Agenda

- Sensors: research with Myro's senses(
- 1, 2 or 3 member team project:
  - Preliminary version due in 1 week (Wednesday)
  - Improved version due in 2 weeks (following Wed.)
  - Presentation at Parents/Prosp. Students day Sat, October 10.
- Project must give user a menu of choices of robot demonstrations.
- Lists and other program organization ideas to combine behaviors.
Scribbler: Myro Reference

Camera
- takePicture()
- takePicture("color")
- takePicture("grey")
- takePicture("blob")
- Image is 256x192.

Brightness
- getBright()
- getBright("left"/"center"/"right")
- getBright(0/1/2)
- Higher values imply brightness
- Lower values imply dark segments

Obstacles
- getObstacles()
- getObstacles("left"/"center"/"right")
- getObstacles(0/1/2)
- Values returned in 0..7000
- o implies all clear
- 7000 implies obstruction

Speaker
- beep(SECS, FREQ)
- beep(SECS, FREQ1, FREQ2)

Misc.
- getStall()
- getBattery()
- getName()
- setName(NAME)

Motors
- motors(LEFT, RIGHT)
- move(TRANS, ROT)
- forward(SPEED, SECS)
- backward(SPEED, SECS)
- turnLeft(SPEED, SECS)
- turnRight(SPEED, SECS)
- All values are in -1.0..1.0
- SECS can be any float value.

IR
- getIR()
- getIR("left"/"right")
- getIR(0/1)
- 1 implies all clear
- 0 implies obstruction

Light
- getLight()
- getLight("left"/"center"/"right")
- getLight(0/1/2)
- Values returned in 0..5000
- 0 implies very bright light
- 5000 implies darkness
TWO Kinds of Myro motor control functions

- motors(LSpeed, RSpeed)
- move(TranslationSpeed, RotationSpeed)
- translate(speed)
- rotate(speed)

What's the difference? In each case, what makes the robot stop?

---

The distinction is **VERY IMPORTANT**

The wait() function is very useful for combining custom motions activated by motors or move with a time that the motion should last. Similarly, for Python/Myro timing functions for EVEN MORE customized control.

```python
for I in range(0,10):
    print I
motors(1.0, 1.0)
```

```python
for I in range(0,10):
    print I
    forward(1.0, 5.0)
```
Light Sensors

- `getLight()` returns list of 3 integers `[ 1, 5, 103 ]`
- `getLight( 0 )`, `getLight( 1 )`, `getLight( 2 )`
- `getLight( 'left' )`, `getLight( 'center' )`, `getLight( 'right' )`
- Brighter light: lesser integer values: (but really, really dark: 0 )
- Directionality?
- Normalization dance: circle in place, take measurements, average them.
Bright(ness) Sensor

- Not a separate physical sensor. The Fluke does a calculation on the image from the camera.
- **YOUR JOB:** Read and RESEARCH as with light sensors.
- Your program can take a picture and then calculate something from the picture in a way YOU invent.
- `getBright( )`: call and return type API is just like `getLight( )` but the brightness-to-number dependency has OPPOSITE SIGN, and of course a different scale.
IR Obstacle Sensors

- Return value is 0 means IR light is bouncing back to the receiver (off of an obstacle)
- Return value is 1 means NO bounce-back detected.
IR Obstacle Sensors

- `getIR()` returns a list of two items [1,1]. You can also call `getIR("left")` to get just the left sensor, and similarly with `getIR("right")`. The function also accepts 0 and 1 as the parameter to select which sensor value to return.
Fluke Board – IR Obstacle sensors

- Larger numbers indicate that more IR light is bouncing back. Unlike getIR(), they have a WIDE RANGE.

- `[0, 1842, 0] = getObstacle()`

3 clear Infrared Light Emitting Diodes.

Big black eye
Fluke Board – Camera

- You can take a picture using the camera on the Fluke board.
- p = takePicture()
- show(p)
- SEE chapter 9 for lots of picture operations..

```python
>>> init()
You are using fluke firmware 2.6.2
You are using scribbler firmware 2.0.0
Hello, I'm Scribble!
>>> joyStick()
<myro.graphics.Joystick instance at 0x7f1
>>> p = takePicture()
>>> show( p )
```
HW Project Assignment—based on Georgia Tech's dance assignment..

This is largely what you did in Labs 1 and 2......use that experience as a base....

