Applied Estimation (EL2320)

Lab 2

Particle Filter

Introduction

This lab consists of two parts:

1. A series of preparatory questions that we will go over in class
2. Two separate MATLAB exercises
   - A preparatory case study with the particle filter where you learn more about the behavior of the particle filter. Similar to the warm-up part of lab 1, very little extra code is needed.
   - The main problem in which you need to complete an implementation of a Monte Carlo Localization (MCL) algorithm.

It is suggested to go through the lecture notes again before you start this lab. If you are using the recommended course book, it is a good time to read chapter 4 to do a recap about the particle filter and chapter 8 to further familiarize yourself with the MCL problem. In order to pass this lab you need to

1. Provide written answers to the preparatory questions, Part I.
2. Follow the instructions given for the warm-up particle filter problem
3. Write the code to complete the (incomplete) functions in the MCL problem
4. Provide written answers to the questions in the sections 1 and 2
5. Upload your (**working**) code along with sample plots at illustrative times during the runs on each data set. The plots should be analysed in the report with a short text on each that explains why it looks the way it does compared to the other plots. Note that you do **not** need to upload the datasets, test cases or warmup code.

The answers to most questions are 1 (sometimes up to 3) sentences per part. So a question that asks 'What is A and why is B?' can be answered in two sentences most of the time. Being concise and correct (or wrong) will get you better feedback than writing a few paragraphs that are hard to follow.

There will be help sessions for the lab. You need to do the readings and answer the questions in Sections 1 and 2 (as many as you can) before you start to write the code and attend the help sessions.

PART I - Preparatory Questions

These questions are intended to encourage you to think about what we have covered so far on the Particle filter and localization.

You should write out your answers to the questions neatly and legibly and bring them to class on the assigned day. Be sure to include your name. It is advised that you go ahead and use a pdf document that you then print out for class as you will need to upload these answers later. This will be graded and give bonus points as in lab 1. You will then be able to correct the answers before uploading the lab report to the course web. In other words, getting all the answers correct for class is not necessary, but participation is of great benefit to you.

As you see, the number of questions is less than on the first lab as you are all now familiar with the basic problem of estimation and localization from the first lab. However, there are other issues with this lab that have to do with the need for efficient MATLAB code. These will be discussed in detail during the preparative session.

Particle Filter:

1. What are the particles of the particle filter?
2. What are importance weights, target distribution, and proposal distribution and what is the relation between them?
3. What is the cause of particle deprivation and what is the danger?
4. Why do we resample instead of simply maintaining a weight for each particle?
5. Give some examples of situations in which the average of the particle set is not a good representation of the particle set.
6. How can we make inferences about states that lie between particles?
7. How can sample variance cause problems and what are two remedies?
8. For robot localization and a given quality of posterior approximation, how are the pose uncertainty (spread of the true posterior) and number of particles we chose to use related?

PART II - MATLAB Exercises

1 Warm-up Problem: Particle Filter

In this section we will go through a rather simple solution to a 2D estimation problem using the Sampling-Importance Re-sampling (SIR) algorithm (Algorithm 1).

Algorithm 1 SIR General Algorithm

Inputs: Particle set $S_{t-1} = \{< x_{t-1}^i, w_{t-1}^i > | i = 1, ..., N\}$, Observation z_t, Control signal u_t
$\bar{S}_t = S_t = \emptyset$
for m = 1 to M **do**
  sample $x_t^m \sim p(x_t|u_t, x_{t-1}^m)$ {`Prediction`}
end for
for m = 1 to M **do**
  $w_t^m = p(z_t|x_t^m)$ {`Weighting`}
end for
$\bar{S}_t = \bigcup_{m=1}^{M} < x_t^m, w_t^m >$
for m=1 to M **do**
  draw $i \propto w_t^i$ {`Re-sampling`}
  $S_t = S_t \cup < x_t^i, \frac{1}{M} >$
end for
return S_t

