2009

The Pyro toolkit for AI and robotics

Doug Blank
Bryn Mawr College, dblank@brynmawr.edu

Deepak Kumar
Bryn Mawr College, dkumar@brynmawr.edu

Lisa Meeden

Holly Yanco

The Pyro toolkit for AI and robotics

Douglas Blank
Computer Science
Bryn Mawr College
Bryn Mawr, PA 19010
dblank@cs.brynmawr.edu

Deepak Kumar
Computer Science
Bryn Mawr College
Bryn Mawr, PA 19010
dkumar@cs.brynmawr.edu

Lisa Meeden
Computer Science
Swarthmore College
Swarthmore, PA 19081
meeden@cs.swarthmore.edu

Holly Yanco
Computer Science
Univ. of Mass. Lowell
Lowell, MA 01854
holly@cs.uml.edu

Abstract

This article introduces Pyro, an open-source python robotics toolkit for exploring topics in AI and robotics. We present key abstractions that allow Pyro controllers to run unchanged on a variety of real and simulated robots. We demonstrate Pyro's use in a set of curricular modules. We then describe how Pyro can provide a smooth transition for the student from symbolic agents to real-world robots, which significantly reduces the cost of learning to use robots. Finally we show how Pyro has been successfully integrated into existing AI and robotics courses.

Keywords: AI education, robotics, python, integrated programming environment, toolkit

Introduction

In this article we present Pyro, an open-source, python-based programming environment for exploring robotics and artificial intelligence. Pyro, which stands for *Py*thon *Ro*botics, enables users to easily write sophisticated AI programs in python to control a variety of robots and agents. Pyro provides a high-level interface to robots, relieving the user from low-level, robot-specific details. Further, robot programs written in Pyro can be used to control several different kinds of robots without any modifications. Pyro has already been successfully used in a number of undergraduate and graduate-level AI courses at several different institutions. In this article, we will introduce Pyro as a programming environment for teaching robotics and AI. To find out more about the underlying design principles and evolution of Pyro see Blank *et al.* (2005).

One of the main goals of Pyro is to reduce the cost of learning to program robots and AI agents. The last decade has seen a proliferation of mobile robot platforms that has led to their introduction in undergraduate and graduate-level AI curricula. However, each robot comes with its own, often proprietary, programming environment or API. Thus, the cost of learning to program robots includes the overhead of learning the specific robot's programming paradigm, and, in many cases, the programming environment. Despite the trend towards low-cost robot platforms, this overhead serves as a barrier against the pedagogical aims of learning to build AI-based robot agents. Pyro solves this problem by introducing generic robot abstractions that are uniform across a number of robot platforms (real and simulated) regardless of their size or morphology. This significantly reduces the cost of learning to program robots and makes robotics more accessible to students. Pyro's abstractions, much like the abstractions provided in high-level programming languages, provide a robot-independent programming interface so that programs, once written in Pyro, can control several different kinds of robots using the same code. The current version of Pyro supports the Khepera robot (Mondada, Franzi, & Ienne 1993), the Pioneer robot (ActivMedia 2003), the Sony AIBO robot (Sony 2005), and dozens of other robots in simulation.

All Pyro programs are written in the python programming language. Python is a relatively new programming language that is quite powerful and embodies several modern programming paradigms. Yet it is an easy programming language to learn for students and instructors alike. One of the reasons Pyro was developed in python was to take advantage of the support available in the language for the re-use of existing code. This enables easy integration of existing robot APIs, as well as existing libraries of AI code. For example, any code written in C/C++ can be used from within python code. We have taken advantage of this feature to integrate several existing robot APIs as well as existing APIs for AI modeling (such as self-organizing maps, and tools for image processing).

Pyro comes integrated with several existing robot simulators (including Robocup Soccer (Group 2005), Aria (ActivMedia 2003), Player/Stage (Gerkey, Vaughan, & Howard 2003), and Gazebo, a newer 3D simulator (Koenig & Howard 2004)). Figure 1 shows Pyro running a wander program on a Pioneer robot in the Gazebo simulator. Schools that do not currently own robot platforms can still make use of Pyro by introducing robot programming in their courses through the use of the integrated simulators. Even when an institution owns robots, simulators can be used by students to effectively test and debug programs before they are run on actual robots.

