Perception in Robotics
Term 2, 2018. PS1

Gonzalo Ferrer
Skoltech

October 26, 2018

This problem set has four tasks, comprising 10% of your course grade, which is individual work. You are encouraged to talk at the conceptual level with other students, discuss on the equations and even on results, but you may not show/share/copy any non-trivial code.

Submission Instructions

Your assignment must be submitted by 11:59p on **Sunday, 11 November**. You are to upload your assignment directly to Canvas as two attachments:

1. A `.tar.gz` or `.zip` file *containing a directory* named after your uniquename with the structure shown below.

   ```
   alincoln_ps1.zip:
   alincoln_ps1/
   alincoln_ps1/task1.py
   alincoln_ps1/task2.py
   alincoln_ps1/plot2dcov.py
   alincoln_ps1/task3.py
   alincoln_ps1/task4.py
   ```

2. A PDF with the written portion of your write-up. Scanned versions of hand-written documents, converted to PDFs, are perfectly acceptable (reduced size). No other formats (e.g., .doc) are acceptable. Your PDF file should adhere to the following naming convention: `alincoln_ps1.pdf`.

Homework received after 11:59p is considered late and will be penalized as per the course policy. The ultimate timestamp authority is the one assigned to your upload by Canvas. No exceptions to this policy will be made.

Important: For all (x, y) plots you will want to use the `Axes.set_aspect('equal')` command in the `matplotlib` library to set the aspect ratio so that equal tick mark increments on the x-,y- and z-axis are equal in size. This makes spheres look like spheres, instead of an ellipsoid.

Task 1: Probability (25 points)

For this task, write a Python script that generates all the different figures, as requested on each problem.

A. (5 pts) Imagine that you are tele-operating a robot and an obstacle appears in front of us. We can guess that the distance to the obstacle x is approximately one meter away from the robot just by taking a quick look; this *prior* estimation is not exempt error, which might be modeled as Gaussian distribution. Thus, the *probability density function*, $p(x)$, is a one dimensional Gaussian distribution, $\mathcal{N}(x; 1, 1)$. Plot the corresponding prior distribution, $p(x)$.

   Hint: you might want to look at the library `scipy.stats` and use the function `norm.pdf()`.

B. (5 pts) What is the probability that we are actually colliding with the wall, given the prior $p(x)$?

   Hint: you might want to use the function `norm.cdf()`.

C. (5 pts) Fortunately for us (and for the robot), there is a sensor (laser, sonar, etc.) that provides an observation z regarding the distance to the nearest obstacle in front of us. The sensor is not exempt from uncertainty; thus, the likelihood function, $p(z|x)$, is required, i.e., the *pdf* of z conditioned by the real position of the obstacle, x. The likelihood function is again a Gaussian distribution given by $p(z|x) = \mathcal{N}(z; x, \sigma^2)$, with variance $\sigma^2 = 0.2$. However, we are interested in the real distance x. Use the Bayes' theorem to derive the posterior distribution, $p(x|z)$, given an observation $z = 0.75$ and plot it. For a better comparison, plot the prior distribution, $p(x)$, too.

D. (5 pts) Estimate the expected value of the posterior distance to the wall $E\{x|z\}$ qualitatively by inspection on the plot. The numerical solution is not necessary.

E. (5 pts) Plot the joint probability density function $p(x, z)$.

Task 2: Multivariate Gaussian (25 points)

A. (10 pts) Write the function `plot2dcov` which plots the 2d contour given three core parameters: mean, covariance, and the iso-contour value k. You may add any other parameter such as color, number of points, etc.

   Hint: Make use of the Cholesky decomposition `scipy.linalg.cholesky` or SVD and project a circumference with radius k, as explained in class. Use, for instance, 30 points.

   Hint: the Cholesky decomposition routine, could solve for the upper triangular matrix, $A^\top \cdot A = \Sigma$ which is not what you may want. Read the function help and make sure the decomposition maps back to the original covariance in the form $A \cdot AA^\top = \Sigma$.