Now, consider the problem of 2D target tracking: we want to repeatedly estimate where the object is located at each time-step given some (inaccurate) information about how the target has moved from the previous time-step, some measurements and the initial position of the target. The system is described by (1):

$$\begin{aligned} x_t &= \begin{bmatrix} x_{t,x} \\ x_{t,y} \end{bmatrix} \\ x_t &= x_{t-1} + u_t + \varepsilon_t \\ z_t &= x_t + \delta_t \end{aligned} \tag{1}$$

Where ε_t and δ_t are the process and the measurement noises and they are not necessarily Gaussian nor white. The measurements are given using a vision algorithm and therefore, $z_{t,x}$ and $z_{t,y}$ are the pixel coordinates of the (mass center of the) target in some input images.
In the unzipped lab file go to the folder `Warm-Up`. In MATLAB, load the `visiondata.mat` file. You can find the data for three types of experiments:

1. **Stationary target**: variables starting with **fixed**.
2. **Moving target**: Target moving on the diagonal axis of the image frame (variables starting with **mov**).

3. **Rotating target**: Target rotating around a point (variables starting with **circ**).

For each type of experiment there are three sets of measurements differing in the measurement noise:

1. **Small white gaussian**: Measurement noise is white gaussian with a small standard deviation (variables ending with 1)
2. **Big white gaussian**: Measurement noise is white gaussian with a big standard deviation (variables ending with 2)
3. **Outliers**: Measurement noise is white gaussian with a small standard deviation but with approximately 50% outliers (variables ending with 3)

Measurement noises are similar for different experiment types with the same ending number. You can find the ground truth information about each experiment type in the variables ending with **true**. Take a look at the data using the function `visualize_vision_data.m`. For example, try:

```
visualize_vision_data(fixed_meas_1,fixed_true);
```

We do not know about the control sequences (u_t), but we can choose a model and compensate with model imperfection with compensation noises. Compensation noise is an artificial noise that we have to consider in the model due to model imperfection. We can consider for example a fixed target, a target moving on a line, target moving with a fixed angular velocity or any other motion model that you can think of. We will go through some examples of the motion model and will analyze their drawbacks and advantages.

The choice of the motion model affects the modeled system and it might even change the dimension of the model's state-space. In the following sections, $\bar{x}_t$ represents the modeled state and $\bar{u}_t$ represents the motion model and the following holds for all models

$$\begin{aligned} \bar{x}_t &= \bar{x}_{t-1} + \bar{u}_t + R \\ \bar{z}_t &= \bar{C}\bar{x}_t + Q \end{aligned} \tag{2}$$

where R is the modeled process noise and Q is the modeled observation noise. Although there is no constraint on the process noise being Gaussian in particle filter based approaches, one usually models Gaussian noises mainly because working with Gaussians is easy. Therefore, in the following, we assume that the noises are white Gaussians: $Q \sim \mathcal{N}(0, \Sigma_{Q(2\times 2)}), R \sim \mathcal{N}(0, \Sigma_{R(2\times 2)})$.

1.1 State-Space Models

In this section, we introduce the state space models for the three experiments. Depending on the dynamics we are interested in modeling, the number of states can change. For example, for a stationary target or the target moving along a line, we don't necessarily need the heading of the target and can therefore model it using two states.

2D State Space

$$\begin{aligned} \bar{x}_t &= \begin{bmatrix} \bar{x}_{t,x} \\ \bar{x}_{t,y} \end{bmatrix} \\ \bar{C} &= I \end{aligned} \tag{3}$$

1. **Fixed target**

$$\bar{u}_t = 0 \tag{4}$$

2. **Target moving on a line**

$$\bar{u}_t = \begin{bmatrix} dx_0 \\ dy_0 \end{bmatrix} = v_0 \begin{bmatrix} \cos\theta_0 \\ \sin\theta_0 \end{bmatrix} dt \tag{5}$$

The control signal for the target moving on the line only depends on the initial heading θ_0 and the initial velocity v_0.