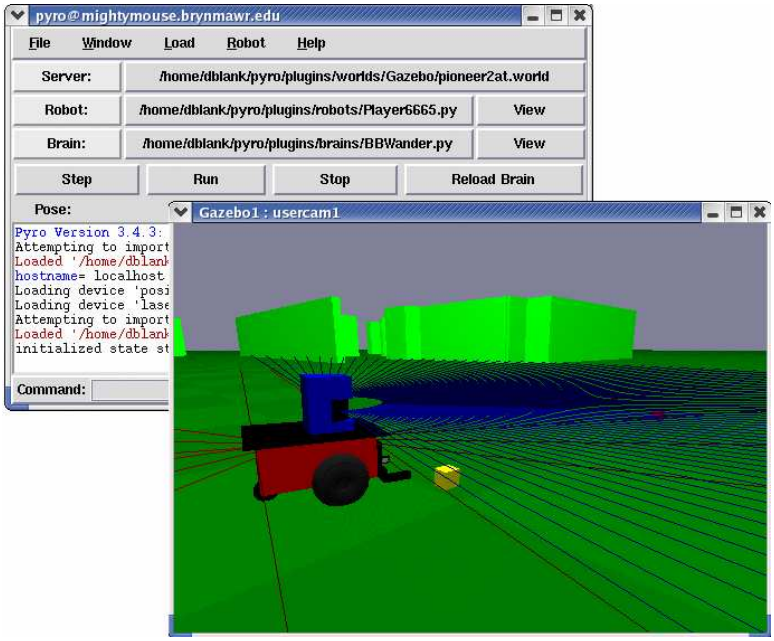

Figure 1: A view of Pyro controlling a Pioneer robot in the Gazebo simulator.

We have developed extensive materials that can be used by instructors to teach robot programming to students. The materials include beginner's tutorials, examples of robot programming paradigms, and several AI modules (such as neural networks and evolutionary computation) that can also be used for doing advanced research in AI. We are continuously adding more modules. Plans are already under way to integrate the AI modules available in python from Russell and Norvig's AI text (Russell & Norvig 2002).

Pyro is an open source project. We are committed to the inclusion of contributed materials and code that enhances the functionality of Pyro. We are also committed to adding support for more robot platforms as the robots and their APIs become available. Currently, work is under way to integrate support for the low-cost Hemisson robot (KTeam 2005), and also the ER1 robot (Robotics 2005) platforms.

In what follows, we provide a quick first look at writing robot programs in Pyro. This is followed by an overview of the curricular materials currently available and a few more examples. Next we show how Pyro can also be used to span topics in traditional AI to those in robotics, and describe how Pyro has been integrated into various courses at different institutions. We will conclude by sketching possible future directions for Pyro.

A Pyro example

In this section we present a simple wall-following program to demonstrate the unified framework that Pyro provides for using the same control program across many different robot platforms. This type of basic controller is an example of reactive control where the robot maintains very limited state information and primarily determines its actions based on its current sensor readings. This form of control is normally the first control method introduced to students learning robotics.

The program shown in Figure 2 is written in an object-oriented style, and creates a class called `FollowBrain` that inherits from a Pyro class called `Brain` (Figure 2, line 2). A Pyro brain is required to have a `step` method (line 5) that implements the decision procedure and is executed on every control cycle, which occur about 10 times a second. A Pyro brain may also have a `setup` method (line 3) that is called when the brain is instantiated and can be used to initialize class variables.

The brain shown in Figure 2 always tries to follow walls on its left side. On each control step, it first queries the range sensors on its front and left side (lines 6–9). If the front sensors indicate that the robot is approaching something, then the robot turns right so as to align its left side with the wall. Once it senses that its left sensors are close enough to the wall, then it sets its class variable `self.follow` to be true and goes straight. When `self.follow` is true, the robot makes small adjustments to try stay aligned with the wall, which are based on readings from its front-left and back-left range sensors. Otherwise, if the robot is not sensing a wall on its left, it sets its class variable to false and goes straight until it encounters a wall.

It is not crucial to understand all of the details of this Pyro program; however, it is important it recognize how Pyro's abstractions are used to create a platform-independent implementation. One of the key ideas un-

Figure 3: Two very different robot platforms, the tiny Khepera and the much larger Pioneer, for which the same Pyro program from Figure 2 can be used for wall following.

```
1  from pyro.brain import Brain
2  class FollowBrain(Brain):
3     def setup(self):
4        self.follow = 0
5     def step(self):
6        f = self.robot.range.values('front')
7        fl= self.robot.range.values('front-left')
8        bl= self.robot.range.values('back-left')
9        l = self.robot.range.values('left')
10       if(min(f) < 0.5):
11          print "wall ahead, turn right"
12          self.robot.move(0, -0.2)
13       elif(self.follow and min(fl) < 0.55):
14           print "following, adjust right"
15           self.robot.move(0.2, -0.05)
16       elif(self.follow and min(bl) < 0.55):
17           print "following, adjust left"
18           self.robot.move(0.2, 0.05)
19       elif(min(l) < 0.9):
20           print "following"
21           self.follow = 1
22           self.robot.move(0.2, 0)
23       else:
24           print "looking for wall"
25           self.follow = 0
26           self.robot.move(0.5, 0.0)
27 def INIT(engine):
28    return FollowBrain('FollowBrain', engine)
```

Figure 2: A platform-independent wall-following program in Pyro.

derlying the design of Pyro is the use of abstractions that make the writing of basic robot behaviors independent of the type, size, weight, and shape of a robot. Consider writing a robot controller for wall-following that would work on both a fifty pound, twenty-four inch diameter Pioneer robot with sonar sensors and on a three ounce, two inch diameter Khepera robot with IR sensors. Figure 3 illustrates the vast difference in size between the Khepera and the Pioneer robots. The following key abstractions were essential in achieving this.

