Pyro Workshop 2005

Douglas Blank, Bryn Mawr College
Deepak Kumar, Bryn Mawr College
Lisa Meeden, Swarthmore College
Holly Yanco, UMass Lowell

This work is supported by the National Science Foundation: “Beyond LEGOs: Hardware, Software, and Curriculum for the Next Generation Robot Laboratory,” NSF CCLI DUE-0231363.

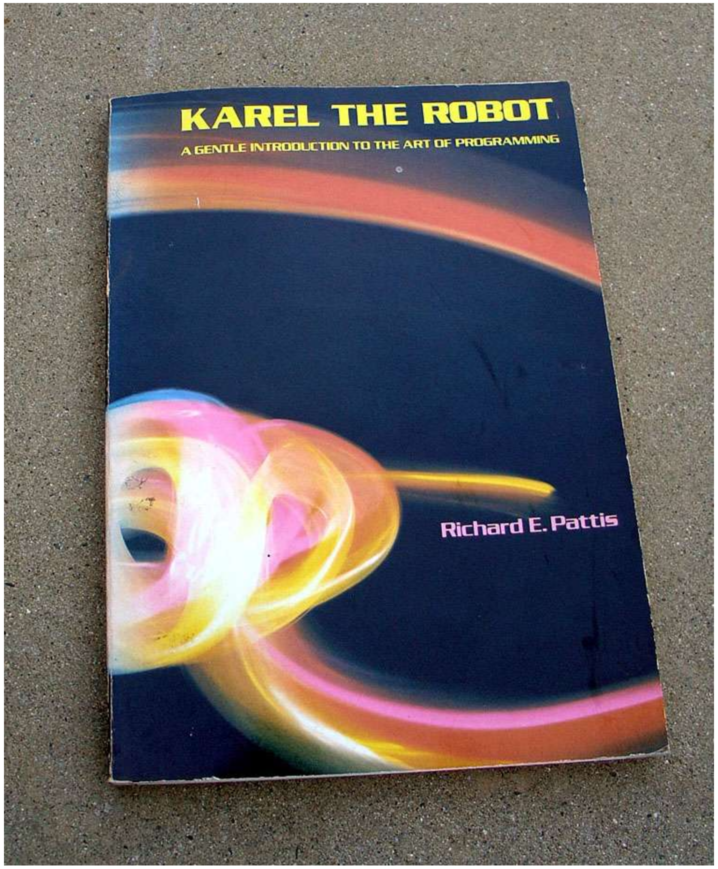
Motivations and Goals

Doug Blank

“Robotics wouldn’t work in our department.”

- Karel, Jeroo, LOGO, Alice, hunting the wumpus, etc: all robot control problems
- We’re interested in teaching AI concepts using a robotics paradigm
- Push the big ideas in so that they are the motivations for learning to program and “doing” computer science

Karel the Robot



- Small set of concepts
- Small set of actions
- Makes it easy to learn procedural programming
- ...But, loses utility for more advanced topics

“Ok, LEGOs will do just fine.”

- LEGO limitations? “Only your imagination”
- Limited sensors. Vision is a great motivation to study 2D arrays, but requires a camera.
- More complex models require more memory and faster CPU (for example, neural networks, developing area maps, planning, etc)
- Need to provide real AI and robotics research opportunities, which LEGO can’t do

Real Robotics: Real Painful Robotics

- Sophisticated, medium-cost robotics are now available (ActivMedia, iRobot, K-Team, Sony, and many more); however:
  - Vertical learning curve
  - Use their proprietary programming environment and control software, or write your own
  - Control software usually tightly integrated to a particular framework

Project Goals

- Provide a well-supported research-level hardware platform
- Build an open-source software system that abstracts from robot-specific details and enables exploration of high-level robot control strategies.
- Design a project-based curriculum integrated with the hardware and software.

Challenge

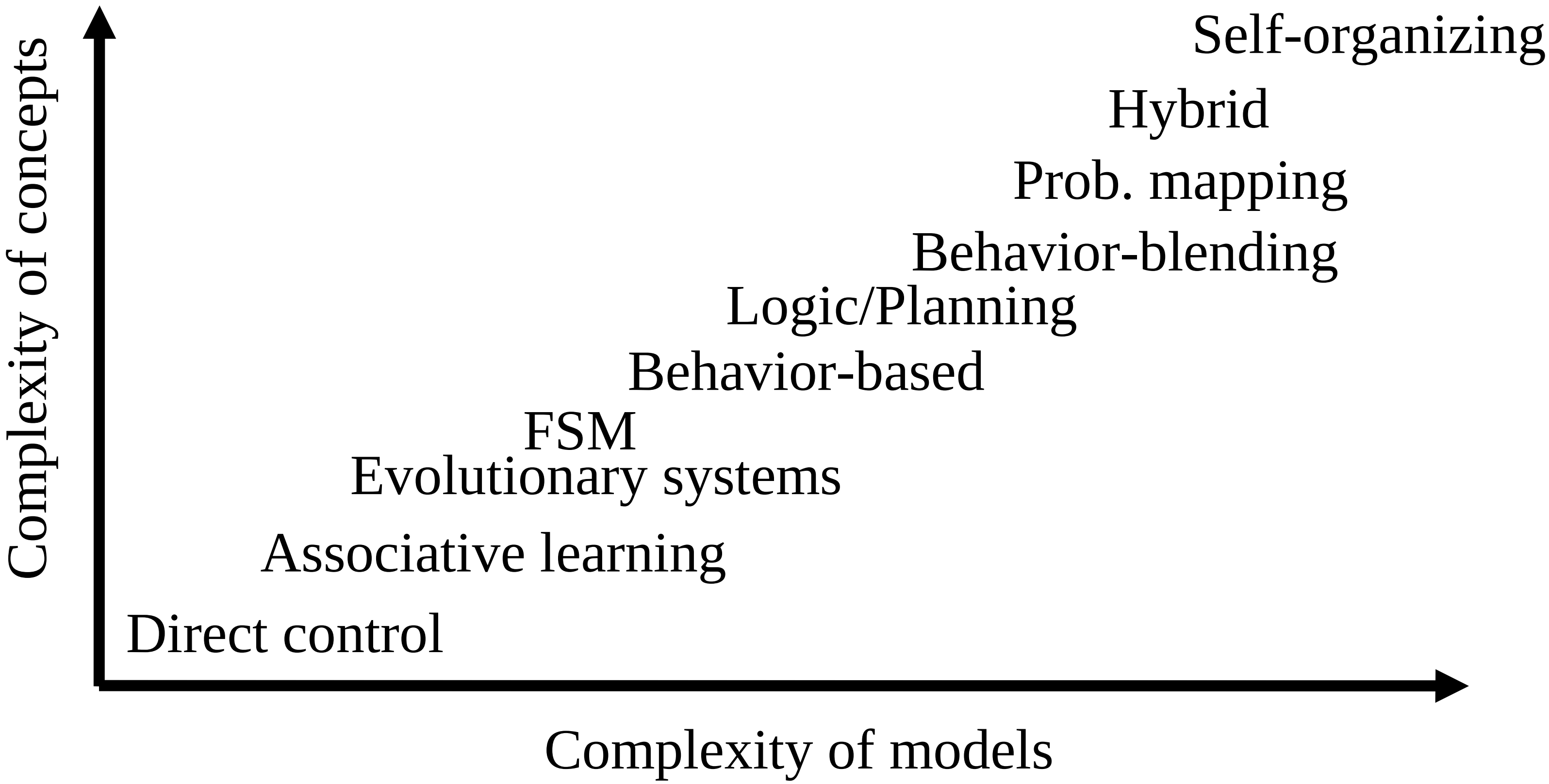
How can we start simply,

yet retain the utility of learned concepts as we explore more advanced topics?

