Contact Information

- E-mail: holly@cs.uml.edu (best contact method)
- Phone: 978-934-3642
- Office: Olsen 206
- Lab: Olsen 304
- Office hours, held in lab:
  - Tuesdays 2:30 - 4:00
  - Thursdays 11:00 - 12:30
  - By appointment

Course Topics

- Sensors and Effectors
- Machine Vision
- Manipulation
- Robot Architectures
- Machine Learning
- Evolutionary Robotics
- Mapping and Localization
- Planning and Reasoning
- Multi-Agent Robotics
- Human-Robot Interaction
- Current Research in Robotics

Readings

- Other readings will be distributed in class or on the website
Course Structure

- Tuesdays, 1:00 - 2:15: lecture in Olsen 415
- Thursday, 1:00 - 2:50: lab in Olsen 304

Labs

- Pioneer robots
- Player/Stage simulator
- Programmed in Pyro
- Labs must be checked off before class starts on due date

Exams

- The midterm will be given in class on Tuesday, 22 March
- The date of the final exam will be determined by the registrar

Project

- Topics selected by students
- Discussed with Holly
- Large part of the lab work in the second half of the course will be spent developing projects
Grading

• Grading based on labs (30%), midterm (20%), project (30%) and final (20%)

Today’s Lecture

• Sensors
• Locomotion
• Python tutorial
• Pyro
• Introduction to this week’s lab

Sensors

• Also known as transducers
• Measure internal or external state
• Internal state, also called proprioception
  – Battery level
  – Wheel rotations
  – Position of gripper
• External state:
  – Location of objects with respect to the robot
  – Color of objects

Sensors

• Active
  – Emit energy into the environment to make observations
  – E.g., infrared, sonars
  – Don’t confuse active sensors (devices that emit energy) with active sensing (e.g., pan-tilt camera)
• Passive
  – Receive already existing energy from environment
  – E.g., bump sensor, video camera (without additional lighting)
Sensors

• Sensors which measure same form of energy and process it in similar ways form a modality
• Modalities refer to the raw input
  – Sound
  – Pressure
  – Temperature
  – Light

Types of Data Returned

• Range (distance to objects)
• Absolute position (e.g., latitude and longitude)
• Environmental (e.g., temperature or color of objects)
• Inertial (e.g., acceleration)

Sensors: Inertial Sensors

• Accelerometers
  – Measure acceleration in a single direction
• Gyroscopes
  – Measure change in orientation
• Compasses
  – Measure position relative to magnetic north
• Inclinometers
  – Measure orientation of the gravity vector

Sensors: Global Positioning

• Global Positioning Sensors (GPS) use satellite readings to determine location
• Differential GPS (DGPS) uses a base station to eliminate the intentional error in the satellite signals
Ranging Sensors: Sonar

- Sends an acoustic pulse and listens for echo
- Can determine how far away an obstacle is based upon time of flight and speed of sound
- Can have problems with specular reflections

Ranging Sensors: Infrared

- Fast and inexpensive proximity sensors
- Emit an infrared pulse and detect the reflected signal
- Infrared can be washed out by bright light

Ranging Sensors: Laser Ranging

- Uses one of the following methods
  - Triangulation (relationship between outgoing light beam, incoming ray and its position on the film plane)
  - Time-of-flight (similar to sonar or infrared)
  - Phase-based (based on difference between phase of emitted and reflected signals)
- Low-power ranges a few meters
- High-power can range over a kilometer

Sensors: Cameras

- Images from cameras must be processed in order for the robot to use the information
  - Edge detection
  - Tracking movement
  - Finding particular colors
  - Distance of objects from the camera
- We’ll talk more about machine vision during the term
Problems with Sensors

- Noisy
- Return an incomplete description of the environment
- Can not be modeled completely

- We will come back to these issues throughout the term, particularly when we talk about mapping

Constructing a Sensor Suite

Consider the following for each sensor:
1. Speed of operation
2. Cost
3. Error rate
4. Robustness
5. Computational requirements
6. Power, weight and size requirements

Attributes of a Sensor Suite

Consider for entire suite:
1. Simplicity
2. Modularity
3. Redundancy
   - A sensor might be noisy or inaccurate by itself
   - False positives / False negatives
   - Complementary sensors can be used to provide disjoint information about a percept