Range Sensors: Regardless of the kind of hardware used, IR, sonar, or laser, these sensors are categorized as *range* sensors. Sensors that provide range information can thus be abstracted and used in a control program.

Robot Units: Distance information provided by range sensors varies depending on the kind of sensors used. Some sensors provide specific range information, like distance to an obstacle in meters or millimeters. Others simply provide a numeric value where larger values correspond to open space and smaller values imply nearby obstacles. In our abstractions, in addition to the default units provided by the sensors, we have introduced a new measure, *a robot unit*: 1 robot unit is equivalent to the diameter of the robot being controlled.

Sensor Groups: Robot morphologies vary from robot to robot. This also affects the way sensors, especially range sensors, are placed on a robot's body. Additionally, the number and positions of sensors present also varies from platform to platform. For example, a Pioneer3 has 16 sonar range sensors while a Khepera has 8 IR range sensors. In order to relieve a programmer from the burden of keeping track of the num-

ber and positions of sensors and their unique numbering scheme, we have created *sensor groups*: `front`, `left`, `front-left`, etc. Thus, a programmer can simply query a robot to report its front-left sensors in robot units. The values reported will work effectively on any robot, of any size, with any kind of range sensor given appropriate coverage, yet will be scaled to the specific robot being used.

Motion Control: Regardless of the kind of drive mechanism available on a robot, from a programmer's perspective, a robot should be able to move forward, backward, turn, and/or perform a combination of these motions. We have created the motion control abstraction: *move(translate, rotate)* where movements are given in terms of turning and forward/backward changes. This is designed to work even when a robot has a different wheel organization or four legs (as with the AIBO). As in the case of range sensor abstractions, the values given to this command are independent of the specific values expected by the actual motor drivers. A programmer only specifies values in a range [-1.0,1.0].

The wall-following program in Figure 2 illustrates the use of all of the above abstractions. Each case of the `if` statement (starting on line 10) queries the robot's range sensors based on a specific sensor group and checks for range values in terms of robot units. For example, the robot will respond to obstacles in the front when they are within half a robot unit and the robot is considered to be following a wall when it is within 0.9 robot units on its left side. In addition, each case of the `if` statement uses the abstract `move` command to control the robot's next action.

This first glimpse of Pyro demonstrates how Pyro's abstractions allow students to focus on the robot's behavior and relieves them from having to understand the low-level details of the robot's morphology and control mechanisms. Even the very simple wall-following program of Figure 2 offers an immediate opportunity to connect to broader topics in AI, such as using machine learning techniques to learn appropriate parameter settings for the control parameters. The next section expands on such opportunities by giving an overview of the curricular modules available within Pyro and provides several more examples of Pyro's capabilities.

Curricular materials

The Pyro library includes several modules that enable the exploration of robot control paradigms, robot learning, robot vision, localization and mapping, and multi-agent robotics. Within robot control paradigms there are several modules: sequential control using finite state machines, subsumption architecture, and fuzzy logic control. The learning modules provide an extensive coverage of various kinds of artificial neural networks: feed-forward networks, recurrent networks, self-organizing maps, etc. Additionally we also have modules for evolutionary systems, including genetic algorithms, and genetic programming. The vision modules provide a library of the most commonly used filters and vision algorithms enabling students to concentrate on the uses of vision in robot control. The entire library is open source, and can be used by students to learn about the implementations of all the modules themselves. We have also provided tutorial-level educational materials for all of the modules. Similar to the software's open source license, these modules are available under a Creative Commons license. This enables instructors to tailor the use of Pyro for many different curricular situations. In the remainder of this section, we provide two more examples of Pyro programs written using the available libraries.

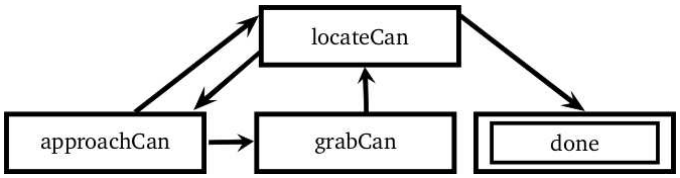

Figure 4: A graph of behaviors for implementing a recycling robot. The robot begins in the `locateCan` state and ends in the `done` state.