   Then, use `plot2dcov` to draw the iso-contours corresponding to 1,2,3-sigma of the following Gaussian distributions: $\mathcal{N}\left(\begin{bmatrix}0\\0\end{bmatrix}, \begin{bmatrix}1 & 0\\0 & 2\end{bmatrix}\right)$, $\mathcal{N}\left(\begin{bmatrix}5\\0\end{bmatrix}, \begin{bmatrix}3 & -0.4\\-0.4 & 2\end{bmatrix}\right)$ and $\mathcal{N}\left(\begin{bmatrix}2\\2\end{bmatrix}, \begin{bmatrix}9.1 & 6\\6 & 4\end{bmatrix}\right)$. Use the `set_aspect('equal')` command and comment on them.

B. (5 pts) Write the equation, in vector form, for computing the sample mean and covariance matrix of a set of points $\{x_i\}$.

C. (10 pts) Draw random samples from a multivariate normal distribution. You can use the python function that draws samples from the univariate normal distribution $\mathcal{N}(0, 1)$. In particular, draw and plot 200 samples from $\mathcal{N}\left(\begin{bmatrix}2\\2\end{bmatrix}, \begin{bmatrix}1 & 1.3\\1.3 & 3\end{bmatrix}\right)$; also plot their corresponding 1-sigma iso-contour. Then, using the results from B, calculate the sample mean and covariance and plot again the 1-sigma iso-contour for the estimated Gaussian parameters. Run the experiment multiple times and try different number of samples. Comment on the results.

Task 3: Covariance Projection (25 points)

For this task, we will model an omni-directional robotic platform, i.e., a holonomic platform moving as a free point without restrictions. Somehow, the platform is malfunctioning; thus, it is moving strangely and uncontrollably. We will consider a time step of $\Delta t = 0.5$. The corresponding propagation model is as follows:

$$\begin{bmatrix} x \\ y \end{bmatrix}_t = \begin{bmatrix} 1+\Delta t & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}_{t-1} + \begin{bmatrix} 2\Delta t \\ 0 \end{bmatrix}_t .$$

A. (5 pts) Draw the propagation state *pdf* (1-sigma iso-contour and 500 particles) for times indexes $t = 0, \dots, 5$ assuming that the *pdf* for the initial state is $P\left(\begin{bmatrix} x \\ y \end{bmatrix}_0\right) = \mathcal{N}\left(\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0.1 & 0 \\ 0 & 0.1 \end{bmatrix}\right)$.

B. (5 pts) We manage to fix the platform, and now it moves properly. The propagation model is the following: $\begin{bmatrix} x \\ y \end{bmatrix}_t = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}_{t-1} + \begin{bmatrix} \Delta t & 0 \\ 0 & \Delta t \end{bmatrix} \begin{bmatrix} v_x \\ v_y \end{bmatrix}_t + \begin{bmatrix} \eta_x \\ \eta_y \end{bmatrix}_t$, where the controls $u = [v_x, v_y]^\top$ are the velocities which are commanded to the robot. Unfortunately, there exists some uncertainty on command execution $\begin{bmatrix} \eta_x \\ \eta_y \end{bmatrix}_t \sim \mathcal{N}\left(\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0.1 & 0 \\ 0 & 0.1 \end{bmatrix}\right)$.

Write the equations corresponding to the mean and covariance after a single propagation. How can we use this result iteratively?

C. (5 pts) Draw the propagation state *pdf* (1-sigma iso-contour) for times indexes $t = 0, \dots, 5$ and the control sequence $u_1 = [3, 0]^\top$, $u_2 = [0, 3]^\top$, $u_3 = [3, 0]^\top$, $u_4 = [0, -3]^\top$ and $u_5 = [3, 0]^\top$. The *pdf* for the initial state is $\begin{bmatrix} x \\ y \end{bmatrix}_0 \sim \mathcal{N}\left(\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0.1 & 0 \\ 0 & 0.1 \end{bmatrix}\right)$.

