Problem Set 9: Particle Filters (Good Bug Hunting)
Computational Perception and Artificial Intelligence

Description:

The Particle Filter algorithm is related to the Genetic Algorithm and uses a model of natural selection to locate a target. The Particle Filter creates several hundred instances of "guesses" as to the location of the target. Then the filter moves these guesses in a random manner and takes a measurement from the sensor (in our case a camera and SSD template matching). Then, we the sensor data, the algorithm selects the particles (guesses) that 'survive' based on a system of weights of the distance to the sensor reading. The survivors become the new population of particles and the process repeats. The moment (mean location) of the particles represents the systems best guess as to the location of the target object. We will implement a version of the Particle Filter while searching for a 'toy' using live camera data. Many systems use Particle Filters to approximate sensor readings in noisy data environments. (Autopilots, GPS systems . . .)

Setup:

A. Download and unzip the following folder into your Perception Lastname directory.

http://www.nebomusic.net/perception/ps09.zip

B. Follow the requirements to complete the project.

Requirements:

Part A: Tracking Toy using the SSD template matching Algorithm

1. Given the course setup and 'toy' to track, take a picture of the 'toy' and create a patch to track via the SSD algorithm. The patch will need to be resized and cropped using Photoshop or other software. Save your image of the patch as `patch.png` in the input folder of ps09.

2. Complete the python module called `ps09_ssdfollow.py` where you track the movements of the 'toy' in the course in real time using the cv2.matchTemplate() function, your patch from step 1, and the SSD algorithm. The code must display the image from the video feed and place a green circle on the locations of the best match.

Part B: Tracking Toy combining SSD matching with Particle Filter

1. You will now implement a Particle Filter algorithm to track the 'toy' in the course. You are provided the following python modules:
   a. `robot.py` : A class modeling an individual robot object. This is the 'particle'
   b. `ParticleClassDemo.py`: An implementation of particles that only moves randomly without resampling.
   c. `particleDemo.py`: A module demonstrating implementation of camera and particles without resampling.
   d. `Particles.py`: This is the class you will complete to implement the Particle Filter
   e. `ps09.py`: You will complete this module to track the 'toy' using the particle filter algorithm

2. Complete the code in the Particle.py module to implement the Particle Filter algorithm. Make note of the comments indicating which sections of the code you are to implement. The Particle Filter algorithm is shown below for reference. Note that the Algorithm is spread throughout the Particle.py module in different parts of the class. The resampling step has been coded for you. (Using the Roulette Wheel algorithm).

```
Algorithm Particle Filter (N, targetPoint):

Initialize Particles P as empty Array
For i = 0 to N:
    Set random location x and y
    Set random direction 0 to 2*PI
    Set random velocity
    Set noise for velocity and turn
    Initialize a new robot r with location (x, y), velocity, and direction
    Append P with robot r

// Move each particle
For each particle p in P:
    Set p heading to random 0 to 2*pi
    Set p velocity to Gaussian random from current velocity
    Move each particle p by p.heading and p.velocity

// Measurement update and assign weight
Initialize weights W to empty List
For each particle p in P:
    error = 1
    Get (x, y) of particle p
    distance = distance between $(x_p,y_p)$ of p and measurement targetPoint $(x_{tp},y_{tp})$
    adjustedDistance = distance / 100.0
    error *= exp(adjustedDistance * -1)// reduces error to value between 0 and 1 in logarithmic scale
    Append W with error
```

```
// Resample (Code provided for this step)
// Keep the particles close by, let the distant particles "die"
Initialize newParticles as newP
index = int(random.random() * N)
beta = 0.0
mw = max(W)
 for i in range(self.N):
            beta += random.random()*2.0*mw
            while beta > w[index]:
               beta -= w[index]
               index = (index + 1) % self.N
            r = robot()
            r.x = self.p[index].x
            r.y = self.p[index].y
            r.heading = self.p[index].heading
            r.distance = self.p[index].distance
            newP.append(r)

        # Copy newP to self.p
        self.p = newP

// Get moment of Particles (Average x and y position)
Initialize avgPoint as tuple (x, y)
avgPoint[0] = average of X values of resampled particles
avgPoint[1] = average of Y values of resampled particles

Return P, avgPoint
```

3. Complete the code in ps09.py to implement tracking of the 'toy' using the Particle.py class. The comments in ps09.py will guide you to which sections to complete. The particleDemo.py module can serve as a model of how to read data from the Particle class instance and display the particles to the camera image. Your code should track the 'toy' in real time placing the particles as blue circles and the centroid (guess of location) as a red circle.

Part C: Turning Project In

1. Download Snaggit or another screen-casting tool. Make a demonstration video of your particle filter tracking the 'toy' and explain in the video how the code works to track the object. Be sure to cover the elements of the Particle Filter (Move Particles, Weight particles against input, Sample Particles from Weight). Publish this Screencast video to your Marist YouTube channel and email the link to Mr. Michaud.

2. Zip your ps09 folder and email to Mr. Michaud as an attachment.