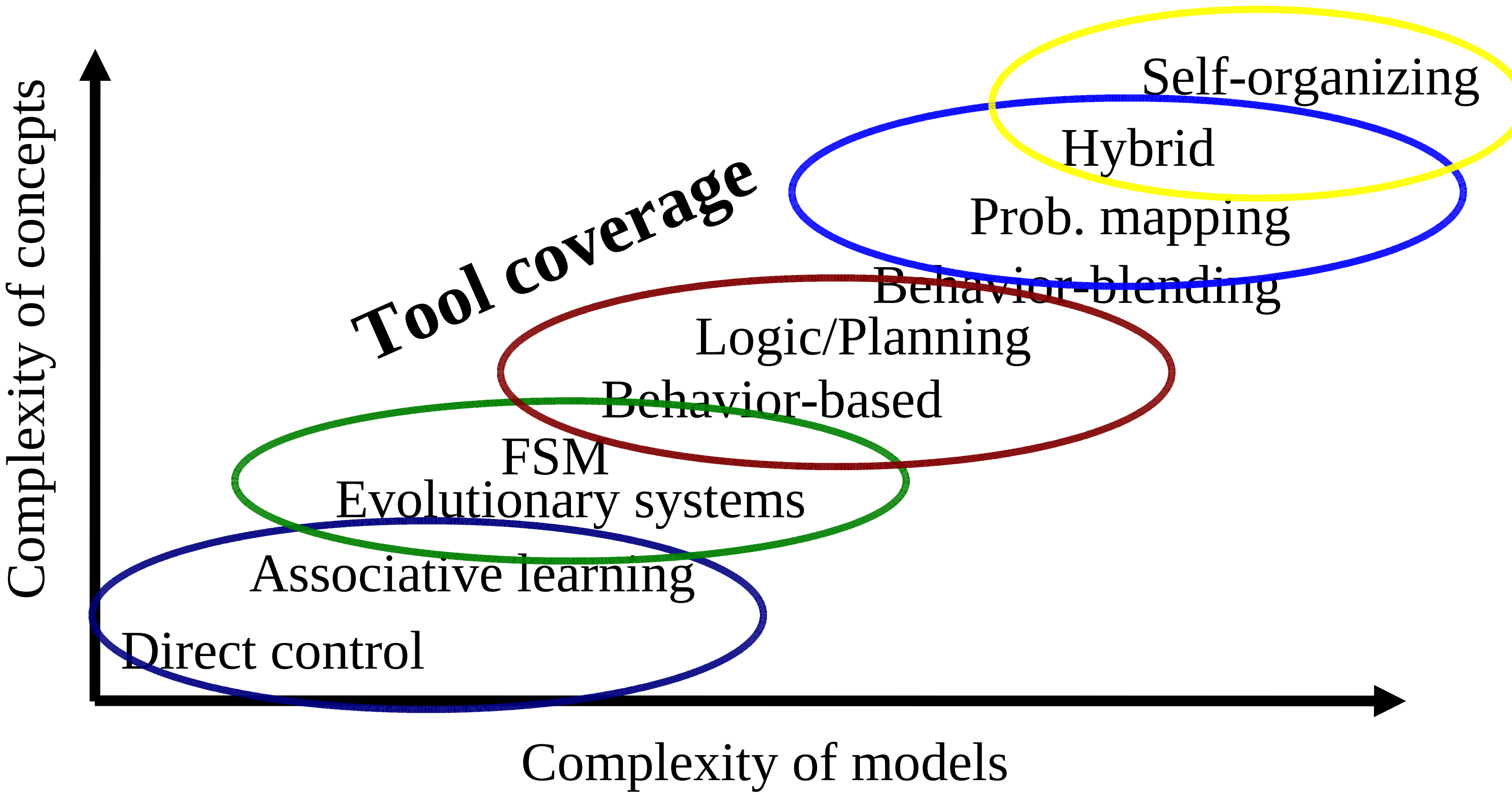
Pedagogical Scalable Frameworks (PSF)

- Creates a single, unifying architecture
- Composed of uniform and consistent conceptualizations
- Reduces “cost of learning” for the student
- Can be extended rather than abandoned

Robotics Topics in the Classroom



Robotics Topics



What is Pyro?

- **Py**thon **Ro**botics
- Programming environment for advanced topics
  - Mobile Robotics
  - Artificial Intelligence
- Architecture independent
  - Robot architectures are often robot specific
  - Architectures are often difficult to learn
  - Architectures are often VERY different from each other
- Powerful research tool
- Open source
  - Easy to add functionality
  - Easy to study underlying system
  - FREE!!

Pyro

- Written in Python
- Easy to learn
- Works on a variety of real and simulated robots
- Programs work unchanged across robotics platforms
- Large number of curriculum modules

PYRO Python Robotics

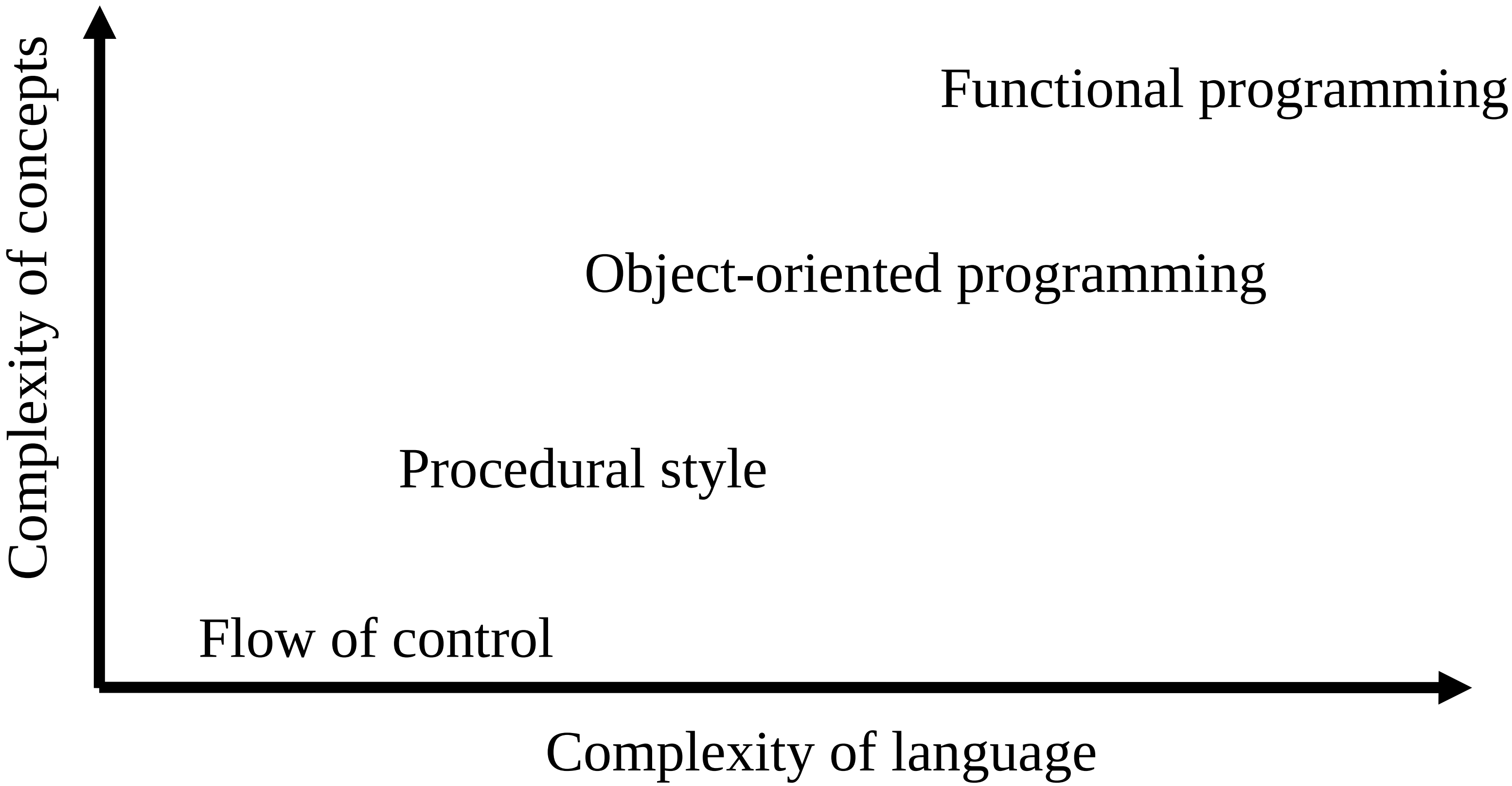
What is Python?

- 10+ years old; developed with learners in mind
- Clean syntax; Interpreted, but fast
- Supports multiple styles of programming (procedural, event oriented, threads, object oriented, and some functional)
- Support for most of the modern standards (XML, SOAP, OpenGL, HTTP, etc.) through libraries

Why Python?

- Interpreted language
  - Direct interaction with robots
- Platform independent
  - Portability
  - Simplify research done on multiple platforms
- Simple yet powerful
  - Easy to learn
  - Easy to use
  - Similar to pseudo-code
- Easily extended by other languages
  - SWIG

Python as a PSF



Python

- Looks like pseudo-code
- Indentation matters
- Object system built on top of functions
- Large collection of libraries
- Interactive
- Can be easily extended by other languages (via SWIG)

Hello, Python

```
print "Hello World!"
```

Hello, Python

```
def display():
    print “Hello World!”

display()
```

Hello, Python

```
def display(msg):
    print msg

display("Hello World!")
```

Hello, Python

```
class Greeter:
   def display(self, msg):
       print msg

x = Greeter()
x.display("Hello World!")
```

Python Example

- No curly braces, just indentation
  - Whitespace matters!
- Constructor named __init__
- Automatic typing
- Also supports easy multiple inheritance

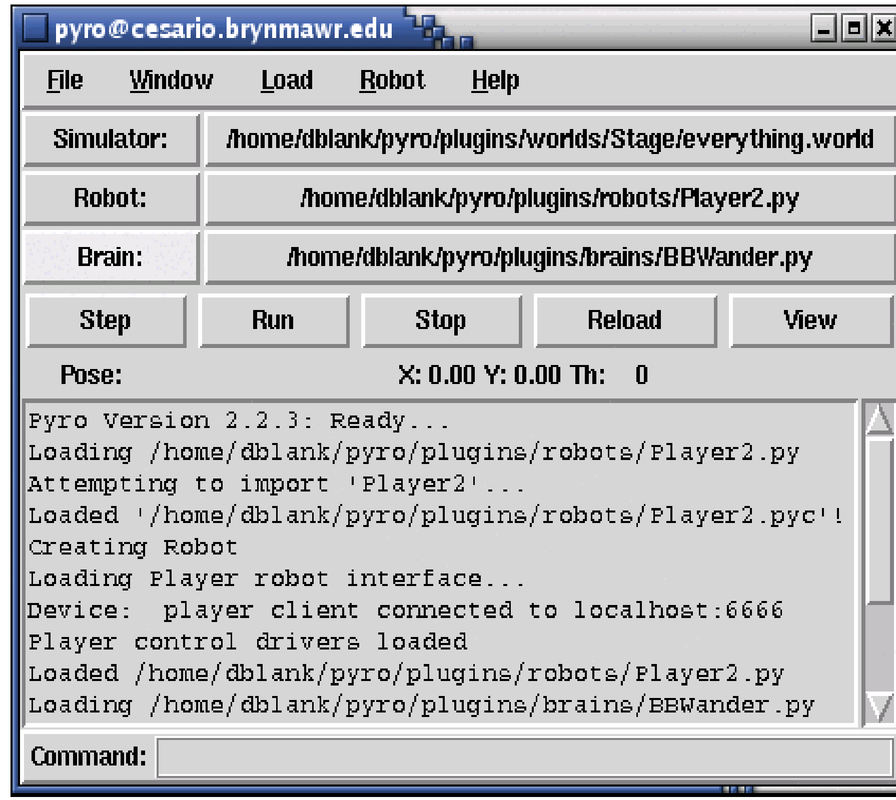
```
from math import sqrt

class Point:
    def __init__(self, myX = 0, myY = 0):
        self.x = myX
        self.y = myY

class Line:
    def __init__(self, pointA, pointB):
        self.a = pointA
        self.b = pointB

    def len(self):
        return sqrt( (self.a.x - self.b.x) ** 2 +
                     (self.a.y - self.b.y) ** 2 )

p1 = Point(5, 6)
p2 = Point(11, 23)
line = Line(p1, p2)
print "Line is ", line.len(line), "meters long."
```