3D State Space

$$\begin{aligned} \bar{x}_t &= \begin{bmatrix} \bar{x}_{t,x} \\ \bar{x}_{t,y} \\ \bar{x}_{t,\theta} \end{bmatrix} \\ \bar{C} &= (I \mid 0)_{(2\times 3)} \end{aligned} \tag{6}$$

1. **Target moving on a line**

$$\bar{u}_t = \begin{bmatrix} dx_t \\ dy_t \\ 0 \end{bmatrix} = v_0 \begin{bmatrix} \cos x_{t-1,\theta} \\ \sin x_{t-1,\theta} \\ 0 \end{bmatrix} dt \tag{7}$$

- **Question 1:** What are the advantages/drawbacks of using (5) compared to (7)? Motivate.

2. **Target moving on a circle**

$$\bar{u}_t = \begin{bmatrix} dx_t \\ dy_t \\ d\theta_0 \end{bmatrix} = \begin{bmatrix} v_0 \cos x_{t-1,\theta} \\ v_0 \sin x_{t-1,\theta} \\ \omega_0 \end{bmatrix} dt \tag{8}$$

- **Question 2:** What types of circular motions can we model using (8)? What are the limitations (what do we need to know/fix in advance)?

1.2 Prediction

In general, assuming that the process noise is independent of the state, we can break down the prediction step into two steps (9):

$$\underbrace{\bar{x}_t^m}_{\texttt{predicted state}} \equiv \underbrace{\bar{x}_{t-1}^m + \bar{u}_t}_{\texttt{applying motion}} + \underbrace{\mathcal{N}(0, \Sigma_R)}_{\texttt{diffusion}} \tag{9}$$

Where Σ_R is the modeled process noise covariance matrix (assuming the noise is white Gaussian).

1.3 Sensor Model

Having modeled a Gaussian noise represented by Σ_Q, the likelihood function for an observation is given by (10).

$$p(z|x, \Sigma_Q, \bar{C}) = \frac{1}{2\pi|\Sigma_Q|^{\frac{1}{2}}} \exp\left(-\frac{1}{2}[z - \bar{C}x]^T \Sigma_Q^{-1}[z - \bar{C}x]\right) \quad (10)$$

Obviously, how you model the measurement noise affects how the likelihood function will look like. A simple interpretation is that if you model an accurate sensor (smaller $|\Sigma_Q|$), the likelihood function will be sharper (and vice versa).

- **Question 3:** What is the purpose of keeping the constant part in the denominator of (10)?

1.4 Re-Sampling

1.4.1 Vanilla (Multinomial) Re-Sampling

The vanilla re-sampling method is the most simple method to carry out the re-sampling step. The method draws particles independently using N random variables and the Cumulative Distribution Function (CDF) of the weights of the particle set. Algorithm 2 shows the Multinomial re-sampling method.

Algorithm 2 Multinomial Re-Sampling

$S_t = \emptyset$
for m = 1 to M **do**
  $CDF(m) = \sum_{i=1}^{m} w_t^i$
end for
for m=1 to M **do**
  $r_m = rand\{0 \leq r_m \leq 1\}$
  $i = arg\min_j CDF(j) \geq r_m$
  $S_t = S_t \cup < x_t^i, \frac{1}{M} >$
end for
return S_t

1.4.2 Systematic Re-Sampling

Systematic re-sampling (Stochastic Universal re-sampling) is an alternative method for re-sampling which offers better speed in addition to better variance. Algorithm 3 shows the Systematic re-sampling method.

- **Question 4:** How many random numbers do you need to generate for the Multinomial re-sampling method? How many do you need for the Systematic re-sampling method?

- **Question 5:** With what probability does a particle with weight $w = \frac{1}{M} + \epsilon$ survive the re-sampling step in each type of re-sampling (vanilla

Algorithm 3 Systematic Re-Sampling

$S_t = \emptyset$
for m = 1 to M **do**
  $CDF(m) = \sum_{i=1}^{m} w_t^i$
end for
$r_0 = rand\{0 \leq r_0 \leq \frac{1}{M}\}$
for m=1 to M **do**
  $i = \min j : CDF(j) \geq r_0 + \frac{m-1}{M}$
  $S_t = S_t \cup < x_t^i, \frac{1}{M} >$
end for
return S_t

and systematic)? What is this probability for a particle with $0 \leq w < \frac{1}{M}$? What does this tell you? **Hint**: it is easier to reason about the probability of not surviving, that is M failed binary selections for vanilla, and then subtract that amount from 1.0 to find the probability of surviving.