Example of sequential control

In order to create more complex robot controllers, it is useful to be able to group low-level robot commands into logical units, typically called behaviors. There are a number of robotics textbooks that focus on this style of control, known as the behavior-based approach (Arkin 1998; Murphy 2000). In this approach, each behavior is triggered by a particular condition in the environment, and responds appropriately. Once the initiating condition has been addressed, the current behavior can pass control off to another behavior. One straightforward method of implementing this style of behavior-based control is through finite state machines (FSMs). Each state in the FSM represents a robot behavior. Using a FSM the designer can build up a graph of states and designate appropriate sequences of control between states. In a sense, the FSM represents a "plan" for both accomplishing higher-level tasks through the compositions of lower-level primitives and for reacting to unpredictable situations.

To illustrate this style of robot control, we can implement a simplified version of a recycling robot. We demonstrate this using a simulated Pioneer robot with a gripper and a "blob" camera (discussed below). The cans are represented as randomly positioned red pucks in a circular environment without obstacles. The robot's goal is to collect all of the red cans. Once the robot has picked up a can, it immediately stores it, and moves on to finding more cans.

Figure 4 shows one way of decomposing this problem into a set of four behaviors: `locateCan`, `approachCan`, `grabCan`, and `done`. The FSM begins in the state `locateCan`. While in this state the robot rotates, look-

```
class locateCan(State):
   def onActivate(self):
      #initializes a class variable to count rotations
      self.searches=0
   def step(self):
      #get a list of all blobs:
      blobs=self.robot.camera[0].filterResults[1]
      #checks if there are any blobs
      if len(blobs)!=0:
         #stops robot when a blob is seen
         self.robot.move(0, 0)
         print "found a can!"
         #transfers control to homing behavior:
         self.goto('approachCan')
      #checks if robot has done a complete rotation
      elif self.searches > 275:
         print "found all cans"
         #transfers control to completion behavior:
         self.goto('done')
      #otherwise keep rotating and searching
      else:
         print "searching for a can"
         #updates rotation counter:
         self.searches+=1
         #rotates robot and remains in locate behavior:
         self.robot.move(0, 0.2)
```

Figure 5: The implementation of the `locateCan` behavior in a FSM-style Pyro program. All of the behaviors of the FSM are represented as instances of the `State` class. The filters defining the blobs for identifying the red cans are set in the brain constructor which isn't shown here.

ing for a blob, which would indicate that a red can is in sight. As soon as a can is found, the FSM goes into state `approachCan` to move the robot toward the closest visible can. If for some reason the robot loses sight of the can, the FSM will go back to the state `locateCan`. Once the robot is positioned with its gripper around a can, the FSM goes to the state `grabCan` to cause the robot to pick it up and store it. Then the FSM will return to the state `locateCan` to search again. The state `locateCan` keeps track of how long it searches on each activation of the state. If the robot has done a complete rotation and not seen any cans, the FSM goes to the state `done` and stops the robot.

Figure 5 shows the definition of only one of the four states that make up the complete recycling robot's FSM brain: the `locateCan` state. Each state in a FSM must implement the `step` method, which is called on every control cycle. States use the `goto` method to transition to other states. The optional `onActivate` method may be used to initialize class variables.

Figure 6 shows the robot as it passes through various states during the execution of its FSM brain. First it begins searching for cans (A), then it closes in on a particular can (B), grabs it (C), and starts pursuing a new can (D). This style of sequential control is a very effective method of implementing complex robot behaviors.

Example of vision processing

To explore topics in computer vision, Pyro also comes with camera and image processing modules. Students can write python programs to implement vision algorithms, such as color histograms, motion detection, object tracking, edge detection, etc. However, python is currently too slow for this code to be used in real time. To alleviate this problem, we have developed a method such that the low-level vision code is written in C++ but the students can interactively use this code to build layers of *filters* in Pyro. Thus, students can develop the computationally expensive code in C++, and still have the high-level, interactive interface of python. For example, in the background of Figure 7, a Sony AIBO robot is looking at a ball. The foreground of Figure 7 shows the raw image on the left before the application of any filters, and an image-processed view on the right after having a series of filters applied to it. Pyro applies all filters to a copy of the current image.

In this example, the filters were: color matching, supercolor, and blob segmentation. The color matching filter marks all pixels in an image that are within a threshold of a given red/green/blue color triplet. The supercolor filter magnifies the differences between a given color and the others. For example, the *supercolor red* filter makes reddish pixels more red, and the others more black. Finally, the blob segmentation filter connects adjacent like-colored pixels into regions, computes a box completely surrounding the matching pixels, and returns a list of these bounding boxes. All of these filters can be sequenced and applied without students having to engage in the implementation details of the low-level image-processing routines. Figure 7 shows the ball as the largest matching region by drawing a bounding box around it (foreground, right). Once the position of the bounding box is known, the robot can then be programmed to look or move toward the ball.

