Assessment task 1: Localization of a mobile robot

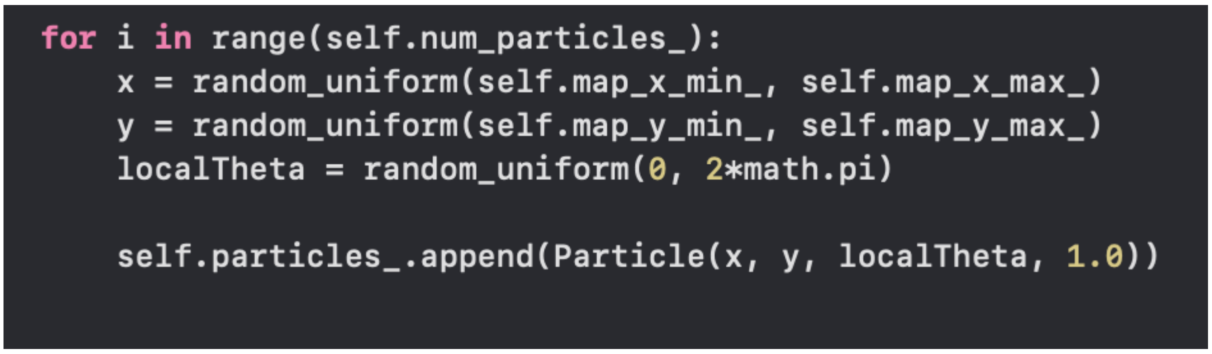
By Md Shawkath Hossain
Student Id: 12631558

Introduction

This assignment is Localizing a mobile robot within an environment. To do so, I have used Particle Filter. To achiever the goal of localization of a mobile robot, I have completed 5 assigned tasks which are Initialize the Particles, Normalize the weights, Motion updates, Observation update and Pose estimate. After implementation, I have built the Python code and run the code into a simulator to simulate the robot localization achievement. According to the simulation it was working well. However, thou Pose estimation was working but I found it is not that efficient. I have created a demo video of the simulation which could be found in the following link: https://youtu.be/2nZooC5vZZw

Initialize the Particles

As Particle Filter works with particles, as a first step I have initialized a big number of particles all over the robot environment. I have given a random location for each particle on the environment and assigned a same initial weight for each particle 1.0.



```
for i in range(self.num_particles_):
    x = random_uniform(self.map_x_min_, self.map_x_max_)
    y = random_uniform(self.map_y_min_, self.map_y_max_)
    localTheta = random_uniform(0, 2*math.pi)

    self.particles_.append(Particle(x, y, localTheta, 1.0))
```

Normalize the Weights

I have normalized the weight of all the initialized particles so that the sum of all the weight become 1.0. To do so, I have looped over all the particles and summed up them into a variable.
Then updated the initial weight of each particle by dividing it with the total weight (summed weight).

```python
def normalise_weights(self):
    # Normalise the weights of the particles in "particles_"

    ####################
    ## YOUR CODE HERE ##
    ####################

    totalWeight = 0
    for index in range(len(self.particles_)):
        totalWeight += self.particles_[index].weight

    for index in range(len(self.particles_)):
        self.particles_[index].weight = self.particles_[index].weight/totalWeight
```

Motion update

In the third task, I was given with distance and rotation of the robot when the robot moved from one position to next. With the given value, I have updated the motion of each initialized particle by updating its x, y and theta values.

```python
for index in range(len(self.particles_)):
    self.particles_[index].x = self.particles_[index].x + (distance +
        random_normal(self.motion_distance_noise_stddev_)) *
        math.cos(self.particles_[index].theta)

    self.particles_[index].y = self.particles_[index].y + (distance +
        random_normal(self.motion_distance_noise_stddev_)) *
        math.sin(self.particles_[index].theta)

    self.particles_[index].theta = wrap_angle(self.particles_[index].theta + rotation +
        random_normal(self.motion_rotation_noise_stddev_))
```