Of course, you’ll have to teach your robot to dance. Using the movement functions: http://wiki.roboteducation.org/Myro_Reference_Manual#Movement_Functions have your robot do a little dance. The dance should last for at least 30 seconds, and contain at least 3 distinct dance moves. Don’t just go back and forth for 30 seconds; vary the dance a bit. Pretend your robot is wellversed in rhythm and soul, or is at least a little spastic. In addition to the movement, your robot should also make some noise! The beep() function is very helpful – it allows the robot produce various tones. You are allowed (and encouraged) to make your own helper functions that contain individual dance moves.
Menu User Interface Requirement:

For this part of the assignment, you'll use conditionals ("if" statements) and a while loop to create a menu that allows the user to select a dance step. Name the function that contains this menu `menu()`, and save it in a file called `menu.py`. If you used helper functions in your `dance()` from the first part of the assignment, you can reuse those. (copy and paste them into the new file) If not, you'll need to create a few short functions that cause the robot to do a particular dance move for a few seconds. Your menu must implement at least 3 distinct dance choices, plus an option to exit the program. If the user types something invalid, you should let them know and print the menu again. Here's an example of how `menu()` might work. The user's input is shown in blue.

1. The Charleston
2. The Tango
3. The Foxtrot
0. Exit

Which dance step would you like? 1
(The robot does a dance move called the Charleston. The program doesn't show the menu again until this move finishes a few seconds later.)
User Interface Continued..

1. The Charleston
2. The Tango
3. The Foxtrot
0. Exit
Which dance step would you like? 5
I'm sorry, I don't know that one.
1. The Charleston
2. The Tango
3. The Foxtrot
0. Exit
Which dance step would you like? 0
Have a good day! (The danceMenu() function terminates.)
Some Project Requirements

- Make the first couple of choices very simple demos.
- Implement the action for EACH CHOICE by a SEPARATE FUNCTION.
- More complex choices must involve (1) the robot seeking light (2) the robot avoiding obstacles (3) a combination of seeking something AND avoiding obstacles.
More Project Requirements

- It must include a choice that provides a COMBINATION of behaviors controlled by the list-based control and arbitration decision structure from Kumar, Chapter 7.
Do This: Place your Scribbler on the floor, turn it on, start Python, and connect to it. Also connect the game pad controller and start the manual drive operation (joyStick() not gamepad() except if you have a graduated control). Next, issue the senses command to get the real time sensor display. Our objective here is to really "get into the robot's mind" and drive it around without ever looking at the robot. Also resist the temptation to take a picture. You can use the information displayed by the sensors to navigate the robot. Try driving it to a dark spot, or the brightest spot in the room. Try driving it so it never hits any objects. Can you detect when it hits something? If it does get stuck, try to maneuver it out of the jam! This exercise will give you a pretty good idea of what the robot senses, how it can use its sensors, and to the range of behaviors it may be capable of. You will find this exercise a little hard to carry out, but it will give you a good idea ...
Python Lists

- List: A data structure AND a Python type (like int, float, string, boolean)
- Create a list and then define variable X so that the value of X is that list:
  
  \[
  X = [\ 35, \ 'ABCD', \ aFunFun]\n  
  - Try:
    
    for xxx in X:
      print xxx
Orthogonality

A list can contain ANY TYPE of Python value: ints, floats, strings, boolean and even FUNCTIONs

(it can also contain lists...as in
[ [ ], [1], [1, 2], [1,2,3], [1,2,3] ]
)
This function implements the light-seeking behavior.

```python
def seekLight():
    L, C, R = getLight()
    if L < lightThresh:
        return [True, cruiseSpeed/2.0, turnSpeed]
    elif R < lightThresh:
        return [True, cruiseSpeed/2.0, -turnSpeed]
    else:
        return [False, 0, 0]
```

Note that lists are returned.
The program has a “three-way” if statement. I explained the elif clause in the way I explained while and if before. That was in terms of the SEQUENCE of STEPS that the statement makes the Python interpreter do.

FIRST STEP: The condition of the if clause (L < lightThresh) is evaluated.
    If it is False, the
SECOND STEP is: Evaluate the condition of the elif (for “else if”) clause ( R<lightThresh ).
    If that is False the THIRD step is execute the body of the else clause.

On the other hand, if the condition of the if clause evaluated to True, the
SECOND STEP is: Execute the body if the if clause.

(Now ask yourself, what is the sequence when the if clause gives value False but in the
SECOND step, the condition of the elif clause is True?)
# list of behaviors, ordered by priority (left is highest)
behaviors = [seekLight, avoid, cruise]

**this list contains FUNCTIONs!**

def main():
    while True:  # Infinite loop!
        T, R = arbitrate()
        move(T, R)
main()

# Decide which behavior, in order of priority
# has a recommendation for the robot
def arbitrate():
    for behavior in behaviors:
        output, T, R = behavior()
        if output:
            return [T, R]