Integrating Pyro into the curriculum: From agents to robots

Teaching artificial intelligence as a coherent subject can be a challenging task. AI is already filled with a wide spectrum of ideas and methodologies that run the gamut from logic to evolution, from information theory to perception. Surely, the idea of incorporating even a bit of robotics would cause an AI course to explode, spewing predicates, symbols, and rules in all directions, right? We think that including robotics in the standard AI course, if done appropriately, can actually help bridge otherwise disparate facets of the field. We recognize that not everyone thinks that robots in the AI classroom is a good idea. Marvin Minsky recently was quoted as being appalled at the amount of time that students were wasting on "soldering and repairing" such "stupid little robots" (*Wired News*, May 13, 2003). Although we feel his criticisms were largely misguided, we also can appreciate appropriate efficacy in the classroom. Here, we suggest the use of pre-built,

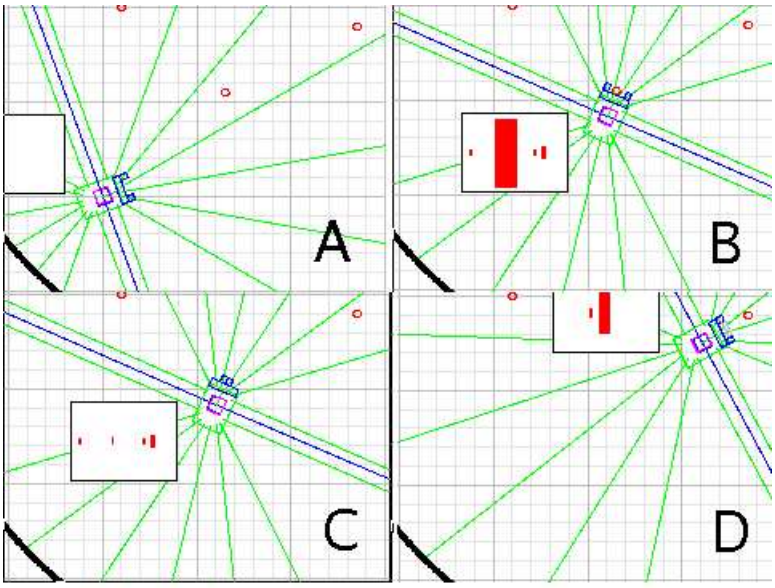

Figure 6: A Pioneer robot in the Player/Stage simulator controlled by the finite state machine Pyro brain for recycling. The small box to the left of the robot represents the blob camera data. Each rectangle in this box represents a red blob. The larger the rectangle, the closer the blob. The four sub-pictures depict various behaviors within the FSM. A: At the start of `locateCan`. B: At the moment when `approachCan` passes control to `grabCan`. C: Just after the successful completion of `grabCan`. D: As the robot homes in on another can while in state`approachCan`.

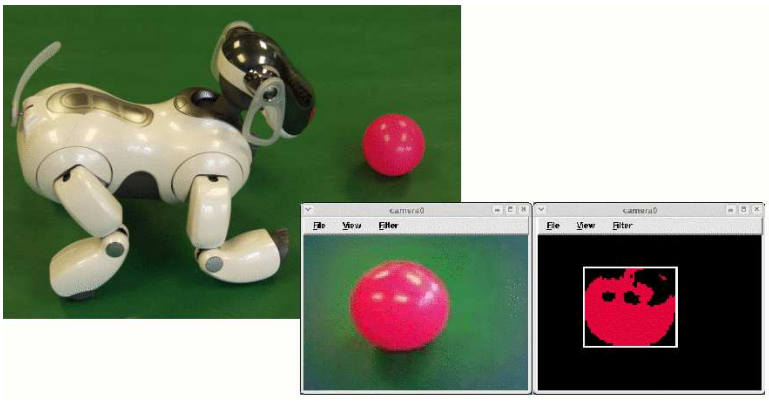

Figure 7: In the background, an AIBO robot is shown under the control of Pyro over a wireless network connection. In the foreground, the robot's raw image (left) is shown next to a processed image (right).

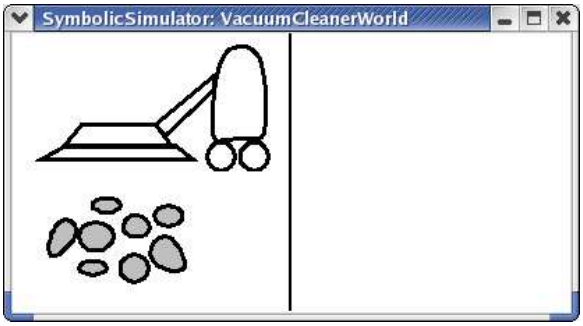

Figure 8: Simple Vacuum Cleaner World (after Russell and Norvig, 2003). This simulation, itself, was written in less than 100 lines of python code, including the graphics, thread, and socket code. Although we have other AI-based simulations, such as Konane (a Hawaiian checkers game), students can also write their own simulations and games fairly easily.

commercial robots and simulators.

