Exercise 4 solutions

May 25, 2023

Exercise 1: Sampling

Implement three functions in Python which generate samples of a normal distribution $\mathcal{N}(\mu, \sigma^2)$. The input parameters of these functions should be the mean μ and the variance σ^2 of the normal distribution. As only source of randomness, use samples of a uniform distribution.

- *In the first function, generate the normal distributed samples by summing up 12 uniform distributed samples, as explained in the lecture.*
- *In the second function, use rejection sampling.*
- *In the third function, use the Box-Muller transformation method. The Box-Muller method allows to generate samples from a standard normal distribution using two uniformly distributed samples u_1, $u_2 \in [0, 1]$ via the following equation:*

$$x = \cos(2\pi u_1)\sqrt{-2\log u_2}.$$

Compare the execution times of the three functions using Python's built-in function `timeit`.
Also, compare the execution times of your own functions to the built-in function `numpy.random.normal`.

Please find the code listing below.

```
#!/usr/bin/env python

import math
import numpy as np
import scipy.stats
import timeit
import matplotlib.pyplot as plt

"""
  Exercise 1: Normal distribution sampling
  mu -- mean of the normal distribution
  sigma -- std_dev of the normal distribution
"""
```

```
def sample_normal_twelve(mu, sigma):
  """ Sample from a normal distribution using 12 uniform samples.
  See lecture on probabilistic motion models slide 19 for details.
  """

  # Formula returns sample from normal distribution with mean = 0
  x = 0.5 * np.sum(np.random.uniform(-sigma, sigma, 12))
  return mu + x

def sample_normal_rejection(mu, sigma):
  """Sample from a normal distribution using rejection sampling.
  See lecture on probabilistic motion models slide 25 for details.
  """

  # Length of interval from wich samples are drawn
  interval = 5*sigma

  # Maximum value of the pdf of the desired normal distribution
  max_density = scipy.stats.norm(mu,sigma).pdf(mu)

  # Rejection loop
  while True:
    x = np.random.uniform(mu - interval, mu + interval, 1)[0]
    y = np.random.uniform(0, max_density, 1)
    if y <= scipy.stats.norm(mu, sigma).pdf(x):
      break
  return x

def sample_normal_boxmuller(mu, sigma):
  """Sample from a normal distribution using Box-Muller method.
  See exercise sheet on sampling and motion models.
  """

  # Two uniform random variables
  u = np.random.uniform(0, 1, 2)

  # Box-Muller formula returns sample from STANDARD normal distribution
  x = math.cos(2*np.pi*u[0]) * math.sqrt(-2*math.log(u[1]))
  return mu + sigma * x

def evaluate_sampling_time(mu, sigma, n_samples, sample_function):
  tic = timeit.default_timer()
  for i in range(n_samples):
    sample_function(mu, sigma)
  toc = timeit.default_timer()
  time_per_sample = (toc - tic) / n_samples * 1e6
  print("%30s : %.3f us" % (sample_function.__name__, time_per_sample))
```

```
def evaluate_sampling_dist(mu, sigma, n_samples, sample_function):
  n_bins = 100
  samples = []
  for i in range(n_samples):
    samples.append(sample_function(mu, sigma))
  print("%30s : mean = %.3f, std_dev = %.3f" % (
    sample_function.__name__, np.mean(samples), np.std(samples)))
  plt.figure()
  count, bins, ignored = plt.hist(samples, n_bins, normed=True)
  plt.plot(bins, scipy.stats.norm(mu, sigma).pdf(bins), linewidth=2, color='r')
  plt.xlim([mu - 5*sigma, mu + 5*sigma])
  plt.title(sample_function.__name__)

def main():
  mu, sigma = 0, 1
  sample_functions = [
    sample_normal_twelve,
    sample_normal_rejection,
    sample_normal_boxmuller,
    np.random.normal
  ]

  for fnc in sample_functions:
    evaluate_sampling_time(mu, sigma, 1000, fnc)

  n_samples = 10000
  print("evaluting sample distances with:"  )
  print("  mean    :", mu)
  print("  std_dev :", sigma)
  print("  samples :", n_samples)

  for fnc in sample_functions:
    evaluate_sampling_dist(mu, sigma, n_samples, fnc)

  plt.show()

if __name__ == "__main__":
  main()
```

Exercise 2: Odometry-based Motion Model

A working motion model is a requirement for all Bayes Filter implementations. In the following, you will implement the simple odometry-based motion model.

