

91.451, Robotics II
Prof. Yanco
Spring 2003

Lab 4: More Behavior Based Control

Out: Thursday, 6 March 2003

Due: Wednesday, 12 March 2003

(Note: Check off hours on Wednesday changed to 11:30 – 2:00 due to a faculty candidate talk.)

Midterm: The midterm exam will be held in class on Tuesday, 25 March. This lab will be graded and returned to your lab benches by Thursday, 20 March. (I'll e-mail the class when the graded lab has been put on everyone's desks.) For the midterm, you can bring one sheet of 8.5"x11" paper with whatever notes you'd like to write on it.

Office Hours for Next Three Weeks:

Tuesday, 11 March	12:45 – 2:00
Wednesday, 12 March	11:30 – 2:00
Thursday, 13 March	No office hours: Jordan Pollack talk from 3:00 – 4:00
Spring Break	No office hours, but will be around. Send e-mail.
Tuesday, 25 March	No office hours. Send e-mail if questions.
Wednesday, 26 March	No office hours. Send e-mail if questions.
Thursday, 27 March	2:00 – 3:00

Overview: In this lab, you'll work more with behavior based control in Pyro. We'll start with another fuzzy logic based program, then move to behavior sequencing.

There are two methods of using behaviors: sequences and blends. To use blending, you need only have multiple behaviors active simultaneously. These are usually in the same state, but could be in different states.

Part I: Blending behaviors using fuzzy rules

In Lab 3, you wrote code for wall following that used a set of fuzzy rules to produce the desired behavior. In this lab, modify your person following code from Lab 2 to use fuzzy rules instead of direct control.

For this part, show me the robot running your code and turn in your well commented code.

Part II: Behavior sequencing

The easiest method to produce sequencing is to go from one state to another. Here is an example called BBSquare.py (the code is available on the web site and should also be in your brains directory):

```
# BBSquare.py
# A Behavior sequencing sample
# D.S. Blank

# This Pyro example will go (roughly) in a square

# This example has two states, "edge" that goes straight, and "turn"
# that turns 90 degrees to the left. It bounces back and forth between
# these two states.

# Note how it uses onActivate() to remember where it was when it
# started in both cases. It then moves (forward or to the left) until
# it has moved enough.

from pyro.brain.fuzzy import *
from pyro.brain.behaviors import *
from pyro.brain.behaviors.core import * # import distance function

import math
from random import random

class TurnLeftBehavior (Behavior):
    def init(self):
        self.Effects('rotate', .1)
        self.Effects('translate', .1)

    def update(self):
        self.IF(1, 'rotate', .1)
        self.IF(1, 'translate', 0)

class StraightBehavior (Behavior):
    def init(self): # method called when created
        self.Effects('translate', .1)
        self.Effects('rotate', .1)

    def update(self):
        self.IF(1, 'translate', .1)
        self.IF(1, 'rotate', 0)
```

```

class edge (State):
    def init(self):
        self.add(StraightBehavior(1))

    def onActivate(self): # method called when activated or gotoed
        self.startX = self.getRobot().get('robot', 'x')
        self.startY = self.getRobot().get('robot', 'y')

    def update(self):
        x = self.getRobot().get('robot', 'x')
        y = self.getRobot().get('robot', 'y')
        dist = distance( self.startX, self.startY, x, y)
        print "actual = (%f, %f) start = (%f, %f); dist = %f" \
              % (x, y, self.startX, self.startY, dist)
        if dist > 1.0:
            self.goto('turn')

class turn (State):
    def init(self):
        self.count = 0
        self.add(TurnLeftBehavior(1))

    def onActivate(self):
        self.th = self.getRobot().get('robot', 'th')

    def update(self):
        th = self.getRobot().get('robot', 'th')
        print "actual = %f start = %f" % (th, self.th)
        if angleAdd(th, - self.th) > 90:
            self.goto('edge')

def INIT(robot): # passes in robot, if you need it
    brain = BehaviorBasedBrain({'translate' : robot.translate, \
                                'rotate' : robot.rotate, \
                                'update' : robot.update }, robot)

    # add a few states:
    brain.add(edge(1))
    brain.add(turn())

    brain.init()
    return brain

```

A. The square code doesn't avoid obstacles; what must you do to do that? Make the required change so that the robot will avoid obstacles while trying to make a square. What happens if the robot is interrupted during its square to avoid an obstacle?

B. BBSquare.py doesn't use very sophisticated behaviors for turning or moving. It would be better, for example, if the turning slowed down when it got closer to its desired angle. How would you do that? Make the change to BBSquare.py.

For A and B, turn in your well commented code and show me your code running on the robot. You can do both exercises in the same program.