D. (5 pts) Now, suppose that the robotic platform is non-holonomic, and the corresponding propagation model is: $\begin{bmatrix} x \\ y \\ \theta \end{bmatrix}_t = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ \theta \end{bmatrix}_{t-1} + \begin{bmatrix} cos(\theta)\Delta t & 0 \\ sin(\theta)\Delta t & 0 \\ 0 & \Delta t \end{bmatrix} \begin{bmatrix} v \\ w \end{bmatrix}_t + \begin{bmatrix} \eta_x \\ \eta_y \\ \eta_\theta \end{bmatrix}_t$, being $\begin{bmatrix} \eta_x \\ \eta_y \\ \eta_\theta \end{bmatrix}_t \sim \mathcal{N}\left(\begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0.2 & 0 & 0 \\ 0 & 0.2 & 0 \\ 0 & 0 & 0.1 \end{bmatrix}\right)$ and the *pdf* for the initial state $\begin{bmatrix} x \\ y \\ \theta \end{bmatrix}_0 \sim \mathcal{N}\left(\begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0.1 & 0 & 0 \\ 0 & 0.1 & 0 \\ 0 & 0 & 0.5 \end{bmatrix}\right)$.

Propagate for five time intervals as before, using the control $u_t = [3, 2]^\top$ showing the propagated Gaussian by plotting the 1-sigma iso-contour. *Hint:* you can marginalize out θ and plot the corresponding $\Sigma_{(xy)}$ as explained in class.

E. (5 pts) Repeat the same experiment as above, using the same control input u_t and initial state estimate, now considering that noise is expressed in the action space instead of state space: $\begin{bmatrix} x \\ y \\ \theta \end{bmatrix}_t = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ \theta \end{bmatrix}_{t-1} + \begin{bmatrix} cos(\theta)\Delta t & 0 \\ sin(\theta)\Delta t & 0 \\ 0 & \Delta t \end{bmatrix} \begin{bmatrix} v + \eta_w \\ w + \eta_w \end{bmatrix}_t$, being $\begin{bmatrix} \eta_w \\ \eta_w \end{bmatrix}_t \sim \mathcal{N}\left(\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 2 & 0 \\ 0 & 0.1 \end{bmatrix}\right)$. Comment on the results.

Task 4: Kalman Filter (25 points)

Imagine that we want to estimate the translational velocity, x_t, of a car by using the gas pedal, which provides information regarding the acceleration, u_t, plus some noise, ϵ_t. The propagation model is $x_t = x_{t-1} + \Delta t \cdot (u_t + \epsilon_t)$, where $\epsilon_t \sim \mathcal{N}(0, M)$, and covariance $M = 4$.

All relevant data for this task is contained in the `t4` file. You may want to use numpy `load` function. Some parameters are included there too, such as $\Delta t = 0.1$, $x_0 = 0$, $\Sigma_0 = 0$ (vehicle stopped).

A. (5 pts) Unfortunately for us, there is a difference between the controls that we have commanded and the controls that the car has actually executed. This uncertainty is expressed in the form of corrupted controls by the r.v. ϵ_t. Calculate the state trajectory of x_t just by considering the noisy controls $u_t + \epsilon_t$ provided. Write the equations and plot the results, that is, $\bar{\mu}_t$ and $\pm$1-sigma. Compare it to the real state x_t of the velocity. Comment on the results.

   Hint: this a covariance projection problem. The noise should be projected too.

B. (3 pts) In order to circumvent the drifting issue (this might be a hint for A), we need to observe our state x_t. Our observations are the integration over a time step of an accelerometer *perfectly* aligned with the car's translation direction. The observation model $z_t = x_t + \delta_t$, where $\delta_t \sim \mathcal{N}(0, Q)$, being $Q = 10$. Plot the raw observations and compare them with the real state x_t.

C. (5 pts) To integrate the observations z_t with our propagation model, we need to identify all the components of the KF. Write the equations for elements A, B, C and K, and expand until you get a useful result for implementing the KF.

D. (7 pts) Implement the Kalman filter for a recursive state estimation of x_t corresponding to the car's velocity. Plot the output of the algorithm $bel(x_t)$ as well as its $\pm$1,3-sigma. *Hint:* you may use the included `ciplot.py` function to draw the intervals.

E. (5 pts) How does the effect of integrating the observations z_t affect the result compared to part A? Comment on any other interesting observations.