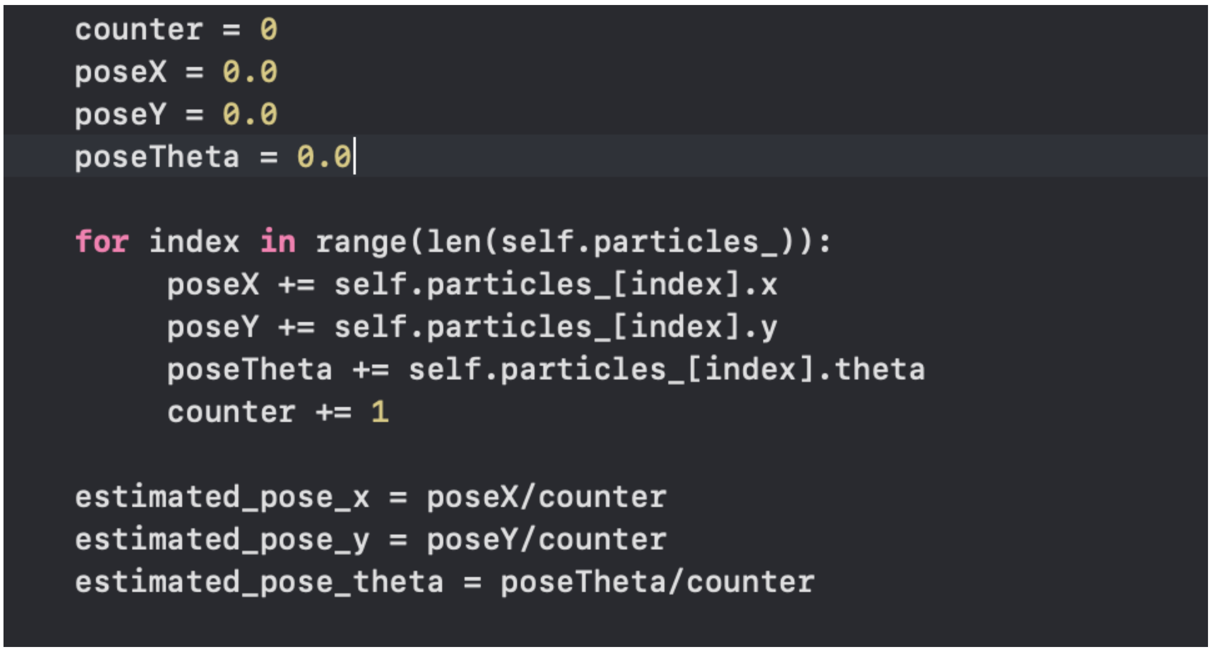
Observation update

To perform an observation update we compare the range values from the laser scanner with the expected range values of the particle.
To update the weight of the particle we multiply it with the likelihood of each particle. The likelihood of a particle is the product of the likelihood of each ray.

```python
    rangeDifference = (particle_range - scan_range) * (particle_range - scan_range)

    sensingNoiseStddev = 2*(self.sensing_noise_stddev_ * self.sensing_noise_stddev_)

    x = 2*math.pi*(self.sensing_noise_stddev_ * self.sensing_noise_stddev_)

    divisionOfRangeDiffAndSensingNoise = (rangeDifference/sensingNoiseStddev)

    divideOne = 1/math.sqrt(x)
    exponential = math.exp(-divisionOfRangeDiffAndSensingNoise)

    likelihood *= divideOne*exponential
```

Pose estimation

To estimate the pose of each particle, I have summed up x, y, and theta of all the particles. Then divided each of them with the total number of particles to get the right location. However, this method is not efficient enough and can fail in some situations.
Pose estimation with weighted sum could be far better approach. I have intension of working with weighted sum in my next classes.



```
counter = 0
poseX = 0.0
poseY = 0.0
poseTheta = 0.0

for index in range(len(self.particles_)):
     poseX += self.particles_[index].x
     poseY += self.particles_[index].y
     poseTheta += self.particles_[index].theta
     counter += 1

estimated_pose_x = poseX/counter
estimated_pose_y = poseY/counter
estimated_pose_theta = poseTheta/counter
```

Conclusion

Although it has some issues with pose estimation, it works well. However, it could be improved further for better performance and accuracy. As we know Particle Filter is not much memory efficient as for better result, we need to initialize more and more particles, in that case we could use data structures with better performance (for example: hash table). For example, looking up for particles on an array has time complexity of O(n), whereas in same scenario a hash table has constant time complexity O(1).