1.5 Experiments

Load the file: `visiondata.mat` if it's not already in your current workspace. The entrance function to the code is `pf_track` which takes a measurement set, a ground truth set and a verbose level as input arguments. You can run the code using for example

```
pf_track(fixed_meas_1,fixed_true,2);
```

- **Question 6:** Which variables model the measurement noise/process noise models?

Read through the code and make sure you understand what is going on in the code. Model the process and measurement noises for the `fixed_meas_1 measurement` set. Run the particle filter algorithm and observe how the algorithm is working. At the end of the simulation, the mass center of the particle set at each time step is considered to be the estimate of the filter and the estimates are compared to the ground truth.

Try to adjust the number of particles for a 2D state space to have a precise estimate for the `fixed_meas_1 measurement` set (precise: mean absolute error should be less than 1 pixels). Compare now to the 3D state space without changing the parameters.

- **Question 7:** What happens when you do not perform the diffusion step? (You can set the process noise to 0)

- **Question 8:** What happens when you do not re-sample? (set RESAMPLE MODE=0)

- **Question 9:** What happens when you increase/decrease the standard deviations(diagonal elements of the covariance matrix) of the observation noise model? (try values between 0.0001 and 10000)
- **Question 10:** What happens when you increase/decrease the standard deviations (diagonal elements of the covariance matrix) of the process noise model? (try values between 0.0001 and 10000)

Now, try the moving targets. For the next three questions try to think about the answer and write what you think will happen.

- **Question 11:** How does the choice of the motion model affect a reasonable choice of process noise model?
- **Question 12:** How does the choice of the motion model affect the precision/accuracy of the results? How does it change the number of particles you need?
- **Question 13:** What do you think you can do to detect the outliers in the third type of measurements? **Hint**: What happens to the likelihoods of the observation when it is far away from what the filter has predicted?

Play with the parameters and observe how they affect the filter and how they change the behavior of the filter in different scenarios. Does this confirm what you wrote for the last three questions?

- **Question 14:** Using 1000 particles, what is the best precision you get for the second type of measurements of the object moving on the circle when modeling a fixed, a linear or a circular motion (using the best parameter setting)? How sensitive is the filter to the correct choice of the parameters for each type of motion?

Congratulations! You have finished the warm-up problem (and most of the questions)!

2 Main problem: Monte Carlo Localization

In this section, we will go through the MCL problem. You will encounter both the tracking problem and the global localization problem. We will be using the SIR algorithm (Algorithm 1) similar to the previous section, with slightly different changes for the prediction and update steps. We will utilize a 3D state-space (x,y,θ) in the following.

2.1 Prediction

For the prediction step, we will utilize an approach based on motion and diffusion similar to (9). The motion model is similar to the motion model in the EKF Localization problem. We will use odometry as the motion model (consult (4) in the lab 1 instructions).

2.2 Observation Model

The sensor model (the observation model) is also similar to the observation model in the EKF Localization problem: Assuming a range-bearing measurement setting and a Gaussian measurement noise, the measurement model for the sensor is given by (11):

$$\begin{array}{lcl} h(x_t, W, j) & = & \begin{bmatrix} \sqrt{(W_{j,x} - x_{t,x})^2 + (W_{j,y} - x_{t,y})^2} \\ atan2(W_{j,y} - x_{t,y}, W_{j,x} - x_{t,x}) - x_{t,\theta} \end{bmatrix} \\ z_t^i & = & h(x_t, W, j(i)) + \eta \\ \eta \sim N(0, Q) & & \end{array} \tag{11}$$

where W is the given map of the environment.

2.3 Data Association

We will utilize the maximum likelihood data association (Algorithm 4).

