Myro-C++: An Open Source C++ Library for CS Education Using AI

John R. Hoare, Richard E. Edwards, Bruce J. MacLennan, and **Lynne E. Parker**
Department of Electrical Engineering and Computer Science
University of Tennessee, Knoxville
{jhoare,redwar15,maclennan,parker}@eecs.utk.edu

Abstract

In this paper we present Myro-C++, developed at the University of Tennessee. Myro-C++ is a C++ port of the Python Myro library that was written by the Institute for Personal Robots in Education (IPRE) at Georgia Tech and Bryn Mawr College. Myro-C++ is publicly available, open source software, released under the GPLv3 open source license. At the time of writing, the library has been used six semesters for the CS1 course at the University of Tennessee, Knoxville. The library contains functions for control of the robot and access to sensor information, and provides the ability to display the live camera image from the robot into a video window. This library is used as a teaching tool in our CS1 course where students learn basic programming fundamentals using multiple artificial intelligence based labs. In addition to the software, the IPRE book, *Learning Computing with Robots*, has been edited to use C++ examples and explanations, and is freely available. We also present example programs that we use as laboratory assignments in our Introduction to Computer Science course, which are also freely available.

Introduction

The Institute for Personal Robots in Education (IPRE) applies robots as a teaching tool in order to enhance student interest and increase student enrollment in computer science (Blank 2006). IPRE's coursework and concepts use AI techniques such as robot control paradigms, computer vision, game playing, learning, etc. as vehicles to teach fundamental computer science concepts and skills (Kumar et al. 2008). By using robots as a teaching tool, IPRE hopes to give computer science students more hands on experience, and generate more interest in the field. A strong value of the program is to allow personal robots to be available for all students so that they may be able to work with the robots wherever they please. This approach gives "the freedom to choose where [the students] work, and sometimes play or show-off with the robots" (Summet et al. 2009). The robot is treated as a peripheral to the computer; students do not write low level software (i.e. the robot's micro-controller), but instead write software for a personal computer. Thus, the robot merely is an accessory of the computer (IPRE 2007).

Copyright © 2011, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

Figure 1: The Parallax Scribbler with IPRE Fluke attached. This is the hardware configuration supported by both Myro and Myro-C++. The Scribbler+Fluke combination is sold for about $200 at the time of writing. (Photo courtesy of IPRE)

The software that supports the IPRE's coursework is called Myro, and is written in Python by Georgia Tech and Bryn Mawr College. This software has many capabilities, including access to the robot's sensors, and control of the robot's motors. Additionally, the Myro library supports several other "helper functions" such as graphic displays and drawing, GUI dialogue, text to speech support, and so on.

Some institutions, however, desire their introductory computer science courses to teach C++ syntax and usage in preparation for higher level courses. In such institutions perhaps an overwhelming curriculum overhaul would be necessary to use Python as an introductory language, or additional instruction would be required in later courses where students switch languages. To address this problem, we have created a C++ version of the Myro software, which we call Myro-C++. By using this C++ library, institutions are able to use robots and the general IPRE philosophy and labs, while also teaching C++ as an introductory language[1].

The purpose of this paper is to increase awareness of the Myro-C++ library, the associated coursework, and materials which we have made freely available. We hope to enable other instructors who wish to use the IPRE philosophy in

[1]The purpose of this paper is not to advocate for the use of C++ over Python as a CS1 language. We instead wish to increase awareness that we have created a C++ alternative to Myro.

their courses, but would otherwise be unable or unwilling due to the Python programming language requirement.

In this paper, we will first discuss the hardware platform used by Myro and Myro-C++. Then, we will discuss the features that Myro-C++ provides, including the new VideoStream and Filters features. We will then give a brief discussion about adapting the textbook *Learning Computing with Robots*, edited by Deepak Kumar, as well as the associated coursework, to use C++. Next, we present example programs and uses that we have students implement in our Introduction to Computer Science course. Finally, we will give a brief discussion of the successfulness of the software, and then conclude the paper and discuss future directions in which we hope to go with the library.

