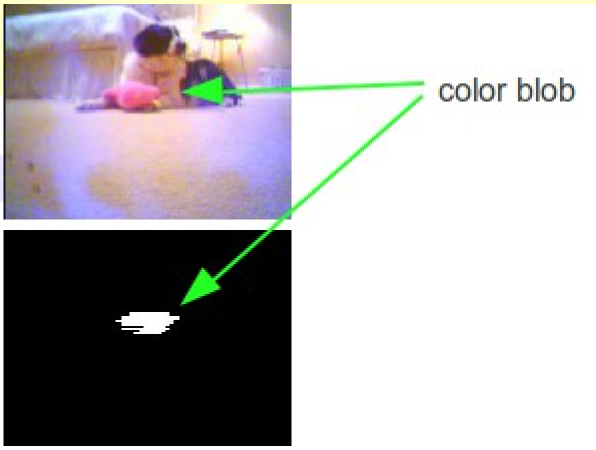
CMSC 251
HW #1: Behavior-Based Robot Systems
Due: 18.Sept.2017



Learning Objectives

- Review object-oriented Python
- Develop a behavior-based robot controller to solve a navigation problem with weak sensors.
- Use finite state machines to aid in design.

This homework will ask you to create more complex behaviors for your robot. You should start by downloading controller.py. Create a directory called hw01 and place the code in it. If you are using minimal Myro, then I **strongly recommend** you down it from http://github.com/drablab/mmyro and place the myro directory inside hw01. This will allow you to use the most current version of the mmyro library.

Goals

You are to improve upon a basic behavior-based robot controller, implemented using a single Myro/Fluke-2/Scribbler robot, to solve the problem of efficiently pushing an object to the edge of a small rectangular space in the presence of arbitrarily place obstacles and other robots.

Your grade will be based on the quality of your code (60%), commenting (10%), and quality of your working solution as determined by the points you score in competition with other robots (30%).

Task Details

The environment for your robot is a rectangular playing field delimited by PVC piping situated in RKC 107 near the north window with the window shades down. The environment contains 2 other robots. All robots are randomly placed near one another facing a border. The task your robot will complete is to push a fluorescent orange pylon that is internally lit until the pylon touches any edge of the field. There is also one randomly placed stationary obstacle in the field. Your robot will participate in two timed runs that are terminated after three minutes or when any robot completes the task. Points are accumulated as follows:

Description	Points
Task completed in less than 30 seconds	3
Task completed in less than 60 seconds	3
Task completed in less than 120 seconds	3
Task completed by end of round.	3
Pylon pushed at least two inches when Fluke is forward.	3
Pylon touched by robot.	2

Edge avoided at least once by robot.	2
No other robot hit.	2
Random obstacle not hit.	2

Each robot can only receive points in each category (row) at most one time per run. Your robot must use a behavior-based architecture for its entire solution. Only one minute is allowed before a run starts for you to ready your robot.

Guidelines

Starting with the code in controller.py, you will modify the existing avoid and wander behaviors within this file. You will then add whatever other behaviors you need to complete the task.

Before working with behaviors you should always make sure to validate your sensors. Before working with any sensors you should make sure that the battery has sufficient charge (usually above 7.5V). You can change the sensitivity of the IR sensors (returned by `getObstacle`) using the `setIRPower(VAL)` command. You will want to verify that you are getting usable readings from the IR sensors when various obstacles are near the Fluke, but don't expect them to be perfect—they never will be. When working with the camera I recommend you set the picture size to small using `setPicSize('small')`. This will enable quicker processing.

You should work with one behavior at a time to perfect the performance of that behavior. Work with simple reactive behaviors first and assume fixed priority arbitration. Once a behavior works, you can worry about combining it with other behaviors. Your behaviors should not access the inner state of one another. It is also optimal if each behavior operates fairly quickly and can be interleaved easily with other behaviors as percepts change.

I recommend using color blobbing as in the picture above, using the on-Fluke `getBlob` and/or `getPicure('blob'),` to find the pylon. Before you can reliably find the pylon you must configure the blob color you want to find. Of course, this can be lighting dependent.

For starters you might try the Y,U,V colors for the pylon that I recorded as in the command below. Here Y is the luminance, while U and V correspond to hue. Use `getPicture('blob')` to make sure the blob is being detected.

```
configureBlob(0, 254, 112, 117, 152, 165)
```

Ultimately, you will want to tune color-based blobbing for your robot and light conditions. If you know a rectangle on the image that contains the desired colors, you can use `robot.set_blob_yuv` to configure blobbing color. If, on the other hand, you know the YUV limits you'd like (printed by `set_blob_yuv`), then use `configureBlob`.

These two methods can be used to convert between RGB and YUV color coding.

```
def yuv2rgb(Y, U, V):
  R = int(Y + (1.4075 * (V - 128)))
  G = int(Y - (0.3455 * (U - 128)) - (0.7169 * (V - 128)))
  B = int(Y + (1.7790 * (U - 128)))
  return [max(min(v,255),0) for v in (R, G, B)]
```

```
def rgb2yuv(R, G, B):
  Y = int(0.299 * R + 0.587 * G + 0.114 * B)
  U = int(-0.14713 * R - 0.28886 * G + 0.436 * B + 128)
  V = int( 0.615 * R - 0.51499* G - 0.10001 * B + 128)
  return [max(min(v,255),0) for v in (Y, U, V)]
```

You might want to use getStall to determine when the robot is unsuccessful at freeing itself from an obstacle.

You should review the various robot commands at http://wiki.roboteducation.org/Calico_Myro to discover additional options for robot and Fluke control.

Submit your controller.py file on Moodle.