Fusing Sensor Data: Sensor Fission
Fusing Sensor Data: Action-Oriented Sensor Fusion

- Sensor
- Percept
- Fusion
- Percept
- Behavior
- Action

Fusing Sensor Data: Sensor Fashion

- Sensor
- Percept
- Sequence Selector
- Percept
- Behavior
- Action

Locomotion

- Terrestrial
- Aquatic
- Airborne
- Space

Locomotion: Wheels

- Differential Drive
  - Two wheels mounted on a common axis controlled by separate motors
- Synchronous Drive
  - Usually three wheels, all of which can be driven and steered
- Ackerman Steering
  - Car drive: front steering wheels turn on separate arms
Locomotion: Tracks

- Usually two treads
- May have ability to change shape

Locomotion: Legs

- Varying numbers of legs: 1, 2, 4, 6, etc.
- Usually more difficult to control than wheeled robots: need to consider gait and balance
- Can traverse areas that wheels can not

Python

- Interpreted language
- Dynamically typed
- Object oriented

Following is a quick Python tutorial. For more on Python, see the reference manuals in the lab or How to Think Like a Computer Scientist, linked on the course web site

```python
principal = 1000
rate = 0.05
numyears = 5
year = 1

while year <= numyears:
    principal = principal * (1 + rate)
    print "%3d %0.2f" % (year, principal)
    year += 1
```
Python Tutorial

• Indentation matters! (No specific amount, but must be consistent within a block.)
• No semi-colons are needed (but can put them in)
• `print` is similar to C’s `printf`

Python: Conditionals

and, or, not for boolean expressions
`==` for testing equality

```python
if a<b:
    print a
else:
    print b
if a<b:
    pass #do nothing
else:
    print b
if a<b and a<c:
    print a
elif b<a and b<c:
    print b
else:
    print c
```

Python: Strings

```python
a = 'Hello World'
b = "Python is fun"
c = """Multi-line string, either with single or double quotes""
```

Indexed starting at 0

```python
a[4] is o
a[0:5] is Hello
a[6:] is World
a[3:8] is lo Wo
```

Python: Strings

Concatenated with `+`

```python
d = a + ". " + b
```

Can convert other data types to string using `str(x)`
`repr(y)`
```
y` #same as repr
```

If use `repr` to convert, can use `eval()` to turn back into original data type
Python: Lists

names = ["Joe", "Harry", "Mary", "Jill"]
Indexed starting at 0
names[2] is Mary
names[0] = "Bob" changes first name from Joe to Bob
Length of a list: len(<list>)

Python: Lists

Use append method to append new items to a list:
   names.append("Kate")
To insert an item to a list:
   names.insert(2, "Bill")
names[0:2] is ["Bob", "Harry"]
names[2:] is
   ["Bill", "Mary", "Jill", "Kate"]
+ concatenates lists

Python: Lists

Lists can consist of any type of Python object, including other lists
Referencing nested lists: a[1][3] is the fourth element of the second element of a list

Python: Tuples

Tuples are like lists but you can’t modify individual elements or append new elements
a = (1, 3, 5, 7)
person = (first, last, phone)
Python: Loops

for i in range(1, 10)
    print i

range(i, j) constructs a list of integers with values
from i to j-1
If i is omitted, assumed to be 0, e.g. range(5)
Optional stride can be third element
range(0,14,3) is [0, 3, 6, 9, 12]
range(8,1,-1) is [8, 7, 6, 5, 4, 3, 2]

Python: Ranges

If creating a large sequence, e.g.
    range(1:1000000), use xrange
xrange is more efficient because it doesn’t
compute the next element until needed
range constructs a list and populates it with
values

Python: Dictionaries

a = { “username” : “holly”,
      “home” : “/home/holly”,
      “uid” : 500 }

To access members of a dictionary
u = a[“username”]
d = a[“home”]
Python: Dictionaries

Inserting and modifying dictionaries:
```
a[“username”] = “yanco”
a[“shell”] = “/usr/bin/tsch”
```
Strings are the most common key in dictionaries, but you can use other Python objects like numbers and tuples.