In addition to having access to affordable robots, another trend over the last decade also helped make robotics a viable topic in AI. In 1995, Russell and Norvig published their first edition of *AI: A Modern Approach* which used an agent-based perspective for exploring all of artificial intelligence. This approach was, to many, a more effective technique of weaving together the disparate topics of AI than past attempts. This resulted in a successful textbook that has been adopted by many colleges and universities, and which has generated a second edition (Russell & Norvig 2002).

Approaching AI in the classroom from the perspective of an agent is a simple but effective methodology. A common approach is to introduce the ideas of the *agent* and its *environment*. Agents are in turn composed of *sensors* and *actuators*. The details of the sensors and actuators are usually downplayed, if not completely ignored, in an AI class. In robotics, of course, these are the core concepts. However, focusing on the sensors and actuators early in an AI class can bring to light important issues in AI. What happens if a sensor is not accurate? What happens if the world changes after a sensor is read? How does the robot know where it is? What happens if a robot doesn't move exactly the way it was supposed to?

Having such issues highlighted early in the semester can make it easier to talk about why one AI technique might be more appropriate than another for a given problem. Of course, having an implementation of an agent-based algorithm can help students by providing a concrete example with which to make these issues more salient. It will also allow them to transition from the symbolic domains of agents to the real-world domains of robots.

Consider the vacuum world shown in Figure 8 and the simple reflex agent controller shown in Figure 9. The algorithm describes a robot vacuuming cleaner that can

function REFLEX-VACUUM-AGENT([*location, status*])
returns an action

  if *status* = *Dirty* **then return** *Suck*
  else if *location* = *A* **then return** *Right*
  else if *location* = *B* **then return** *Left*

Figure 9: Russell and Norvig's reflex vacuum agent (Russell & Norvig 2002) figure 2.8 page 46.

```
from pyro.brain import Brain
class SimpleBrain(Brain):
   def ReflexVacuumAgent(self,location,status):
      if status == "dirty": return "suck"
      elif location == "A": return "right"
      elif location == "B": return "left"
   def step(self):
      # ask the robot for perceptions:
      location = self.robot.location
      status = self.robot.status
      # call the agent program
      act=self.ReflexVacuumAgent(location,status)
      # make the move:
      self.robot.move(act)
def INIT(engine):
   return SimpleBrain('VacuumRobotBrain', engine)
```

Figure 10: A Pyro program to clean up two rooms using a simple reflex brain. After Russell and Norvig (2003), page 46.

suck up dirt and move between two locations, *A* and *B*. Now consider the code to implement it on as a robot within Pyro. Of course, building such a robot for such a simple example would not be worthwhile. However, if the students could easily have access to such a robot, and the concepts would carry over into the rest of the course, then it could be a valuable concrete example from which one can build more complex concepts.

In this Pyro variation, the perceptions and actions are represented by symbols. Figure 10 shows that the perceptual value of *status* is either *dirty* or *clean*, the value of *location* is either *A* or *B*, and movements are *suck*, *left* and *right*. However, the methods of accessing the sensors and affecting the motors are identical to those used to interact with real robots. Using Pyro in this manner could be useful for just exploring AI. However, this agent-based symbolic use also prepares the student for exploring any number of other topics in robotics.

Because Pyro allows students to immediately focus on the most abstract, top-down issues in autonomous control, we have been able to incorporate Pyro into a variety of courses. Specifically, at our institutions, the following courses have used Pyro: *Introduction to Artificial Intelligence, Cognitive Science, Emergence, Androids: Design & Practice, Developmental Robotics, Mobile Robotics, Robotics II, Senior Theses*, and summer research projects for undergraduate students as

well as high-school students. To date, we have recorded that Pyro has been used in courses in at least two dozen educational institutions, and out of the classroom in at least thirty educational institutions.

Based on our own experiences and from those reported by early adopters of the system, it is clear that wherever in the curriculum robotics is used, Pyro can provide an accessible and powerful laboratory environment. In all of the instances, students are able to successfully write several robot control programs for real and simulated robots. In most instances, students learned Pyro and robot programming by following the tutorial materials we have created. The kinds of exercises and the extent to which Pyro was used varied depending on the course and its focus. The exercises span the entire spectrum of difficulty, ranging from modifying existing robot brains to research-level work in robotics. In one instance, three students from Bryn Mawr College developed a robot Tour Guide (Chiu *et al.* 2004) that gave tours of the Science Building.[1] Next, we present a couple of sample instantiations of undergraduate AI courses that were modified to include the use of Pyro.

Pyro in Artificial Intelligence Courses

In this section we present an overview of two versions of an undergraduate junior/senior-level AI course at two similar institutions: Bryn Mawr College and Swarthmore College. Demonstrating how Pyro can be integrated into an upper-level AI course is perhaps the best way to highlight the flexibility available to instructors.