Architecture and Abstractions

Deepak Kumar

Pyro: Python Robotics

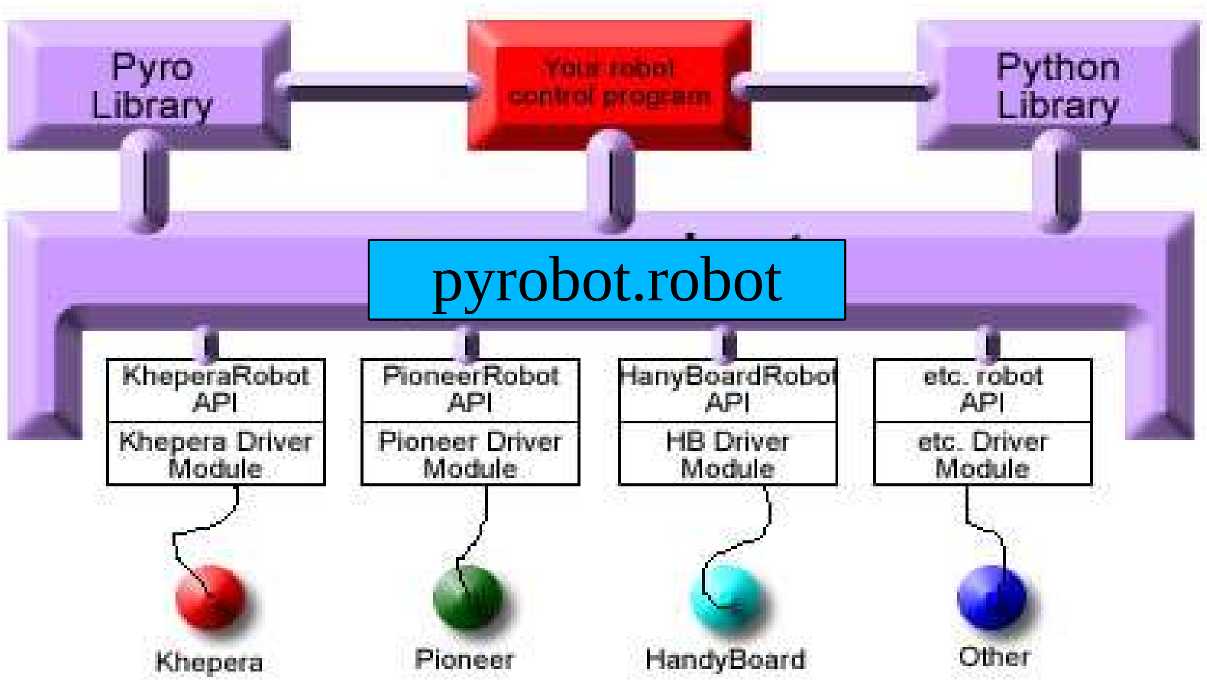
- Core written in Python
- Set of libraries and objects in Python
- API and GUI
- Open Source
- Easy for beginners to pick up
- Extendible



Pyro Design

- Make it a PSF
- Work on variety of robots and simulators
- Library of objects:
  - Robot, Controller, Sensors
- “Pythons all the way down”
- Usable for teaching and research

Pyro Architecture

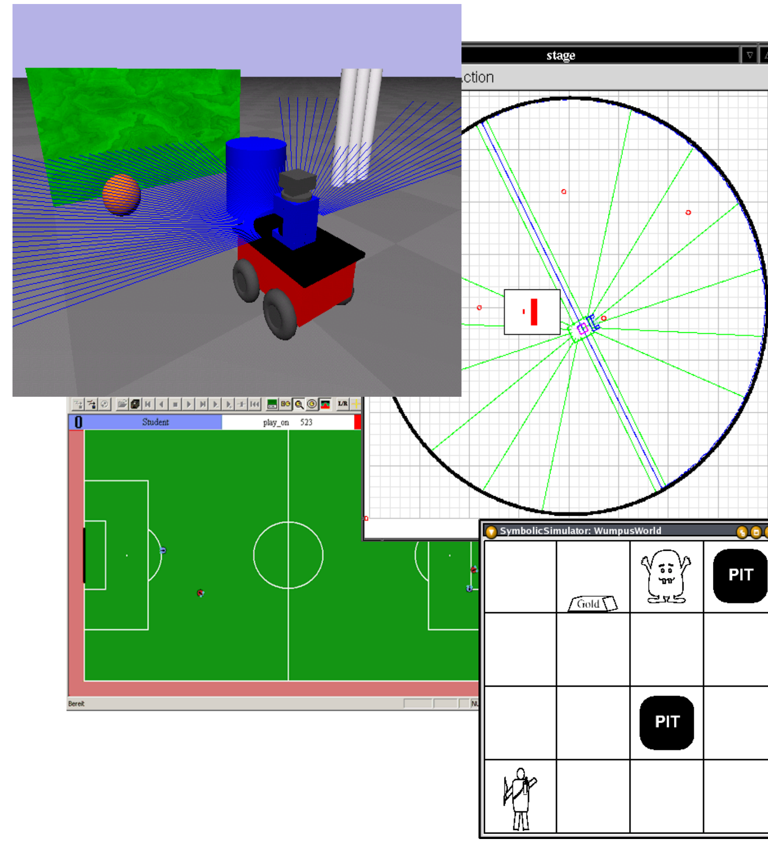


Supported Robots

- ActivMedia
  - Pioneer Robots
  - PeopleBot
- K-Team
  - Khepera
  - Hemisson
- Evolution Robotics
  - ER1
- Sony
  - Aibo Robots
- Others
  - Easy to add support for new robots

Supported Simulators

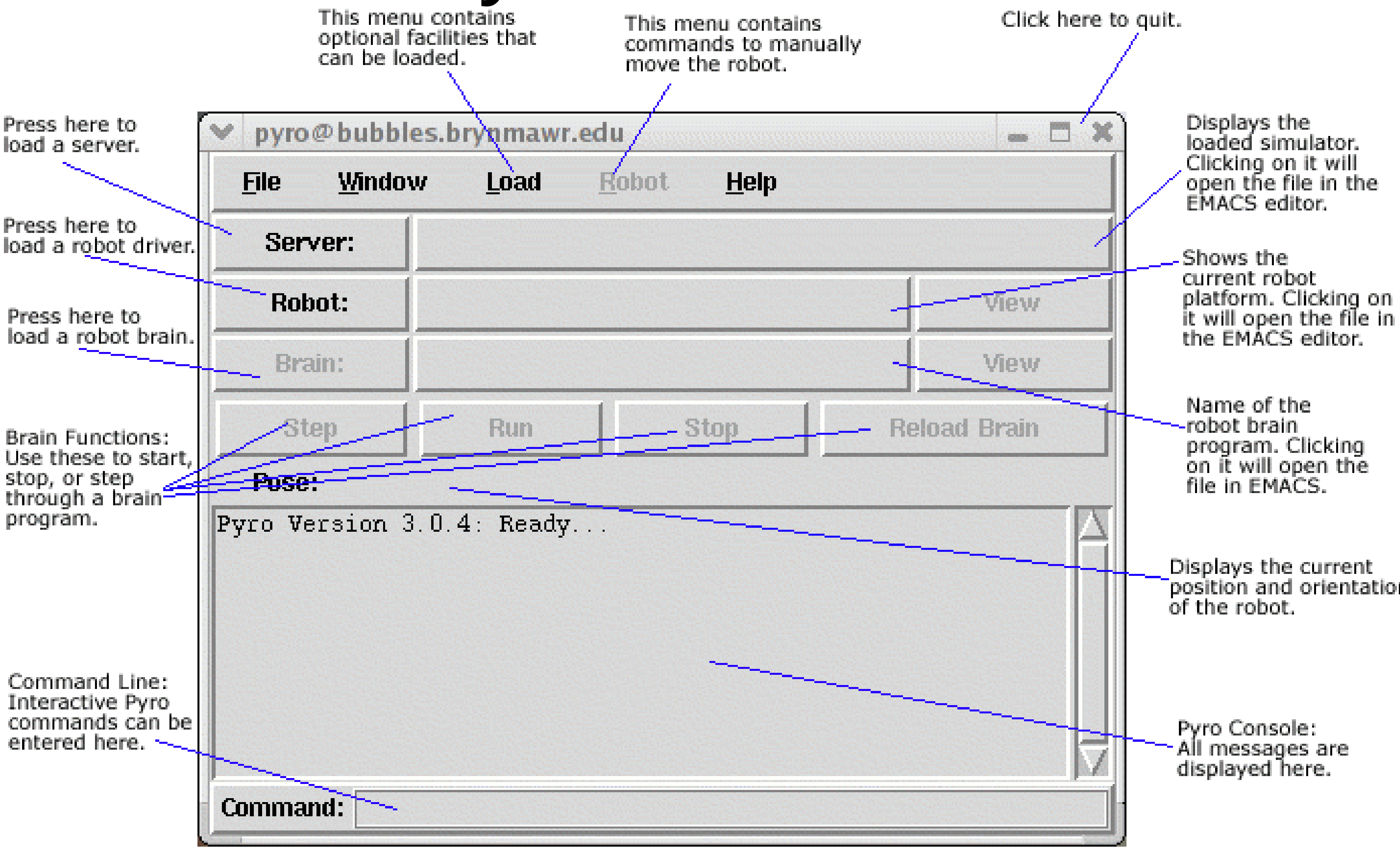
- Stage
  - Low-fidelity 2D simulator
  - Can simulate a large number of robots
- Gazebo
  - High-fidelity 3D simulator
  - Simulates Physics; displays with OpenGL
- RoboCup
  - Simulates RoboCup Soccer
- Pyrobot
  - Discrete action/sensor simulator
  - Continuous, with light sensors
  - Written in Python