Hardware

The robotic platform used is the Parallax Scribbler Robot, with the IPRE fluke that attaches to the robot via the serial port, as seen in Figure 1. The robot has two motors for differential drive control at various speeds, three light sensors, a stall sensor, two downward-facing line sensors, a speaker, and a pen-port. Additionally, the robot has a BASIC Stamp 2 micro-controller, which runs custom firmware to communicate with the IPRE Fluke. The IPRE fluke adds infrared rangers, bluetooth communication, and a color video camera. The robot and fluke combination costs approximately $200, which makes for an affordable platform that can be purchased or loaned to every student.

As stated in the introduction, the robot functions as a peripheral to a computer. While the robot contains a programmable micro-controller, students write their code for a desktop computer. While running their program, the computer sends control signals to the robot though the bluetooth connection on the fluke. This way, students can leverage the computing power and memory of the desktop computer, and not be constrained by the limited resources of the robot's micro-controller.

Myro C++

While the Myro library allows users to program the Scribbler robot using the Python programming language, some institutions desire their introductory computer science courses to teach C++ syntax and usage in preparation for higher level courses. To address this situation, Myro-C++ was developed so the general philosophy of IPRE can be leveraged while using the C++ language.

The Myro-C++ library is based on the Python Myro library, matching the API closely, when possible. While the Myro-C++ library does not yet support all the functionality that the Python Myro provides, such as speech or graphics drawing, the library does currently support access to all of the available sensors discussed in the Hardware section, as well as all movement commands. Myro-C++ also includes an interface to manipulate and display pictures taken with the cameras, as well as the ability to stream the pictures from the robot to a video window, called the VideoStream. The VideoStream supports filters, which provides a way to perform image processing on the live video, and see the results immediately. The library works on POSIX systems, and was tested on linux, Mac OSX, and cygwin on windows.

Connecting and General Usage

When students want to connect to a robot, they have two options:

1. Use connect(), and use the global robot variable.
2. Create a Scribbler object, connect to it manually, and use that object.

Students generally use method (1), because it is simpler. Method (2) could be used to allow for control of multiple robots from one control program. A simple program that drives the robot forward is shown below:

```
#include<Myro.h>
#include<iostream>
int main(){
  connect();
  // Move the robot forward at full speed
  // for 5 seconds
  robot.forward(1,5);
  // disconnect the robot
  disconnect();
  return 0;
}
```

Sensors

Myro-C++ allows access to any sensor by means of a monolithic get() function. Because C++ is a statically-typed language, unlike Python, this function returns a **void***, which the user must cast into the appropriate return type. While we support this interface in order to match the Myro interface, it is not as convenient because it requires students to both cast the return value, and manage the memory that the get() function creates (i.e., must call **delete**). To address this problem, we provide a function with a defined return type for each sensor, so its use is more straightforward. Therefore, the recommended approach is to use the associated get function for the specific sensor. To access the light sensors, for example, one would use the getLights() function, which works as follows:

```
vector<int> lights = robot.getLights();
int left = lights.at(0);
int center = lights.at(1);
int right = lights.at(2);
```

There are similar "get" functions for the each of the sensors on the robot.

Movement

The interface to move the robot is provided by several functions, provided below.

- robot.motors(**double** leftSpeed, **double** rightSpeed)
  Provides a way to directly set the speed of each motor on the robot, until changed with another call of motors, or some other movement command.
- robot.turnLeft(**double** speed, **double** time)

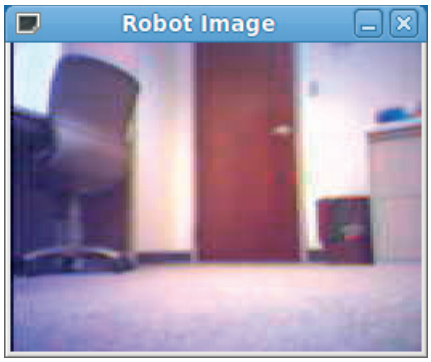

Figure 2: A still capture of the VideoStream window.