At both colleges, the AI course is typically taught every other year. The Bryn Mawr course followed a traditional, agent-oriented approach based on Nilsson's book (Nilsson 1998), while the Swarthmore course had a machine learning focus based on Mitchell's book (Mitchell 1997). At both colleges, the labs were designed to introduce students to the python programming language, the tools available within Pyro, and the topics being covered in class. Most labs were relatively short in duration, typically lasting only a week. Some of the labs were designed to allow students to explore a topic in much more depth and lasted two to three weeks. Tables 1 and 2 provide an overview of the two courses.

At Bryn Mawr College, the AI course includes both computer science majors and non-majors. Typically, anywhere from 30-40% of the students in the class are from outside of computer science, most without much prior knowledge of programming. As final projects in the Bryn Mawr class, students had robots learn to do wall-following using neural networks, created weather prediction systems using neural nets, wrote game playing programs for Connect Four, Othello, and a Checkers variant that uses chance, and had systems which learned static evaluation functions for Konane. All of these projects used Pyro and/or python.

At Swarthmore College, the course is intended for computer science majors who have completed both CS1 and CS2. As final projects in the Swarthmore class, the majority of the students chose a task in which the robot would be controlled by a neural network and the weights of the network would be evolved by a genetic algorithm. The most ambitious robot learning project involved a three-way game of tag in which each robot had a unique color: the red robot was chasing the blue robot, the blue robot was chasing the green robot, and the green robot was chasing the red robot. The neural network brain for each robot had the same structure, but the weights were evolved in a separate species of the genetic algorithm. The reason for this was to allow each robot to develop unique strategies. Other robot learning projects from the class included having a robot gather colored pucks scattered randomly throughout the environment, having a robot navigate a PacMan-inspired maze while avoiding a predator robot, and having a robot trying to capture a puck from a protector robot.

Pyro's infrastructure allowed the students to focus on the most interesting aspects of the project, such as the environment, task, and network architecture. The abstractions provided within Pyro enabled the students to easily integrate various AI modules (neural networks and genetic algorithms, for example) and develop quite sophisticated robot learning projects in a short amount of time (typically two to three weeks). Pyro's accessible interface and comprehensive infrastructure encourages experimentation with AI and robotics algorithms. This experience may then motivate the students to delve more deeply into the algorithms to better understand the details that may impact system performance.

Pyro in Robotics Courses

In addition to AI courses, Pyro has been used in AI-based robotics courses at both the undergraduate and graduate level. Topics covered at the University of Massachusetts Lowell include robot architectures, vision, machine learning (including neural networks and reinforcement learning), mapping and localization, and multi-agent robotics. The course uses Pyro modules for weekly labs, then culminates in a three week project at the end of the term. Student projects included the following.

Laser Tag: Students designed hardware to send and receive infrared signals, then wrote software to make the game-playing robots locate and target each other. Robots were programmed with different strategies to make the game more interesting.

Robot Slalom: Students used computer vision to find gates in a slalom course that ran down a hallway and around corners.

Pick Up the Trash: Students used computer vision to find trash (Styrofoam cups) and recycling (soda cans), then deliver the found items to the appropriate bins (trash can or recycling bin). In two weeks, students

[1] The students applied for and obtained funding for this project from the Computing Research Association's (CRA) Collaborative Research Experiences for Women (CREW) Program.

Text	*Artificial Intelligence: A New Synthesis* (Nilsson 1998)
Topics	Stimulus-response (S-R) agents, Learning in S-R agents, Evolutionary computation, Model-based agents, State-spaces, Search, Game playing, Logic and knowledge representation, Natural language understanding, Augmented transition networks.
Labs	1. S-R Agents (Braitenberg Vehicles) in Pyro (2 exercises). 2. Search: Uninformed searches on 8-puzzle in python. 3. Wall-following behavior in a robot in Pyro. 4. Centering a robot in a room in Pyro. 5. Game Playing: Konane (Hawaiian Checkers) in python. 6. Final Projects: Independently chosen by students.
Web Page	`cs.brynmawr.edu/Courses/cs372/fall2004`

Table 1: Example of an AI course at Bryn Mawr College

Texts	*Machine learning* (Mitchell 1997) and excerpts from: *AI: Structures and Strategies for Complex Problem Solving* (Luger & Stubblefield 1993), *Artificial Intelligence: A Modern Approach* (Russell & Norvig 2002), *Understanding Intelligence* (Pfeifer & Scheier 1999), and other selected papers on machine learning.
Topics	Game playing, Machine learning: Neural networks, Recurrent neural networks, Decision trees; Genetic algorithms, Evolving networks with GA's, Reinforcement learning, Braitenberg vehicles, Behavior-based control, Robot learning
Labs	1. State-space search in python. 2. Game Playing: Konane in python. 3. Neural networks in Pyro. 4. Evolutionary computation in Pyro. 5. Wall-following robot in Pyro and on Pioneer robot. 6. Learning tasks on robots in Pyro.
Web Page	`web.cs.swarthmore.edu/~meeden/cs63/s04/cs63.html`

Table 2: Example of an AI course at Swarthmore College

were able to complete what had been a competition in the 1994 and 1995 AAAI Robot Competition.