Portable "Brains"

- Goals:
  - Create high-level abstractions so that controllers would work on a wide range of robots
  - Retain ability to take advantage of unique abilities of a particular type of robot
  - Develop a standard interface for interacting with robot and peripherals

Pyro Interface



Pyro LiveCD

- Boots on i386 (Intel-based) computers
- Turns your laptop into a Linux computer
- Based on KNOPPIX
- Will not write on your hard drive
- Contains Pyro, Player, Stage, Robocup Soccer Server, Pyrobot simulator, and all examples

Pyro Python Robotics

Pyro Abstractions

- Default range sensor: robot.range
  - Can be IR, sonar, laser, etc.
- Default range units are "ROBOTS": robot.range.units
  - 1 "robot" is the length of the robot being used
- Named sensor groups: robot.range["left-front"]
- Generalized motion control: robot.move(t, r)
- Abstract devices: robot.gripper[0].open()
  - Used to control devices, sensors, or visualizations

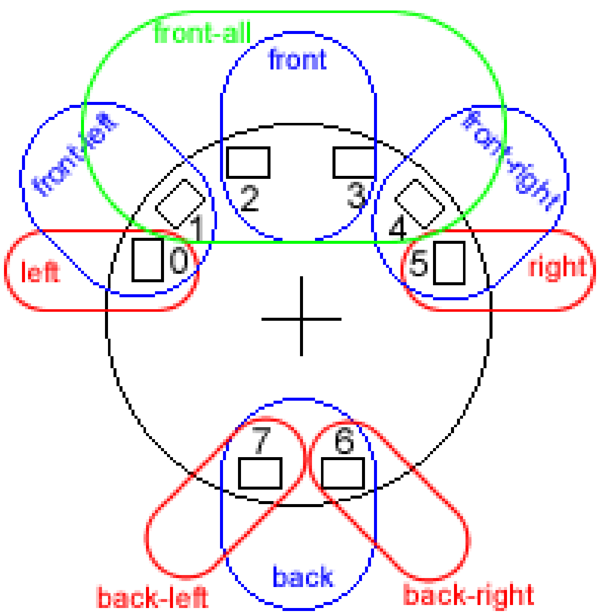
Default Range Sensor

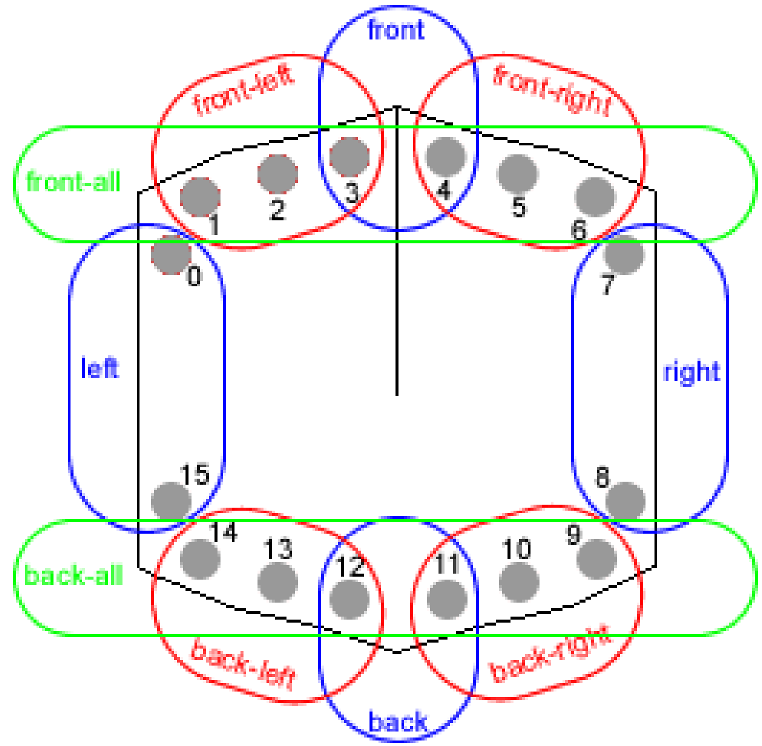
- robot.range is an alias, maybe to:
  - robot.sonar[0]
  - robot.sonar[1]
  - robot.laser[0]
  - robot.laser[1]
- robot.setRangeSensor("laser", 1)

Pyro Range Units

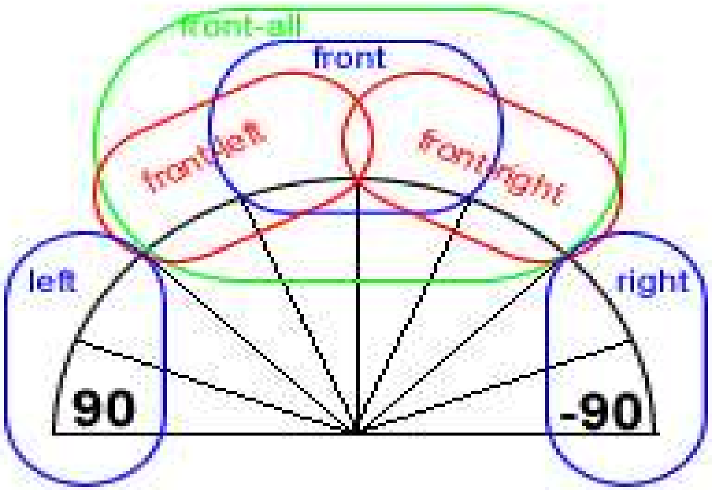
- Default is “ROBOTS” (relative to size)
- Other units include:
  - “SCALED” (0 to 1)
  - Metric (“CM”, “MM”, and “M”)
  - “RAW” (natural units of sensor)

Pyro Abstractions: Sensor Groups





Pyro Abstractions: Sensor Groups



Abstractions for Portability

- robot.range[“left-front”] might be three very different readings on different robots:
  - robot.laser[2][4]
  - robot.sonar[0][3, 5, 8]
  - robot.ir[1][4:7]
- and all range values could be relative

Generalized Motion

- translate(t): translation
- rotate(r): rotation
- move(t,r): translation and rotation
- motors(L, R): as if the robot had two motors
- stop(): stop all movement

All values are given between -1.0 and 1.0
relative to the robot's size.

Abstract Devices

- position: provides x,y,z and movement
- range: laser, sonar, IR distances in units
- light: provides value
- camera: from blobs, points, files, V4L, etc.
- gripper: open(), close(), lift(), etc.
- ptz: provides pan, tilt, zoom
- view: provides visualization

Robot Brains

Lisa Meeden

Python Robotics

Simple Pyro Brain

```
from pyrobot.brain import Brain

class Avoid(Brain):
   def wander(self, minSide):
      if min([s.value for s in self.robot.range[“left”]]) < minSide:
         self.move(0, -0.3)
      elif min([s.value for s in self.robot.range[“right”]]) < minSide:
         self.move(0, 0.3)
      else:
         self.move(0.5, 0)

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

def INIT(engine):
   return Avoid("myAvoid", engine)
```