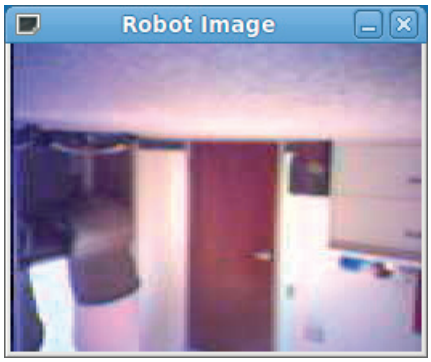

Figure 3: A still capture of the VideoStream window with the invert filter active.

- robot.turnRight(**double** speed, **double** time)
- robot.forward(**double** speed, **double** time)
- robot.backward(**double** speed, **double** time)

The turnLeft(), turnRight(), forward(), and backward() functions move the robot in the specified direction for the specified amount of time.

VideoStream

A major feature provided by Myro-C++ is the VideoStream object. This feature turns the robot into a mobile streaming video platform. Opening the VideoStream is as simple as creating a VideoStream object, and calling its startStream() function, as follows:

```
// Create the VideoStream Object,
// Provide it a pointer to the robot,
// and the Display mode (Color/BW/Blob)
VideoStream vs(&robot, VideoStream::COLOR);
vs.startStream()
// Do other things here,
// move the robot, etc.
...
// When finished with VideoStream
vs.endStream()
// Or just let the object be destroyed
```

The VideoStream runs in its own thread and allows the user to have their code perform other operations, such as

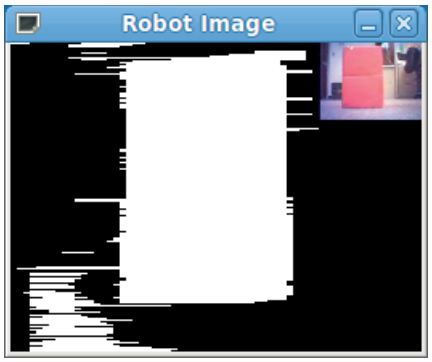

Figure 4: The VideoStream running with the picture-in-picture filter, while tracking a large red box.

driving the robot around while using the VideoStream to "see" what the robot "sees." The example code above creates an image window like in Figure 2, which displays and updates with the current image that the robot can see. With no other communications, the VideoStream window updates approximately 1.8 frames per second.

Filters

In addition to providing an interface for students to turn their robots into mobile cameras, the Myro-C++ library provides an interface for the students to create filters that modify the images before being displayed. For instance, Figure 3 shows a filter that inverts the image before displaying in the VideoStream window. Creating a filter for the VideoStream is done by implementing the filter() function of an inherated Filter class. The code for the invert filter (Figure 3) is shown below:

```
class InvertFilter: public Filter {
  public:
  virtual void filter(Picture * image) {
    int height = getHeight(image);
    int width = getWidth(image);

    Pixel temp;
    for(int h = 0; h < height/2; h++) {
      for(int w = 0; w < width; w++) {
        temp = getPixel(image,w,h);
        setPixel(image,w,h,
            getPixel(image,w,(height-1)-h));
        setPixel(image,w,(height-1)-h,temp);
      }
    }
  }
};

// In main() after the VideoStream has been created
InvertFilter* inv = new InvertFilter();
vs.addFilter(inv);
```

Filters can be added or removed from the VideoStream at any point via the addFilter() and delFilter() functions. This allows students to experiment dynamically with different types of filter combinations. These image filters provide

(a) Without Light Filter (b) With Light Filter

Figure 5: Figure (a) depicts the gray scale VideoStream without the light filter applied. Figure (b) depicts the gray scale VideoStream with the light filter applied. The light filter draws only the brightest region of the image while drawing the rest of the image black.

a mechanism for students to experiment with image processing techniques and receive immediate visual feedback through the VideoStream (Figure 2).