Python: Dictionaries

Dictionary membership tested with `has_key()` method
```
if a.has_key(“username”):
    username = a[“username”]
else:
    username = “unknown user”
```
More compact:
```
a.get(“username”, “unknown user”)```

Python: Dictionaries

To get a list of dictionary keys:
```
k = a.keys()
```
To remove an element of a dictionary:
```
del a[“username”]
```

Python: Functions

Use def statement to create
```
def remainder(a, b):
    q = a / b
    r = a - q * b
    return r
```
To invoke:
```
result = remainder(37, 15)
```
Use a tuple to return multiple values from a function
def divide(a, b):
    q = a / b
    r = a - q * b
    return (q, r)

Function call:
quotient, remainder = divide(1456, 33)

Can invoke functions with arguments in arbitrary order
divide(b=3, a=124)

Variables created or assigned inside a function have local scope
To modify the value of a global from inside a function, use the `global` statement

```
a = 4
...  
def foo():
    global a
    a = 8
```

```
class Stack:
    def __init__(self):
        self.stack = []
    def push(self, object):
        self.stack.append(object)
    def pop(self):
        return self.stack.pop()
    def length(self):
        return len(self.stack)
```
Python: Classes

__init__ is a special method
Used to initialize an object after it’s created

Python: Classes

Other list methods:
- list(s) - Converts sequence s to a list
- s.append(x) - Appends a new element x to end of s
- s.extend(t) - Appends a new list t to the end of s
- s.count(s) - Counts occurrences of x in s
- s.index(x) - Returns smallest I where s[I]==x
- s.insert(i,x) - Inserts x at index I
- s.pop([i]) - Returns the element I and removes it from the list. If I is omitted, the last element is returned
- s.remove(x) - Searches for x and removes it from s
- s.reverse() - Reverses items of s in place
- s.sort(<cmpfcn>) - Sorts items of s in place. cmpfun is a comparison function

Python: Classes

s = Stack()         #create a stack
s.push(“Dave”)    
s.push(42)
#s.push([3, 4, 5])
x = s.pop()         #x gets [3,4,5]
y = s.pop()         #y gets 42
del s               #destroys s

Python: Modules

Can put definitions in a file and use them as a module that can be imported into other programs and scripts
File must have a .py extension

#file: div.py
def divide(a, b):
    q = a / b
    r = a - q * b
    return (q, r)
To use the module in other programs, use import statement:

```python
import div
a, b = div.divide(2305, 29)
```

Can import with a different name:

```python
import div as foo
a, b = foo.divide(2305, 29)
```

To import specific functions:

```python
from div import divide
a, b = divide(2305, 29)
```

To load all contents:

```python
from div import *
```

dir() function lists the contents of a module and can be useful for interactive experimentation

```python
>>> import string
>>> dir(string)
```

Pyro

- Stands for **Python Robotics**
- Pyro development is a joint project of Bryn Mawr College, Swarthmore College and UMass Lowell
- Pyro allows the low level robot control to be abstracted away
- The same Pyro program can run on different robots and different simulators
Pyro

```python
# A simple brain
from pyro.brain import Brain

# Define the robot's brain class
class SimpleBrain(Brain):
    # Only method you have to define is the step method
    def step(self):
        self.robot.translate(0.3)  # go forward

    # Create a brain for the robot
    def INIT(engine):
        return SimpleBrain('SimpleBrain', engine)
```

Sonar Sensors on the Pioneer

- Default units returned from sonars is robot units
- Can read individual sonars or groups of sonars
- You’ll learn more about this in lab

Pyro: Simple Obstacle Avoidance

```python
from pyro.brain import Brain

class Avoid(Brain):
    def wander(self, minSide):
        robot = self.robot  # if approaching an obstacle to the left side, turn right
        if min(robot.get('robot/sonar/front-left/value')) < minSide:
            robot.move(0,-0.3)
        # if approaching an obstacle to the right side, turn left
        elif min(robot.get('robot/sonar/front-right/value')) < minSide:
            robot.move(0,0.3)
        else: # go forward
            robot.move(0.5, 0)

    def step(self):
        self.wander(1)

    def INIT(engine):
        return Avoid('Avoid', engine)
```

This Week’s Lab

- Learning to use Pyro
  - With a simulator called Player/Stage
  - On a Pioneer robot
- Will use direct control, which directly links the sensor readings to the robot’s actions
- Meet in Olsen 304 on Thursday