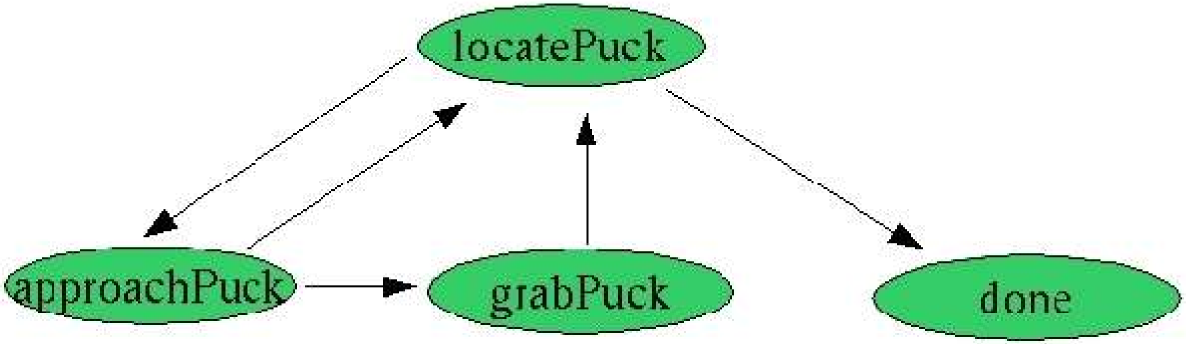
The Anatomy of a Brain

```
from pyrobot.brain.<SomeBrainClass> import *

class <BrainName>(<SomeBrainClass>):

   def setup(self):
       # This is the default constructor (optional) method
       # All code here is run once when the brain is loaded
       # You can initialize fields, and start devices here
   def step(self):
       # All brains must have a step method
       # This method is executed 10 times/sec
       # This is where you define the main control 'loop'
   def destroy(self):
       # This is the default destructor (optional) method
       # Each time a brain is destroyed, this method is executed
       # If you start devices in setup, you should stop them here

# Create a brain instance for the robot

def INIT(engine):
   brain = <BrainName>('BrainName', engine)
   print engine.robot.name + " robot now has " + brain.name + " brain."
   return brain
```

Wall-Following Brain

```
from pyrobot.brain import Brain

class WallFollow(Brain):
    # follows walls on its left, ignores sonar sensors on its right
    def wallFollow(self, dist):
        frontLeft = self.robot.sonar[0][0].value
        backLeft = self.robot.sonar[0][15].value
        front = min([s.value for s in self.robot.sonar[0][2:6]])
        if front < dist:
            print "wall in front"
            self.move(0,-0.5)
        elif (frontLeft < dist or backLeft < dist):
            print "following:",
            if frontLeft < backLeft:
                print "turn slight away"
                self.move(0.3,-0.1)
            else:
                print "turn slight toward"
                self.move(0.3,0.1)
        else:
            print "find wall"
            self.move(0.3,0)
    def step(self):
        self.wallFollow(1)

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

Collecting Pucks Brain



Example of FSM diagram for robot control to find and collect pucks

Anatomy of a Finite State Brain