(a) *Implement the odometry-based motion model in Python. Your function should take*

the following three arguments

$$\vec{x}_t = \begin{pmatrix} x \\ y \\ \theta \end{pmatrix} \quad \vec{u}_t = \begin{pmatrix} \delta_{r1} \\ \delta_{r2} \\ \delta_t \end{pmatrix} \quad \boldsymbol{\alpha} = \begin{pmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \\ \alpha_4 \end{pmatrix},$$

where $\vec{x}_t$ is the pose of the robot before moving, $\vec{u}_t$ is the odometry reading obtained from the robot, and $\boldsymbol{\alpha}$ are the noise parameters of the motion model. The return value of the function should be the new pose $\vec{x}_{t+1}$ of the robot predicted by the model.

As we do not expect the odometry measurements to be perfect, you will have to take the measurement error into account when implementing your function. Use the sampling methods you implemented in Exercise 1 to draw normally distributed random numbers for the noise in the motion model.

Please find the code listing below. The code assumes that the solution to exercise 1 is stored as `sample_normal_distribution.py`.

(b) *If you evaluate your motion model over and over again with the same starting position, odometry reading, and noise values what is the result you would expect?*

Each evaluation will give a different result. The three steps of the motion (rotation 1, translation, rotation 2) will be randomized according to the odometry model:

$$\hat{\delta}_{r1,2} = \delta_{r1,2} + \text{normal}(\alpha_1 |\delta_{r1,2}| + \alpha_2 \delta_t) \tag{1}$$

$$\hat{\delta}_t = \delta_t + \text{normal}(\alpha_3 \delta_t + \alpha_4 (|\delta_{r1}| + |\delta_{r2}|)) \tag{2}$$

The pure rotational uncertainty is given by α_1, while the pure translational uncertainty is given by α_3. The translation affects both rotations through α_2 and both rotations affect the translation through α_4. The most probable result is the one without noise (the robot moves exactly as given by $\vec{u}_t$).

(c) *Evaluate your motion model 5000 times for the following values*

$$\vec{x}_t = \begin{pmatrix} 2.0 \\ 4.0 \\ 0.0 \end{pmatrix} \quad \vec{u}_t = \begin{pmatrix} \frac{\pi}{2} \\ 0.0 \\ 1.0 \end{pmatrix} \quad \boldsymbol{\alpha} = \begin{pmatrix} 0.1 \\ 0.1 \\ 0.01 \\ 0.01 \end{pmatrix}.$$

Plot the resulting (x, y) positions for each of the 5000 evaluations in a single plot.