As is evident from the use of Pyro in AI and Robotics courses, Pyro enables students at all levels to do robotics projects that in the past were only feasible by research teams. This, we believe, is one of the biggest payoffs of Pyro. It brings aspects of current research into the curriculum in an accessible, low-cost manner.

Conclusions and Future Directions

The Pyro project is the latest incarnation of our attempts to make the teaching of autonomous mobile robots accessible to students and instructors alike. We have developed a variety of programs, examples, and tutorials for exploring robotics in a top-down fashion, and we are continuing to add new curricular modules. Some of these modules are created by students in the classes, others by the authors, and some by faculty at other institutions who have adopted Pyro. Modules currently under development include multi-agent communication, reinforcement learning, logic, planning, topics in manipulation (such as inverse kinematics for the AIBO), and localization.

We believe that the current state-of-the-art in robot programming is analogous to the era of early digital computers when each manufacturer supported different architectures and programming languages. Regardless of whether a computer is connected to an ink-jet printer or a laser printer, a computer today is capable of printing on any printer device because device drivers are integrated into the system. Similarly, we ought to strive for integrated devices on robots.

Our attempts at discovering useful abstractions are a first and promising step in this direction. We believe that discoveries of generic robot abstractions will, in the long run, lead to a much more widespread use of robots in education and will provide access to robots to an even wider range of students. Our goal is to reduce the cost of learning to program robots by creating uniform conceptualizations that are independent of specific robot platforms and incorporate them into an already familiar programming paradigm. Conceptualizing uniform robot capabilities presents the biggest challenge: How can the same conceptualization apply to different robots with different capabilities and different programming APIs?

Our approach, which has been successful to date, has been shown to work on a variety of real and simulated robots. We are striving for the write-once/run-anywhere idea: robot programs, once written, can be used to drive vastly different robots without making any changes in the code. This approach leads the students to concentrate more on the modeling of robot brains by allowing them to ignore the intricacies of specific robot hardware. More importantly, we hope that this will allow students to gradually move to more and more sophisticated sensors and controllers. In our experience, this more generalized framework has resulted in a better integration of robot-based laboratory exercises in the AI curriculum. It is not only accessible to beginners, but is also usable as a research environment for our own robot-based modeling.

Acknowledgments and Resources

We would like to thank all of the people that have contributed to the Pyro Project. This includes instructors and students that have provided feedback, code, course modules, bug reports, and problem exercises. This work is funded in part by NSF CCLI Grant DUE 0231363. Pyro source code, documentation, and tutorials are available at *www.PyroRobotics.org*.

References

ActivMedia. 2003. URL for the Pioneer robot and the Aria simulator is `http://www.activrobots.com/`.

Arkin, R. C. 1998. *Behavior-based robotics*. MIT Press.

Blank, D.; Kumar, D.; Meeden, L.; and Yanco, H. 2005. Pyro: A python-based versatile programming environment for teaching robotics. *Journal on Educational Resources in Computing*.

Chiu, C.; Butoi, I.; Thompson, D.; Blank, D.; and Kumar, D. 2004. Greeted by the future: Tour guide robot. URL is http://mainline.brynmawr.edu/TourGuide/.

Gerkey, B.; Vaughan, R.; and Howard, A. 2003. The Player/Stage project: Tools for multi-robot and distributed sensor systems. In *Proceedings of the International Conference on Advanced Robotics*, 317–323.

Group, R. S. S. M. 2005. URL for the robocup soccer server is `http://sserver.sourceforge.net/`.

Koenig, N., and Howard, A. 2004. Design and use paradigms for Gazebo, an open-source multi-robot simulator. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*.

KTeam. 2005. URL for the Hemisson robot is `http://www.hemisson.com`.

Luger, G., and Stubblefield, W. 1993. *AI: Structures and strategies for complex problem solving*. The Benjamin/Cummings Publishing Company, Inc., second edition.

Mitchell, T. M. 1997. *Machine Learning*. Boston, MA: McGraw-Hill.

Mondada, R.; Franzi, E.; and Ienne, P. 1993. Mobile robot miniturization: A tool for investigation in control algorithms. In *Proceedings of the Thrid International Symposium on Experimental Robots*.

Murphy, R. 2000. *Introduction to AI Robotics*. MIT Press.

Nilsson, N. 1998. *Artificial Intelligence: A New Synthesis*. Morgan Kaufmann.

Pfeifer, R., and Scheier, C. 1999. *Understanding Intelligence*. Boston, MA: MIT Press.

Robotics, E. 2005. URL for the ER1 robot is `http://www.evolution.com`.

Russell, S., and Norvig, P. 2002. *Artificial Intelligence: A Modern Approach*. Englewood Cliffs, NJ: Prentice Hall, 2nd edition.

Sony. 2005. URL for the AIBO robot is `http://www.sony.net/products/aibo/`.