```
from pyrobot.brain.behaviors import *
from time import *
class GatherPucksBrain(FSMBrain):
   def setup(self):
class locatePuck(State):
   def onActivate(self):
   def step(self):
       if SEE PUCK:
          self.goto('approachPuck')
       elif NO MORE PUCKS:
          self.goto('done')
class approachPuck(State):
   def update(self):
class grabPuck(State):
   def update(self):
class done(State):
   def step(self):
```

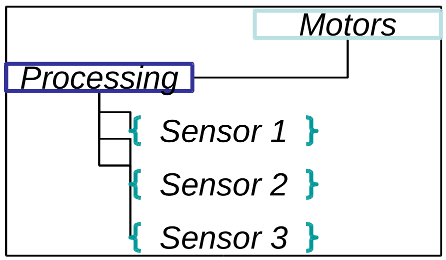
```
def INIT(engine=engine):
    brain = GatherPucksBrain(engine)
    brain.add(locatePuck(1))
    brain.add(approachPuck())
    brain.add(grabPuck())
    brain.add(done())
    return brain
```

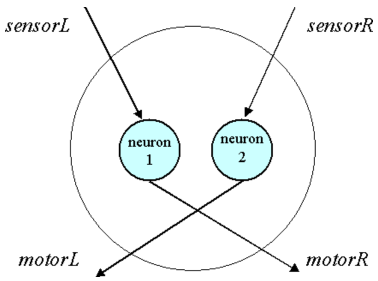
Pyro Modules Overview

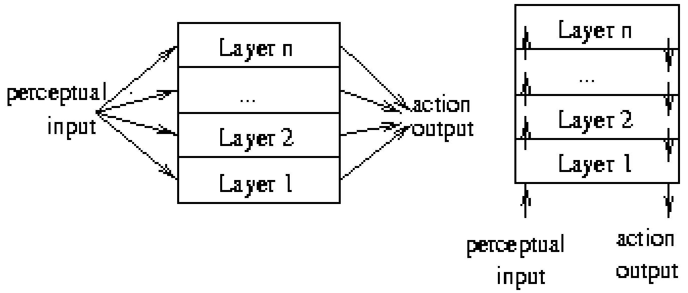
Holly Yanco

Overview of Pyro Modules

- Control Paradigms
  - Direct control
  - Reactive control
  - Behavior-based control
    - Subsumption
    - Fuzzy Logic
  - Finite State Machine



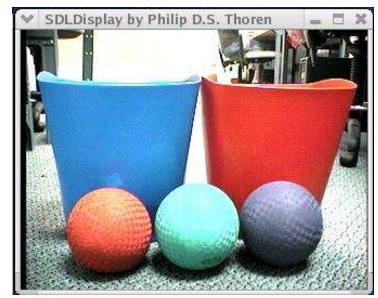


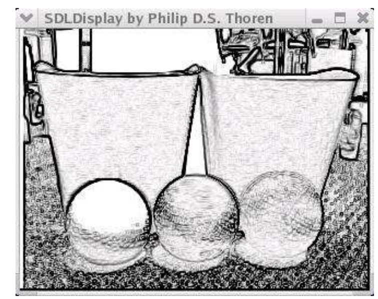


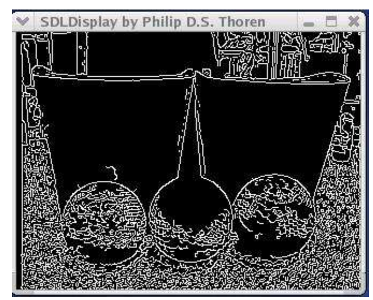
(a) Horizontal layering (b) Vertical layering

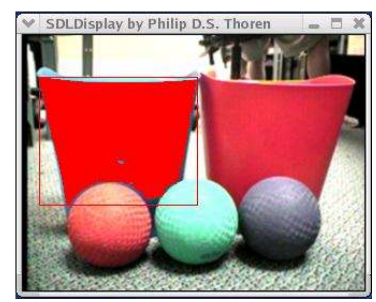
Overview of Pyro Modules

- Vision
  - Supports real and simulated cameras
  - Integrated with Pyro event loop
  - 10 frames per second



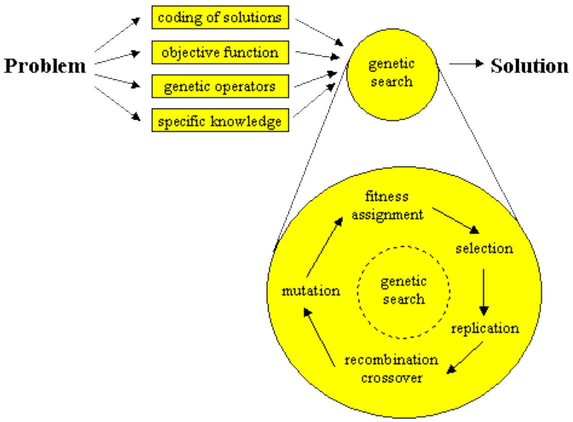


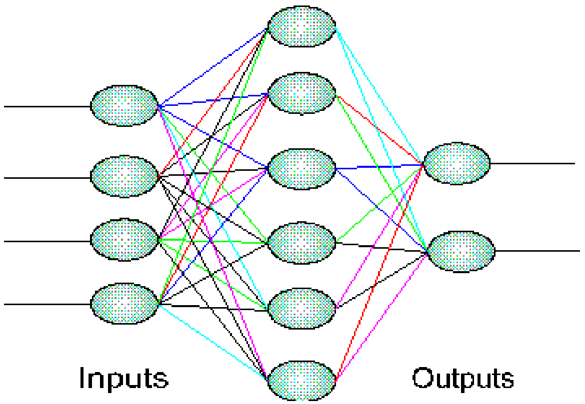




Overview of Pyro Modules

- Evolutionary Algorithms
  - Genetic Algorithms
  - Co-Evolutionary Methods
- Neural Networks
  - Flexible architecture
  - Back-propagation
  - Uses Pyro's "conx" neural network package

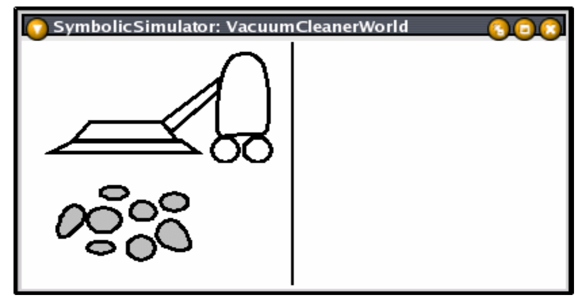


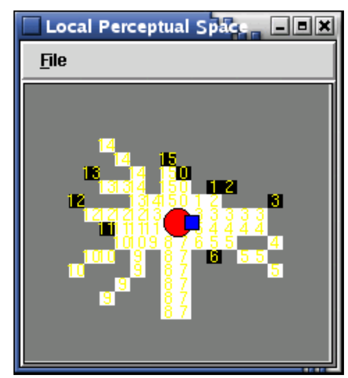


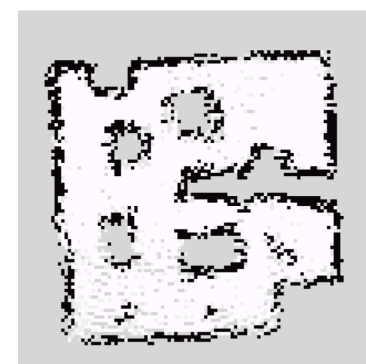
Python Robotics

Overview of Pyro Modules

- Symbolic Logic
  - Wumpus World
  - Konane Game
  - Vacuum Cleaner
- Bayesian Mapping
  - Real Time
  - Probabilistically based
  - Easy to add to any existing brain

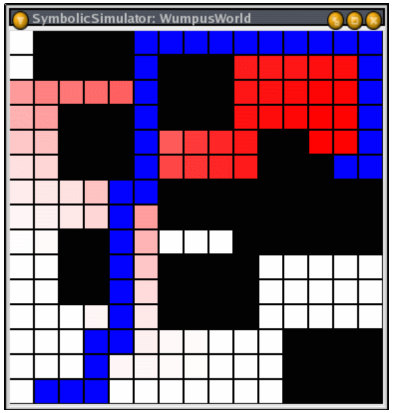


