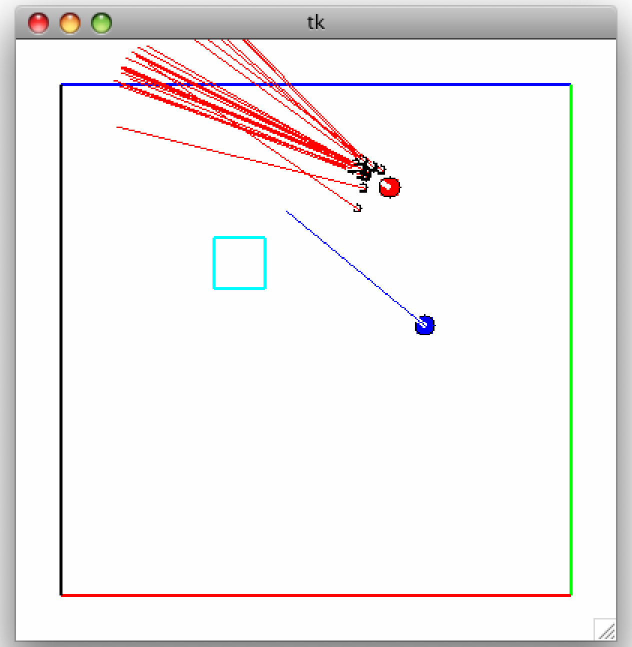
CMSC 360: Intelligent Robotics and Perception

Lab 3: Bats – Monte Carlo Localization

Due November 6, 2012



Localization Simulator
The red circle represents the robot's true location, the lower blue circle represents the estimated position based on odometry, and the small black circles represent the particle filter's estimates.

Overview

In the third unit of the course we are studying robots that do more thinking, in particular, they perform localization and planning. For this lab, we will explore Monte Carlo Localization. We'll use a Python simulator created by Zach Dodds at Harvey Mudd College. Our simulated robot has odometry, bumper, and range sensors. You'll improve on the built-in dead-reckoning localization system by implementing a particle filter localization system. Specifically, you'll have to complete the motion and sensor models.

Learning Objectives

- Understand Monte Carlo Localization
- Implement a sensor model.
- Implement a motion model.
- Experimentally evaluate a localization algorithm.

Understanding the Code

The primary file you will modify is `main.py`. In particular, you will modify the `apply_motion_model` and `apply_sensor_model` functions.

```
def apply_motion_model(self, particles, dx, dy, dt):
    '''Move the particles according to the motion model of the robot'''
    for p in particles:
        # convert the local coordinates back into global coordinates
        gdx = dx*math.cos(p[2]) - dy*math.sin(p[2])
        gdy = dx*math.sin(p[2]) + dy*math.cos(p[2])
        p[0] += gdx
        p[1] += gdy
        p[2] += dt

def apply_sensor_model(self, particles, realpose):
    '''Use the range sensors to set the weight of the particles'''
    # get sensor measurement from range sensor
    # orientation is in DEGREES in realpose
    rd, rr, rg, rb = self.getRangeData(realpose[0],
                                       realpose[1],
                                       math.radians(realpose[2]))
```

Each particle is composed of an estimate of the robot's pose (x, y, theta), the estimate's weight, and the ray-traced range from the particle to the map's walls. Each particle is represented by a list of five items: `[x, y, theta-in-radians, weight, range-data]`. The `realpose` is represented by a list of three items: `[x, y, theta-in-degrees]`. Obviously, you should not use `realpose` in estimating the robot's position. since it wouldn't be available on a real robot. You will need to pass this information to `getRangeData()` to get the range measurements at the robot's true pose.

The `getRangeData(x, y, thr, rangeSensorHeadingd=0)` function takes a pose (and an optional direction (in degrees) in which the range sensor is pointing) and returns a four-tuple of the distance to the environment and the r, g, and b components of the color of that nearest obstacle. One warning: the units are not consistent – the particle pose is expected in radians and the sensor heading in degrees. Fortunately, math.radians and math.degrees are available.

You can initialize the particle filter, by pressing the 'P' key. The 'Q' key quits the program.

You will probably find the random[1] module useful. Specifically, look into the von mises and gaussian routines.

Deliverables

- Complete the motion and sensor models. Submit your `main.py` file.
- Write a paragraph or two explaining your motion and sensor models.
- Write a paragraph describing how the resampling function works.
- How well does your localization routine work? Specifically, compute the error of your localization routine as well as the error of odometry only, how do they compare?
- If you had more time, how would you improve your localization system?

[1] http://docs.python.org/library/random.html