Particle Filter Localization in the CiberRato simulation environment

Paulo Pereira[98430]

Perceção e Controlo, Universidade de Aveiro, 3810-193 Aveiro, Portugal

Abstract. This paper covers the approach for the assignment 1 in the course "Perceção e Controlo".

Keywords: Particle Filter · Perception

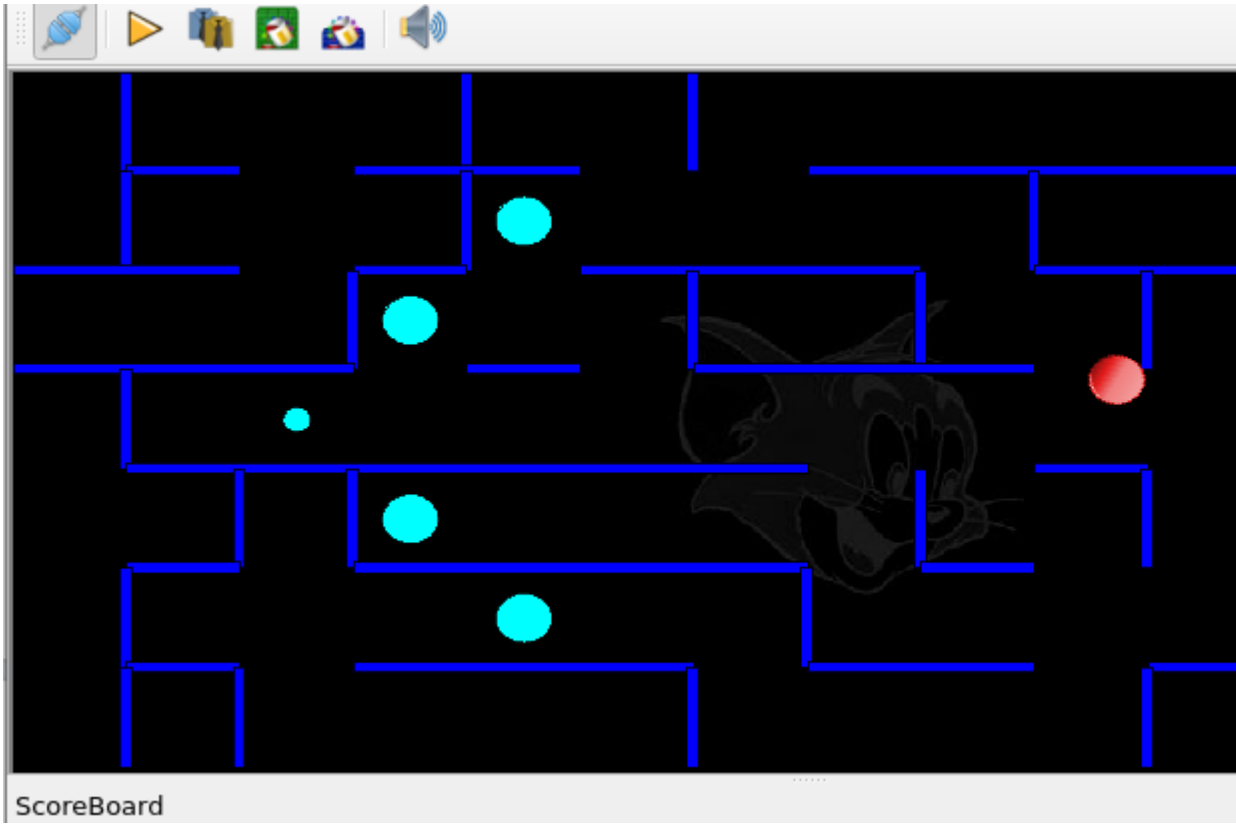
1 Introduction

Particle filters are used to estimate the current state of a robot, given it's environment and some information. For this assignment we were tasked with, using the CiberRato Tools environment, localizing a simulated robot using only the provided map and the robots infrared sensors. We were asked to implement two versions, the first one would not take into account the noise of the motor and the other one would take into consideration noise in the motors and would run until the clock run out.

2 Strategy

We took the mainRob.py robot as a baseline, revamped it in order for it to accommodate the argparse library and all the other components, such as the particle filter. Besides numpy no other external library was used.

Once we start the simulation the agent assesses it's received measures in order to send a drive command. Every time we re-sample and the particles move we save the current particle set to localization.out.



3 Developed code

The code developed uses two classes, the Map class is the same from the example and is used to parse the map provided when we first run the project. The rest of the relevant code is encapsulated by the Rob class, this class is responsible for initializing the particles, distributing them, reads the agent's sensors and using the updateParticlesWithMeasures function is responsible for taking the measures and re-sampling of the particles as well as saving the results to the wanted file.

```python
def driveAndUpdateParticles(self, in1, in2):
    self.driveMotors(in1, in2)

    self.out_left = (in1+self.out_left)/2*self.noise
    self.out_right = (in2+self.out_right)/2*self.noise
    lin = (self.out_left+self.out_right)/2

    def getNewPose(px, py, ptheta):
        x = px + lin*cos(ptheta*pi/180)
        y = py + lin*sin(ptheta*pi/180)


        theta = (ptheta*pi/180 + (self.out_right - self.out_left)/self.diameter) / pi*1
        # keep orientation in [-180, 180] range
        if theta > 180:
            theta = theta % (-180)
        elif theta < -180:
            theta = theta % 180

        return (x, y, theta)

    new_particles = [ getNewPose(x,y,theta) for x,y,theta in self.particles ]
    self.particles = new_particles

    with open("localization.out", "a") as fp:
        fp.write("Motion\n")
        [ fp.write(f"{x} {y} {theta}\n") for x,y,theta in self.particles ]
```

4 Conclusion

To sum up this project, we can conclude that the particle filter was not correctly implemented, the second version wasn't given a lot of attention and the first version was implemented with faults. These faults can be attributed to miscalculations and bad re-sampling as well as an overall implementation of the desired code and functions, which turned this assignment into a bigger task than intended that was refactored multiple times during development.