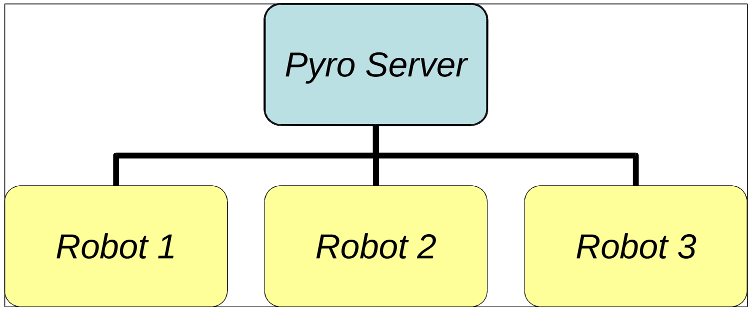




Overview of Pyro Modules

- Reinforcement learning
- Multi-robot Interaction
  - Heterogeneous groups of robots
  - Use for participation or competition





Pyro Module: Direct Control

- Direct reactive control of robot
- Sensor values used to determine current movement
- No blending of behaviors
- Simplest control method
- Usually the first module used to introduce a student to Pyro

Pyro Module:
Finite State Machine Control

- Can create states for controlling the robot, then transition from one state to the next
- Can use this finite state machine (FSM) control with direct control or more complicated behavior blending methods

Pyro Module: Behavior-Based Control

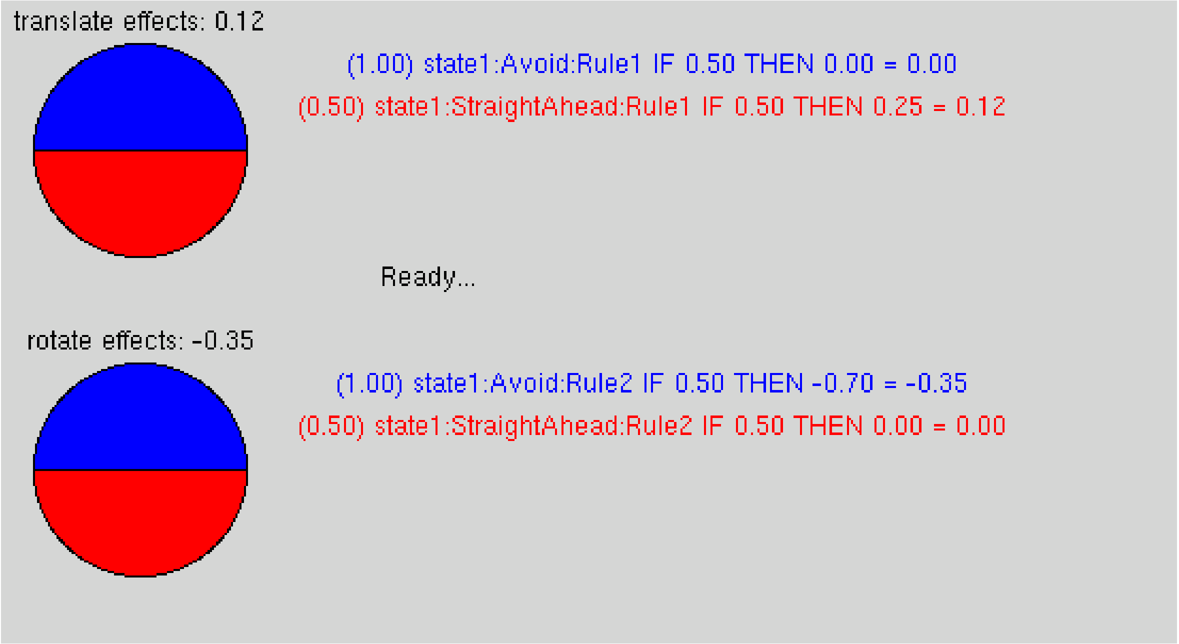
- Bottom-up control
- Many behaviors combined to produce action taken by robot
- Two ways in Pyro to blend behaviors
  - Subsumption
  - Fuzzy logic

Behavior-base Control: Fuzzy Control

- Can use fuzzy module to create variables with truth values that range from 0 (completely false) to 1 (completely true)
- Allows for smoother control of robot movement

Viewing Active Behaviors

- Can view the currently active brain behaviors
- Pie charts update in real time as different rules are triggered



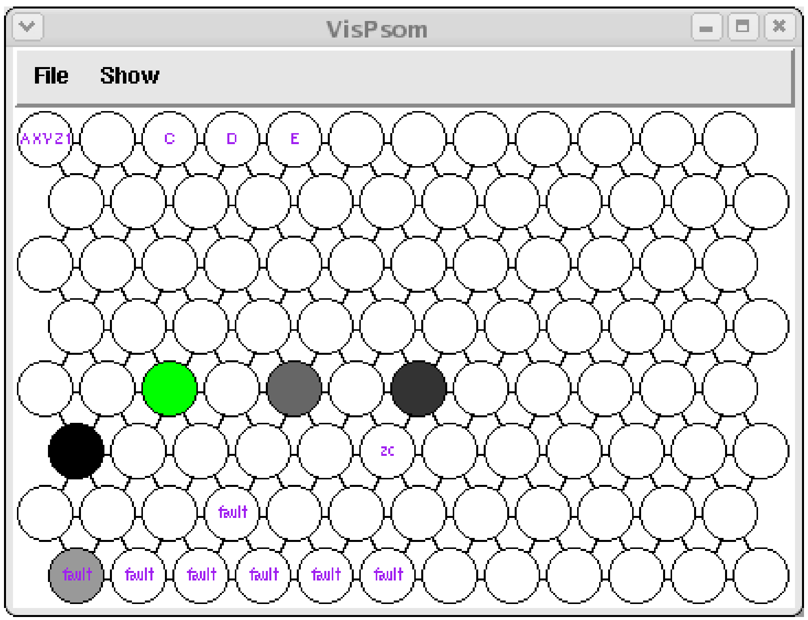
Pyro Module: Neural Networks

- Feed-forward Back-propagation of error simulator; requires a "teacher"
- Implemented in Python using Numerical Python extensions for matrix multiplications
- Implements Network, Layer, and Connection objects
- Reasonably fast, very flexible

Pyro Module: Self-Organizing Map

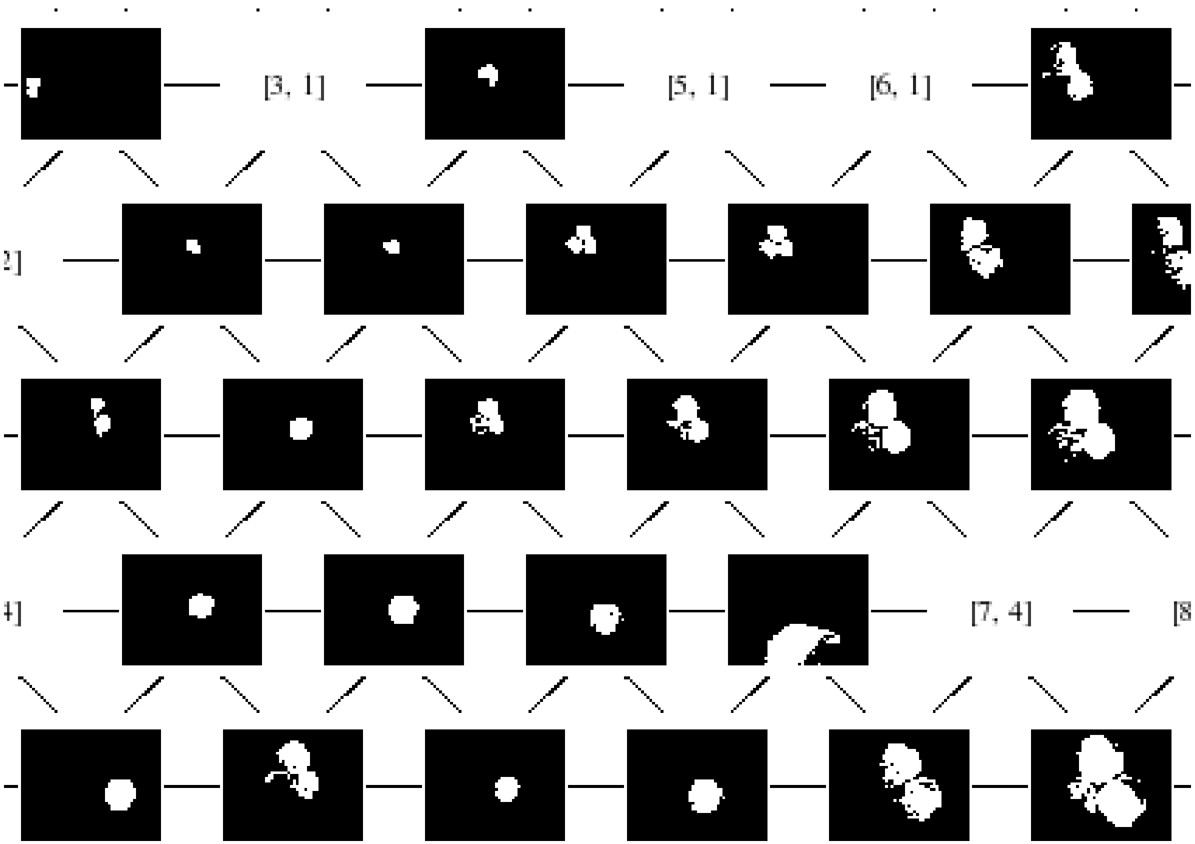
- Learns a 2D topology of discrete "categories" from multi-dimensional vectors
- Categories are actually "model vectors"
- Does not require a teacher or supervisor
- Like backpropagation training, makes small changes to a set of "weights"
- Implemented in C for speed

Self-Organizing Map



- C code is wrapped by SWIG
- Gives Python access to C-level functions