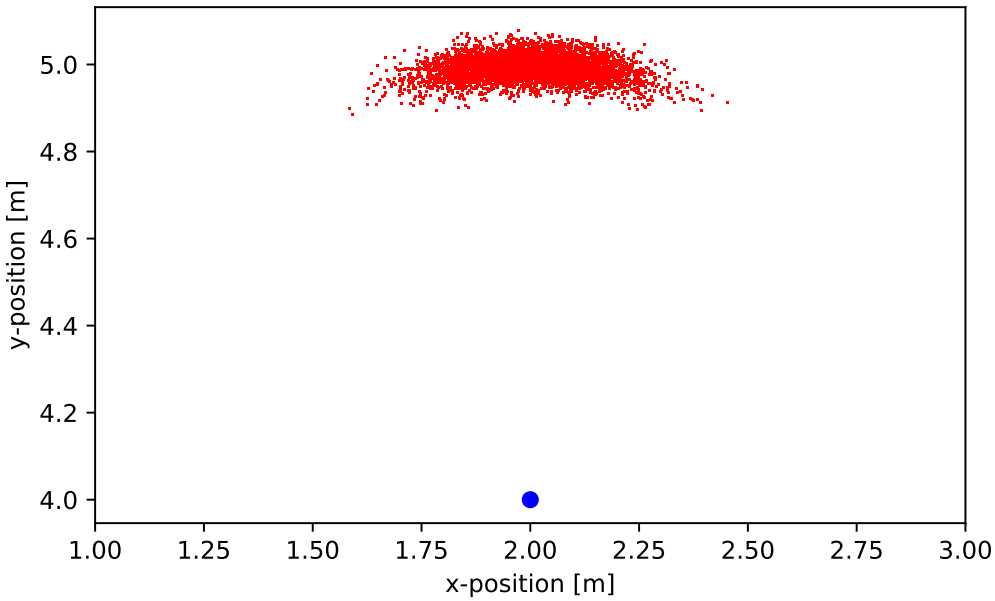
Please find the code listing and figure below.

```
#!/usr/bin/env python

import numpy as np
import math
import matplotlib.pyplot as plt
```

```
""" Exercise 2 a) Implement odometry-based motion model """

def sample_normal(mu, sigma):
  # import sample_normal_distribution as snd
  # return snd.sample_normal_twelve(mu,sigma)
  #return snd.sample_normal_rejection(mu,sigma)
  #return snd.sample_normal_boxmuller(mu,sigma)
  return np.random.normal(mu,sigma)

def sample_odometry_motion_model(x, u, a):
  """ Sample odometry motion model.

  Arguments:
  x -- pose of the robot before moving [x, y, theta]
  u -- odometry reading obtained from the robot [rot1, rot2, trans]
  a -- noise parameters of the motion model [a1, a2, a3, a4]

  See lecture on probabilistic motion models slide 27 for details.
  """
  #
  # delta_hat_r1 = u[0] + sample_normal(0, a[0]*abs(u[0]) + a[1]*abs(u[2]))
  # delta_hat_r2 = u[1] + sample_normal(0, a[0]*abs(u[1]) + a[1]*abs(u[2]))
  # delta_hat_t  = u[2] + sample_normal(0, a[2]*abs(u[2]) + a[3]*(abs(u[0])+abs(u[1])))
  delta_hat_r1 = u[0] + sample_normal(0, a[0]*abs(u[0]) + a[1]*u[2])
  delta_hat_r2 = u[1] + sample_normal(0, a[0]*abs(u[1]) + a[1]*u[2])
  delta_hat_t  = u[2] + sample_normal(0, a[2]*u[2] + a[3]*(abs(u[0])+abs(u[1])))

  x_prime = x[0] + delta_hat_t * math.cos(x[2] + delta_hat_r1)
  y_prime = x[1] + delta_hat_t * math.sin(x[2] + delta_hat_r1)
  theta_prime = x[2] + delta_hat_r1 + delta_hat_r2

  return np.array([x_prime, y_prime, theta_prime])


def sample_odometry_motion_model_new(x, u, a):
  x = np.asarray(x)
  delta_hat_r1 = u[0] + np.random.normal(0, a[0]*abs(u[0]) + a[1]*u[2], size=(x.shape[0], ))
  delta_hat_r2 = u[1] + np.random.normal(0, a[0]*abs(u[1]) + a[1]*u[2], size=(x.shape[0], ))
  delta_hat_t  = u[2] + np.random.normal(0, a[2]*u[2] + a[3]*(abs(u[0])+abs(u[1])), size=(x.shape[0

  x_update = np.stack(
    [
      delta_hat_t * np.cos(x[:, 2] + delta_hat_r1),
      delta_hat_t * np.sin(x[:, 2] + delta_hat_r1),
      x[:, 2] + delta_hat_r1 + delta_hat_r2
    ], axis=-1
  )
```

```
  return x + x_update



from matplotlib import cm
cmap = cm.get_cmap('Pastel1')

""" Exercise 2 c) Evaluate motion model """

def main():
  x = [2, 4, 0]
  u = [np.pi/2, 0, 1]
  a = [0.05, 0.05, 0.01, 0.01]

  utils = [
    [np.pi / 2, 0, 1],
    [np.pi * 3/2, 0, 1],
    [np.pi * 3/2, 0, 3]
  ]

  # num_samples = 5000
  # x_prime = np.zeros([num_samples, 3])
  # for i in range(0, num_samples):
  #   x_prime[i,:] = sample_odometry_motion_model(x,u,a)
  #
  # plt.plot(x[0], x[1], "bo")
  # plt.plot(x_prime[:,0], x_prime[:,1], "r,")
  # plt.xlim([1, 3])
  # plt.axes().set_aspect('equal')
  # plt.xlabel("x-position [m]")
  # plt.ylabel("y-position [m]")
  # plt.savefig("odometry_samples.pdf")
  # plt.show()

  num_samples = 5000
  x_prime = np.repeat(np.asarray([x]).reshape((1, 3)), repeats=num_samples, axis=0)

  # colors = [
  #   [.75, .0, 0.015],
  #   [.35, .7, .9],
  #   [0., .6, .5, ]
  # ]
  colors = [
    [.35, .7, .9],
    [.35, .7, .9],
    [.35, .7, .9],
  ]
```

```
  fig, ax = plt.subplots(dpi=150)
  plt.plot(x[0], x[1], "bo")
  ax.text(x=1, y=3.8, s='start')
  ax.text(x=1, y=5.2, s='step 1')
  ax.text(x=3.2, y=5.2, s='step 2')
  ax.text(x=6.2, y=5.2, s='step 3')
  for i, u in enumerate(utils):
    x_prime = sample_odometry_motion_model_new(x_prime, u, a)

    ax.plot(x_prime[:, 0], x_prime[:, 1], color=colors[i], marker='.', linestyle='None', markersize

    mean_pos = np.mean(x_prime, axis=0)
    ax.plot(mean_pos[0], mean_pos[1], color=(0., 0, 0), marker='x', markersize=7)


    print(mean_pos[2], np.std(x_prime, axis=0))
    # ax.quiver(mean_pos[0], mean_pos[1], np.cos(mean_pos[2]), np.sin(mean_pos[2]) )



  #   # plt.xlim([1, 3])
  # lgnd = ax.legend()
  # lgnd.legendHandles[0]._legmarker.set_markersize(10)
  # lgnd.legendHandles[1]._legmarker.set_markersize(10)
  # lgnd.legendHandles[2]._legmarker.set_markersize(10)

  ax.grid(True, linestyle='-.')
  ax.tick_params(width=3)
  ax.set_xlabel("x-position [m]")
  ax.set_ylabel("y-position [m]")
  ax.set_xticks(np.arange(0, 8))

  ax.axis('equal')
  plt.gray()
  plt.savefig("sampling_motion_model.pdf")
  plt.show()


if __name__ == "__main__":
  main()
```