Algorithm 4 Maximum Likelihood Data Association Algorithm for the t^{th} time step

for all observations i in z_t **do**
  for m=1 to M **do**
    for all landmarks k in W **do**
      $\hat{z}_t^{k,m} = h(x_t^m, W, k)$ {`Predict Measurement`}
      $\nu_t^{i,k,m} = z_t^i - \hat{z}_t^{k,m}$ {`Calculate Innovation`}
      $\psi_t^{i,k,m} = \frac{1}{2\pi|Q|^{\frac{1}{2}}} exp[-\frac{1}{2}(\nu_t^{i,k,m})^T (Q)^{-1} \nu_t^{i,k,m}]$ {`Calculate Likelihood`}
    end for
    $\hat{c}_t^{i,m} = \arg\max_k \psi_t^{i,k,m}$
    $\Psi_t^{i,m} = \psi_t^{i,\hat{c}_t^{i,m},m}$
  end for
end for

2.4 Weighting

Having computed the associations, the weighting is straightforward:

$$p(z_t | x_t^m, \Psi_t^{i_{1:n},m}) = \prod_{j=1}^{n} \Psi_t^{j,m} \tag{12}$$

where n is the number of observations in t^{th} time step. The particle weights are proportional to $p(z_t | x_t^m, \Psi_t^{i_{1:n},m})$.

2.5 Outlier Detection

Having computed the associations, we can define simple outlier detection methods using measures like the average likelihood of particles. A simple outlier

detection measure is given by:

$$\hat{o}_t^i \quad = \quad \frac{1}{M} \sum_{m=1}^{M} \Psi_t^{j,m} \leq \lambda_\Psi \tag{13}$$

where λ_Ψ is a threshold on the average likelihood of the particles taking a measurement.

- **Question 15:** What parameters affect the mentioned outlier detection approach? What will be the result of the mentioned method if you model a very weak measurement noise $|Q| \to 0$?
- **Question 16:** What happens to the weight of the particles if you do not detect outliers?

2.6 Re-Sampling

The re-sampling methods in this lab are the same as in the warm-up part: Multinomial Re-sampling and Systematic Re-sampling.

2.7 Instructions

Go to the `MCL` folder within the lab directory. The structure of this lab assignment (Code + GUI) is very similar to what you worked with in the previous lab. However, unlike in the previous lab, you should be able to perform both, global localization and tracking using your implementation. For a tracking problem, the initial position is given, while for a global localization problem, the filter has no information about the robot's initial location. You will be able to switch between the two modes in the GUI.

Implementation Details

Read the code and get some idea about what is going on in it. In order to be able to start the simulation, you need to complete the following functions:

1. **calculate_odometry.m**: Use (4) in the lab 1 instructions
2. **predict.m**: Use (9)
3. **observation_model.m**: Similar to lab 1
4. **associate.m**: Use Algorithm 4
5. **weight.m**: Use (12)
6. **multinomial_resample.m**: Use Algorithm 2
7. **systematic_resample.m**: Use Algorithm 3

The number of lines of code you write would be approximately 70-80 lines while you can reuse code from what is provided and what you have done in the EKF Localization.

2.8 Data Sets

The three data sets provided for the previous lab are also included here in addition to new data sets. Start with the data sets you are already familiar with (Dataset 1-3). Start with a tracking problem. Try different number of particles. Play with the process and measurement noise covariance matrices and pay attention how the performance of the filter is affected by them. If you have done everything right and your code is functioning well, you will realize that you need compensation noises in order to make the filter work correctly (you need to model stronger noises than the actual noises in the underlying system). You can use Dataset 2 to test if your outlier detection method works correctly. Now, try the global localization problem. Note that you need to use more particles in a global localization problem compared to a tracking problem, especially in cases that multiple hypotheses match the measurements and the particle set also agrees with the hypotheses. For example, if you perform a global localization on Dataset 3, as the environment is perfectly symmetric, you have 160 valid hypotheses for the entire simulation!

