknesses of the EKF? Under what circumstances might an EKF be the best choice for localization?

(k) What differences did you observe between the simulated results from the homework and those from the physical robot? Is this what you expected?

## 2.3 Particle Filter (40 points)

(a) For each member's data, plot the initial particle set, final particle set, true robot trajectory, and the environment walls.

(b) For each member's data, choose the final particle with the highest weight and plot it's trajectory. How does it compare to the true trajectory?

(c) In the homework you were asked to describe a different method for extracting a single pose estimate from a particle filter than simply taking the highest weighted particle. Describe your method and plot the corresponding trajectory. How does this compare to the true trajectory? Does this method perform better than choosing the highest weighted particle?

(d) Assume that the final particle set can be represented by a single Gaussian distribution (just $x$ and $y$, ignore $\theta$). Plot the particles and the Gaussian pdf (using `mesh.m`, `contour.m` or `surf.m`). How well does a single Gaussian represent the final particle set?

(e) What are the strengths and weaknesses of using a particle filter? Under what circumstances might a particle filter be the best choice for localization?

(f) How well did the particle filter estimate the true robot trajectory, compared to the EKF? Why?

(g) What differences did you observe between the simulated results from the homework and those from the physical robot? Were these differences expected?

## 2.4 Beacon EKF (Extra Credit - 50 points)

(a) Adjust your EKF function to use Beacons as the measurement. This will involve writing a function `hBeacon.m` to predict the location of a beacon, given a position in the map and a set of known beacon positions. The beacon positions are provided in `lab2Beacon.mat` as the variable `beaconLoc` (each row gives $[x, y, tagNum]$ for a single beacon). You will also need to find the Jacobian (either analytically or using finite differences) $Hbeacon_t$ at a particular $x$. Explain how your group calculated $h(x_t)$ and $Hbeacon_t$.

(b) Using the stored data (specifically, the odometry and beacon fields) from one of your EKF runs in the lab, initialize the estimated position to the truth pose and run your beacon EKF on the stored data to obtain a series of estimated pose values. What values did you use for `Q` and `R`? Justify your choices.

(c) Plot your estimated pose from the beacon EKF, along with the estimated pose from your original EKF (using either GPS or depth, depending on which data set you chose), and the "true" pose of the robot. For the final pose estimate, plot the $1\sigma$ ellipse for the beacon EKF and for the original EKF (only do this for the final time-step).

(d) Comment on the differences in performance for the two different EKFs. Which performed better, and why? How would changing your `Q` matrix affect the results?

## 2.5 Offline Particle Filter (Extra Credit - 15 points)

(a) Using one of the data sets obtained for particle filter in the lab (specifically, the stored data for depth and odometry), re-run your particle filter with a much larger number of particles (e.g. 5000, 10000), and obtain a new series of particle estimates. Plot the new position estimates (based on the highest weighted particle at each time step), along with the estimate from your original particle filter in the lab (again, based on the highest weighted particle), and the "true" position of the robot (from `dataStore.truthPose`).

(b) Comment on the differences between the two runs of the particle filter. Did one run significantly slower than the other? Why or why not? Did one produce a better estimate than the other? Why or why not?

## 2.6 Lab preparation (mandatory, not graded)

(a) Which robot did you use?

(b) For the code you wrote as part of Homework 4, were there any coding errors that were not discovered when using the simulator?

(c) How much time did you spend debugging your code in the lab?

(d) Did you observe any behaviors in the lab that you did not expect, or that did not match the simulated behaviors?

(e) What was the contribution of each group member to the lab and the report?