Exercise 3: Velocity-Based Motion Model

Remark: This exercise is to be solved without Python.

Consider a robot which moves on a circular trajectory with noise-free constant translational and rotational velocities, v *and* w*. This situation is shown on page 30 of the* Probabilistic Motion Models *slides. The current pose of the robot is* (x, y, θ)*.*

(a) *Derive the following expression for the center of the circle,* (x_c, y_c)*:*

$$\begin{pmatrix} x_c \\ y_c \end{pmatrix} = \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} -\frac{v}{w}\sin\theta \\ \frac{v}{w}\cos\theta \end{pmatrix} \tag{3}$$

Please find the hand-written solution below.

(b) *Now consider the situation where we are given a start pose* (x, y, θ) *and an end pose* (x', y', θ')*, connected by a circular movement. Prove that the center of the circle can be expressed as*

$$\begin{pmatrix} x_c \\ y_c \end{pmatrix} = \frac{1}{2}\begin{pmatrix} x + x' \\ y + y' \end{pmatrix} + \mu \begin{pmatrix} y - y' \\ x' - x \end{pmatrix} \tag{4}$$

with some $\mu \in \mathbb{R}$

Hint*: Consider the line* l *that connects the center of the circle and the half-way*

point P between (x, y) and (x', y'). How are l and P related to the two terms in Equation (4)*?*

Please find the hand-written solution below.