If you feel comfortable using the Particle filter on data sets 1-3, move on to the new data sets:

1. **Dataset 4**: This data set corresponds to a perfectly symmetric environment with 4 landmarks. Ground truth and odometry information are displayed in Figure 1. Try tracking and global localization with this data set. How many valid hypotheses do you have for these 4 landmarks? Start with 1.000 particles. Does your filter keep track of all hypotheses reliably? How about 10.000 particles? What do you think is the reason? Try multinomial sampling. How well does it preserve multiple hypotheses? How well are your hypotheses preserved when you model stronger/weaker measurement noises?

2. **Dataset 5**: This data set corresponds to a nearly symmetric environment with 5 landmarks (see Figure 2). At first, there are multiple valid hypotheses, but after a simulation time of approximately 37s, the robot observes the top right landmark which breaks the symmetry and only one hypothesis remains valid. Does your filter converge to the correct hypothesis? Include in your report an image just before and just after the convergence.

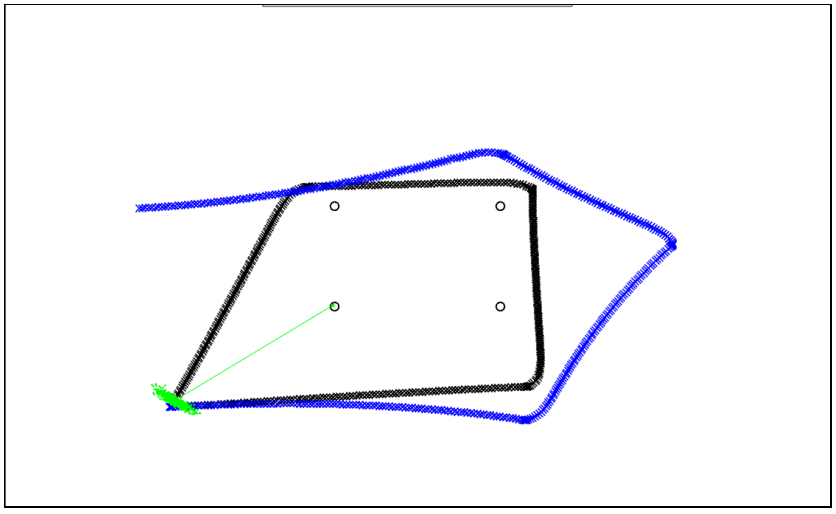

Figure 1: Dataset 4. The environment is perfectly symmetric. However, the landmarks are relatively far apart which results in a small number of available measurement. This should be taken into account when parameterizing the filter.

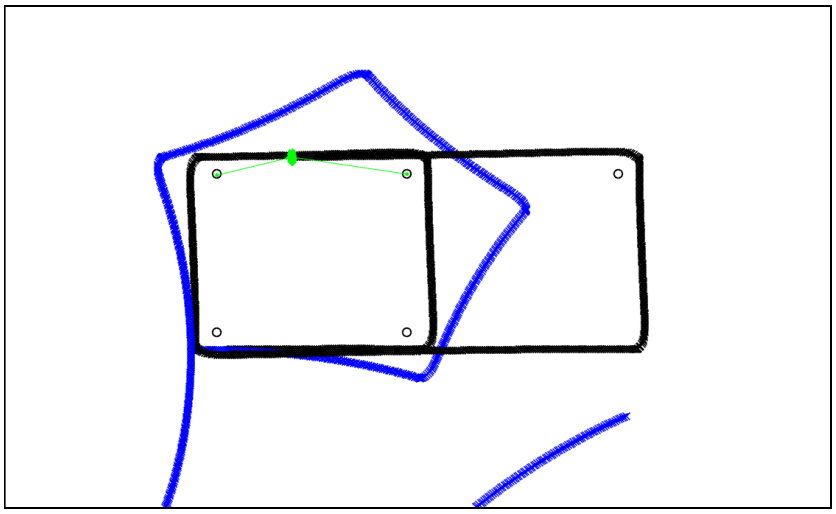

Figure 2: Dataset 5. An almost perfectly symmetric layout. Observing the top right landmark will break the symmetry. If tuned correctly, the number of hypotheses tracked by the filter will be reduced to one and the global localization problem turns into a tracking problem.