- Python + Tkinter provides graphical interface

Self-Organizing Map: Vision



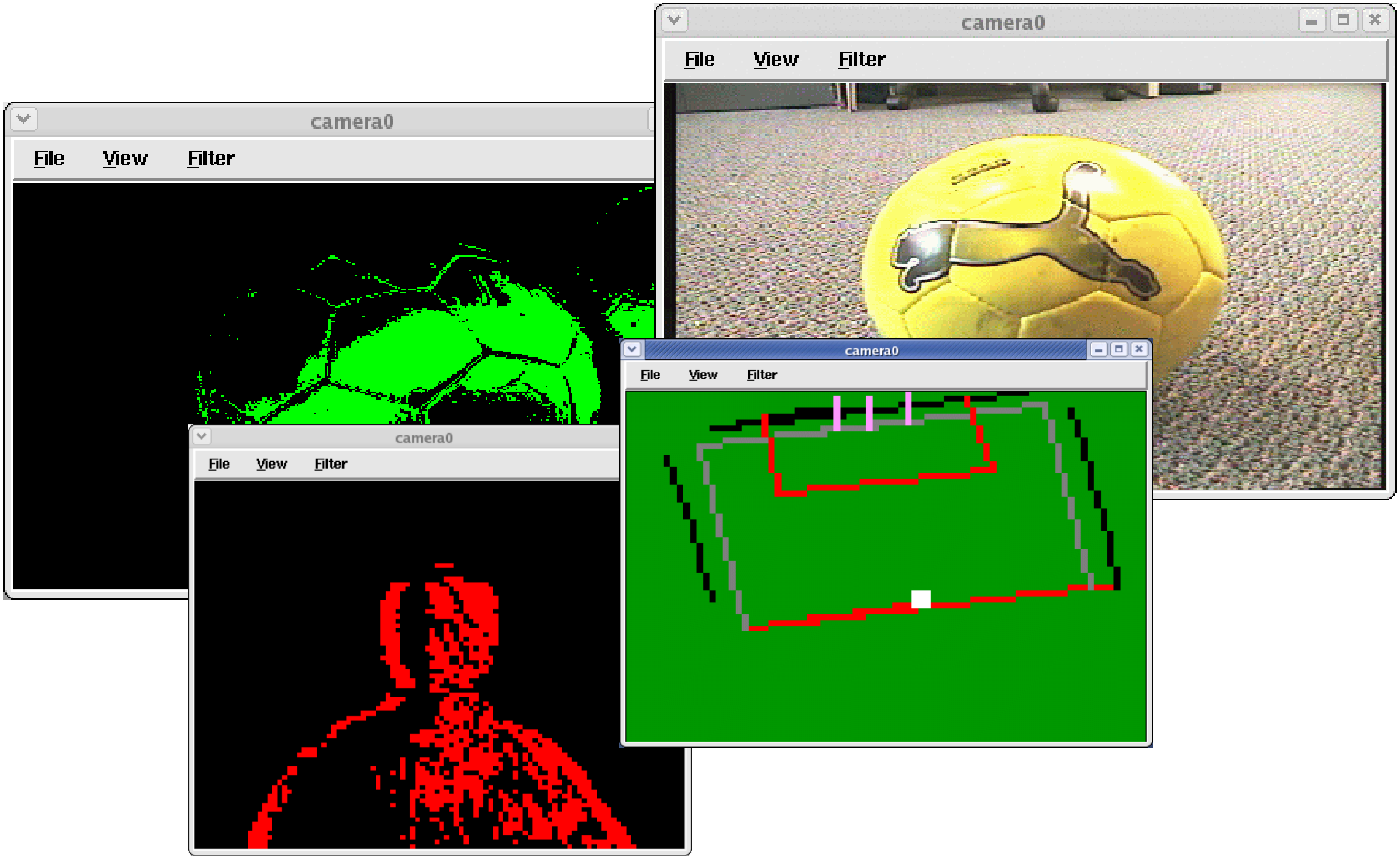
Pyro Module: Computer Vision

- Written in C++ for speed; wrapped with SWIG
- Simple “filter” abstractions for image processing
- Common gateway for all visual input:
  - Real cameras (Video for Linux)
  - Simulated vision from blobs (Stage) and points (Robocup soccer server)
  - Specialty interfaces, such as AIBO and file-based

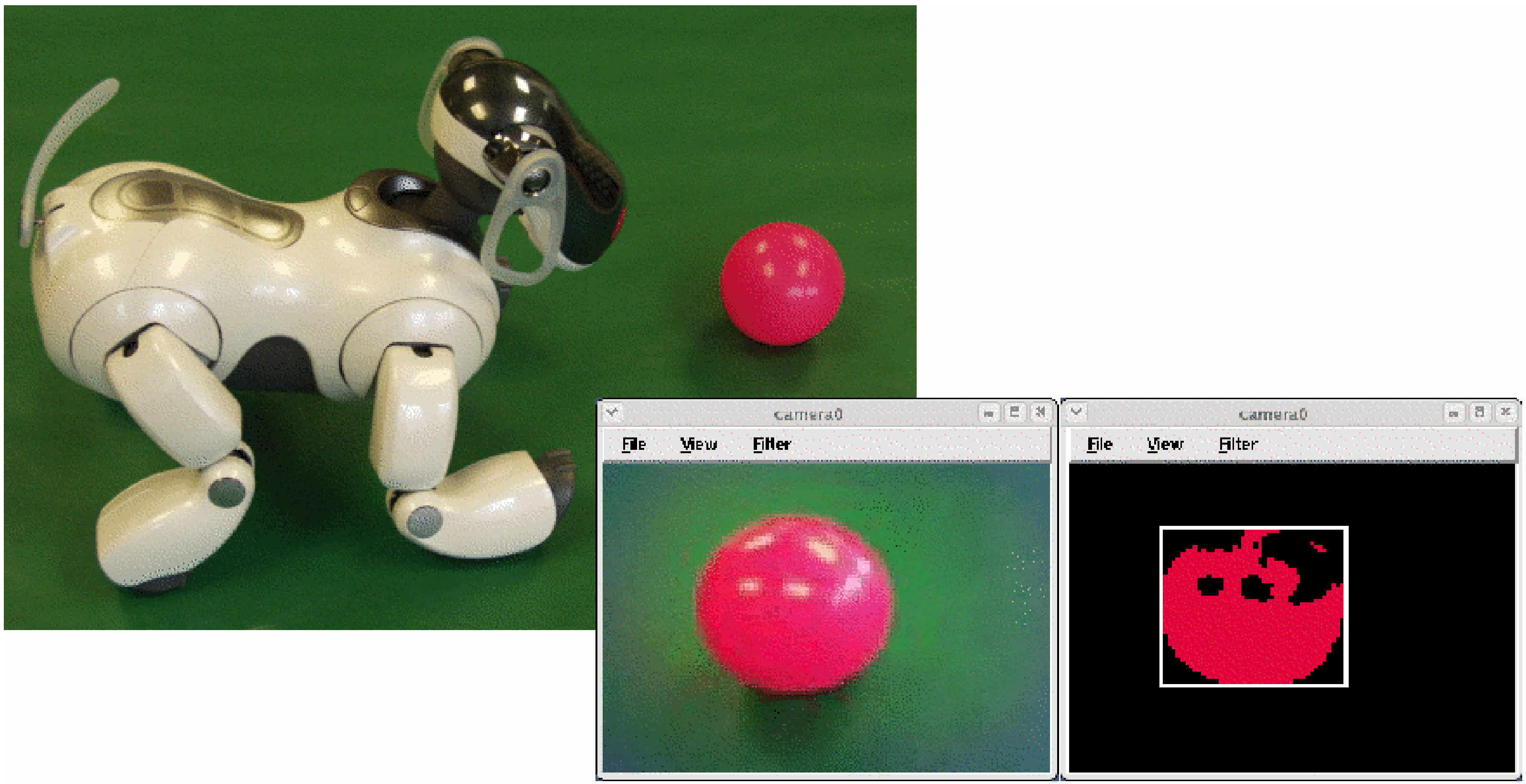
Pyro Computer Vision Filters

- Blur (mean, median, gaussian)
- Blobify
- Supercolor
- Threshold
- Edge Detection
- Motion Detection
- Pixel match (by range or tolerance)
- Gray scale
- Rotate
- Add noise
- Drawing functions
- Clear, copy, restore

Pyro Computer Vision



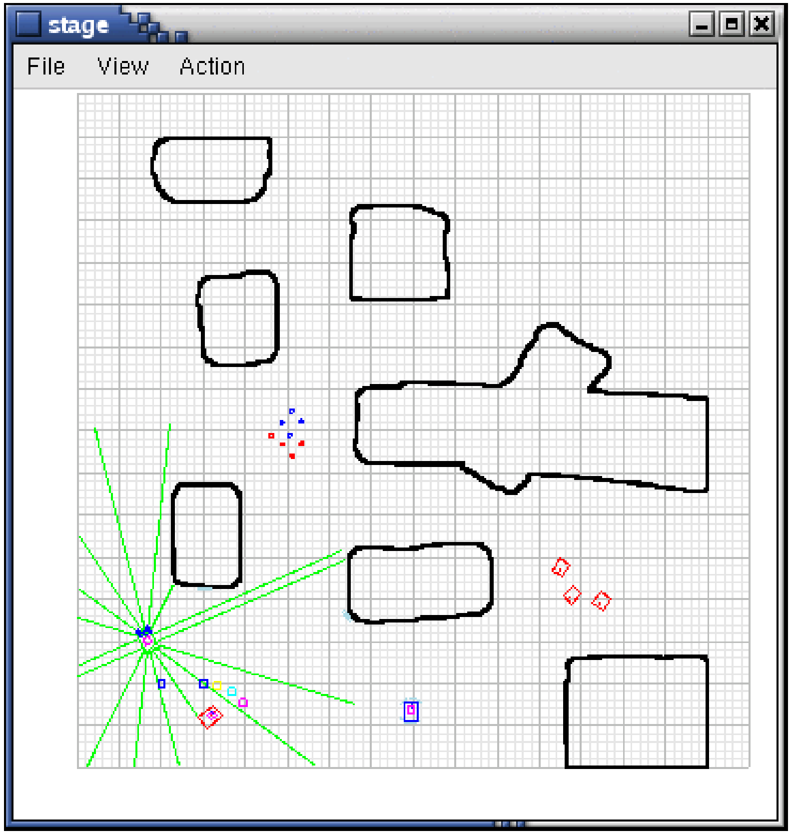
Pyro Computer Vision

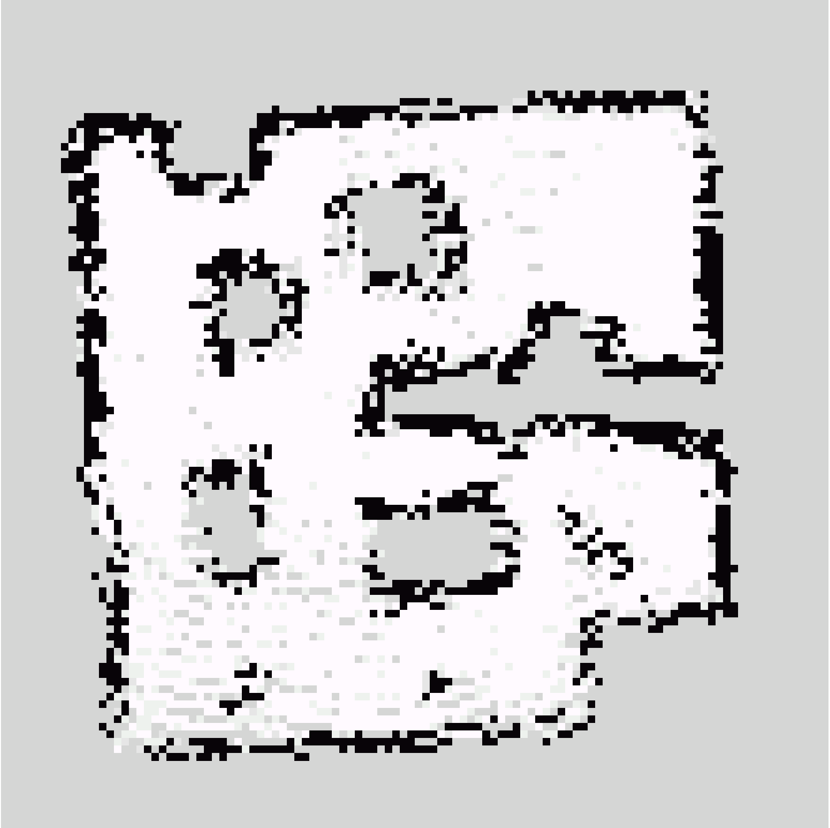


Pyro Module: Mapping

- Builds maps using occupancy grids
- Uses Bayesian updating
- Very accurate in simulator, much less so on a real robot
- Incorporating “ladar” on Pioneer into Pyro for more accurate mapping

Pyro Module: Mapping





Pyro Module: Evolutionary Algorithms

- Genetic algorithm (GA)
  array of bits, integers, floats, or strings
- Genetic programming (GP)
  trees of expressions

- Population of solutions
- Crossover, mutation, and selection

Python Robotics

GA + NN = Evolvable Robot

Combining the Genetic Algorithm with the Neural Network creates an easy to evolve robot controller

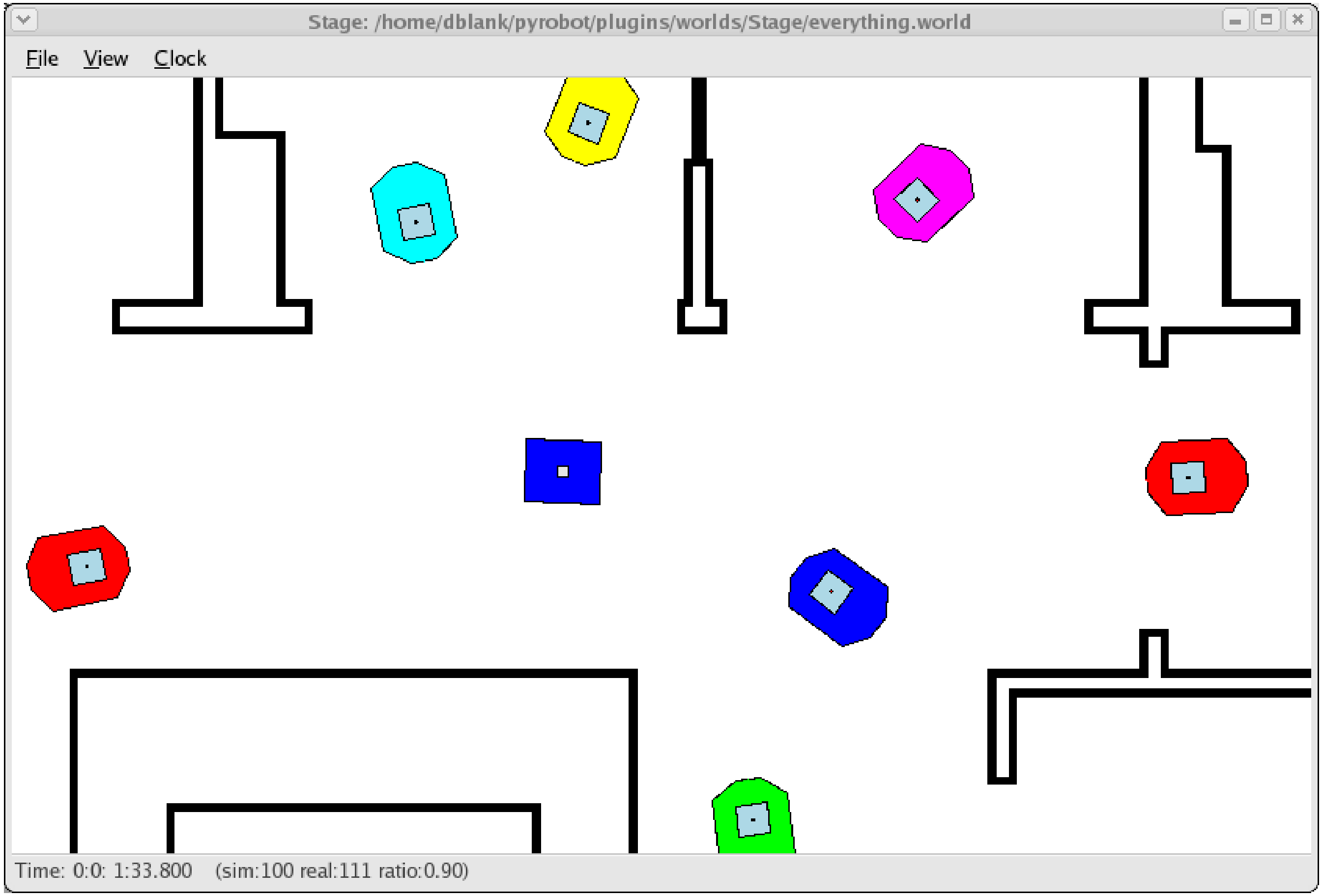
1) Evolve a list of floating point numbers
2) Load as weights in a neural network controller
3) Let robot run for a while; score performance
4) Performance is fitness for that “gene”

Python Robotics

Pyro Module: Multirobot

- Team coordination (Robocup soccer)
- Swarm behaviors (hundreds of robots, Stage simulator)
- Competitions (tag, hide-and-seek, etc.)
- Task decomposition (Gazebo simulator)
- Mapping (search and rescue)
- Communication issues (real robots)

Multirobot Tasks



Multirobot Tasks