(c) *Show that the value of μ is given by*

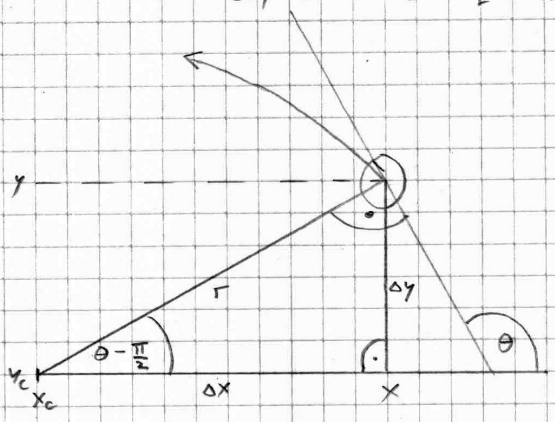
$$\mu = \frac{1}{2} \frac{(x - x') \cos\theta + (y - y') \sin\theta}{(y - y') \cos\theta - (x - x') \sin\theta}.$$

Hint*: μ can be calculated by using the fact that the line described by Equation* (4) *and the line from (x_c, y_c) to (x, y) intersect at (x_c, y_c).*

Please find the hand-written solution below.

We have now found an expression for the center of the circle that does not depend on the velocities anymore, but only on the initial and final pose.

a) Show that $\begin{bmatrix} x_c \\ y_c \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} -\frac{v}{\omega}\sin\theta \\ \frac{v}{\omega}\cos\theta \end{bmatrix}$.



We know from Highschool:

(*1): $|v| = r \cdot |\omega|$

(*2): $\cos(\alpha) = \cos(-\alpha)$

(*3): $\sin(\alpha) = -\sin(-\alpha)$

(*4): $\sin\left(\frac{\pi}{2} - \alpha\right) = \cos(\alpha)$

(*5): $\cos\left(\frac{\pi}{2} - \alpha\right) = \sin(\alpha)$

In the right-angled triangle $(\Delta x, \Delta y, r)$ it holds that

(*6) $\Delta x = r \cdot \cos\left(\theta - \frac{\pi}{2}\right)$

(*7) $\Delta y = r \cdot \sin\left(\theta - \frac{\pi}{2}\right)$

Using all the above equations, we get

$$x_c = x - \Delta x \overset{(*6)}{=} x - r \cdot \cos\left(\theta - \frac{\pi}{2}\right) \overset{(*1)}{=} x - \frac{v}{\omega}\cos\left(\theta - \frac{\pi}{2}\right)$$

$$\overset{(*2)}{=} x - \frac{v}{\omega}\cos\left(\frac{\pi}{2} - \theta\right) \overset{(*5)}{=} \underline{\underline{x - \frac{v}{\omega}\sin(\theta)}}$$

and

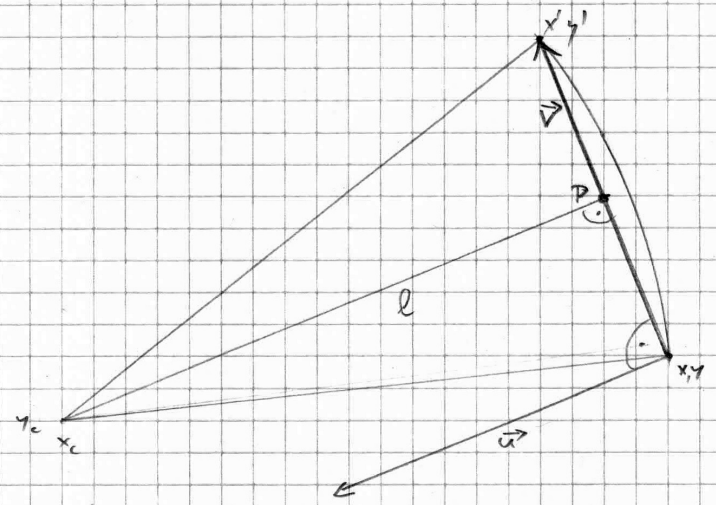
$$y_c = y - \Delta y \overset{(*7)}{=} y - r \cdot \sin\left(\theta - \frac{\pi}{2}\right) \overset{(*1)}{=} y - \frac{v}{\omega}\sin\left(\theta - \frac{\pi}{2}\right)$$

$$\overset{(*3)}{=} y + \frac{v}{\omega}\sin\left(\frac{\pi}{2} - \theta\right) \overset{(*4)}{=} \underline{\underline{y + \frac{v}{\omega}\cos(\theta)}}$$

□

b) The robot moves from (x, y, θ) to (x', y', θ)

Show that $\begin{bmatrix} x_c \\ y_c \end{bmatrix} = \frac{1}{2}\begin{bmatrix} x + x' \\ y + y' \end{bmatrix} + \mu \begin{bmatrix} y - y' \\ x' - x \end{bmatrix}$



$\begin{bmatrix} x_c \\ y_c \end{bmatrix}$ lies on the line l. P, the point halfway between $\begin{bmatrix} x \\ y \end{bmatrix}$ and $\begin{bmatrix} x' \\ y' \end{bmatrix}$ lies also on l. P has the coordinates $\begin{bmatrix} \frac{x+x'}{2} & \frac{y+y'}{2} \end{bmatrix}^T$. The vector $\vec{v}$ between $\begin{bmatrix} x \\ y \end{bmatrix}$ and $\begin{bmatrix} x' \\ y' \end{bmatrix}$ can be written as $\vec{v} = \begin{bmatrix} x' - x \\ y' - y \end{bmatrix}$. Therefore, the vector $\vec{u} = \begin{bmatrix} y - y' \\ x' - x \end{bmatrix}$ is orthogonal to $\vec{v}$.

With this, we can write every point on the line l as

$$P + \mu \cdot \vec{u} \quad \text{for an appropriate choice of } \mu \in \mathbb{R}.$$

As $\begin{bmatrix} x_c \\ y_c \end{bmatrix}$ lies on l, it therefore holds that

$$\begin{bmatrix} x_c \\ y_c \end{bmatrix} = \begin{bmatrix} \frac{x+x'}{2} \\ \frac{y+y'}{2} \end{bmatrix} + \mu \cdot \begin{bmatrix} y - y' \\ x' - x \end{bmatrix} \quad \text{for an appropriate } \mu \in \mathbb{R}$$

□

Note: $\vec{u}$ is $\vec{v}$ rotated by 90°, which can be shown using a 2D rotation matrix.

c) Show that $\mu = \frac{1}{2} \frac{(x-x')\cos\theta + (y-y')\sin\theta}{(y-y')\cos\theta - (x-x')\sin\theta}$

From a) we know that

$$\begin{bmatrix} x_c \\ y_c \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix} + \lambda \begin{bmatrix} -\sin\theta \\ \cos\theta \end{bmatrix} \quad \text{for } \lambda = \frac{v}{\omega} .$$

From b) we know that

$$\begin{bmatrix} x_c \\ y_c \end{bmatrix} = \frac{1}{2} \begin{bmatrix} x+x' \\ y+y' \end{bmatrix} + \mu \begin{bmatrix} y-y' \\ x'-x \end{bmatrix}$$

Putting both together, we get

$$\begin{bmatrix} x \\ y \end{bmatrix} + \lambda \begin{bmatrix} -\sin\theta \\ \cos\theta \end{bmatrix} = \frac{1}{2} \begin{bmatrix} x+x' \\ y+y' \end{bmatrix} + \mu \begin{bmatrix} y-y' \\ x'-x \end{bmatrix} ,$$

which can be written as the two equations

(1) $x + \lambda \sin\theta = \frac{1}{2}(x+x') + \mu(y-y')$

(2) $y + \lambda \cos\theta = \frac{1}{2}(y+y') + \mu(x'-x)$

From (1), we get $\lambda = \frac{x-x'}{2\sin\theta} - \mu \frac{y-y'}{\sin\theta}$.

Plugging this into (2) yields

$$y + (x-x')\frac{\cos\theta}{2\sin\theta} - \mu(y-y')\frac{\cos\theta}{\sin\theta} = \frac{1}{2}(y+y') + \mu(x'-x)$$

$$\Leftrightarrow \mu = \left(y - \frac{1}{2}(y+y') + \frac{1}{2}(x-x')\frac{\cos\theta}{\sin\theta} \right) \Big/ \left((y-y')\frac{\cos\theta}{\sin\theta} - (x-x') \right)$$

$$\Leftrightarrow \mu = \frac{\frac{1}{\sin\theta} \cdot \frac{1}{2}\left[(y-y')\sin\theta + (x-x')\cos\theta \right]}{\frac{1}{\sin\theta}\left[(y-y')\cos\theta - (x-x')\sin\theta \right.}$$

$$\Leftrightarrow \mu = \frac{1}{2} \frac{(x-x')\cos\theta + (y-y')\sin\theta}{(y-y')\cos\theta - (x-x')\sin\theta}$$

□