In addition to providing an interactive learning experience, students can use the filter interface to debug their assignments that involve basic image processing algorithms. For instance, the students have a light following assignment that requires that students use the robot's camera to follow a flash light. In the assignment, students can debug their light following algorithm by creating a filter that uses their light following function to determine and draw the brightest image region, and draw the darker regions black (Figure 5). By providing real-time feedback to students through the VideoStream, students are able to debug their light following algorithm more efficiently and with less frustration.

When using blob-detection capabilities of the IPRE Fluke board (using robot.takePicture("blob")), the picture-in-picture filter provides a simple way to see both the "blob" picture, as well as a normal "color" picture. The picture-in-picture filter displays the black and white blob image from the robot with the original color image scaled down in the top right corner (Figure 4). This allows the students to associate the actual scene with the blob tracking scene that the robot is currently reporting. The visual feedback from the picture-in-picture filter helps students understand which components of the blob belong to the actual object they want the robot to follow, and debug their blob following algorithm accordingly.

The Filter has a lightweight and flexible interface that allows novice students to experiment with basic image processing techniques and receive immediate visual feedback. Ultimately, the filter interface helps students have a more interactive learning experience, which would hopefully lead to an increasing student interest in Computer Science. In addition, the filter interface provides a mechanism to help extend the IPRE philosophy into higher level computer science courses.

Compiling

Unlike Python, C++ is a compiled language. Before running any code, it is necessary to compile it, and link against the Myro-C++ library. Myro-C++ provides a mechanism to make this process straightforward, using the "pkg-config" program, as well as a "myro-cpp-config" shell script. If we want to compile a program, "dance.cpp", we use the following at the command line:

```
g++ `myro-cpp-config --cflags --libs` dance.cpp -o dance
```

This will create an executable named "dance," that can now be executed. Instead of repeatedly running the command above, we provide students with a makefile for each lab; students compile their programs by simply running "make." Example makefiles are provided with Myro-C++.

Open Source Software

Myro-C++ has been released as an open source package, under GPLv3, so any interested party can download, install, and/or modify the software. Because the software is open source, students are able to download and view the source code to see how the library functions. Also, parties interested in improving Myro-C++ are able to get the source code and add additional functionality to the library. Students have found and used functions for personal projects that were not documented or used for the normal laboratory assignments by browsing the source code and Doxygen documentation. The software is available to download in source form, an OS X installer form, and packed for Ubuntu Linux operating systems via a Personal Package Archive hosted on Launchpad at `http://www.launchpad.net/myro-c++`. Additionally, the software can be compiled on windows, through the use of cygwin, which provides a linux-like environment for windows. Future versions of Myro-C++ are planned to run natively in windows, without the use of cygwin.

Adapting the IPRE Curriculum to C++

In addition to developing a C++ version of Myro, it was necessary to adapt the IPRE curriculum to C++. Primarily this involved modifying the textbook *Learning Computing with Robots*, edited by Deepak Kumar, to use C++ examples rather than Python examples. For the most part this was straightforward, but there were occasional challenges. For example, since Python is an interpreted, dynamically typed language, it is easier to present to beginning students than C++, with its elaborate static typing. Nevertheless, in converting the text to C++ we tried to adhere to the original book's philosophy of introducing ideas in the simplest way possible and limiting C++ details to what the student needs to know at a given point. The resulting book, *Learning Computing with Robots in C++*, is available on-line at `http://myro-cpp.sf.net`. In addition, we decided to leverage existing IPRE curricular material as much as possible and to follow Georgia Tech's lead in its introductory IPRE-based computer science course, which uses *How to Think Like a Computer Scientist: Learning with Python* by J. Elkner, A. B. Downey, and C. Meyers in addition to *Learning Computing with Robots*. Therefore we chose Downey's *How To Think Like A Computer Scientist: Learning with C++*, which is available under the GNU Free Documentation License; a few chapters of the available edition required significant updating to conform to standard C++. The class lectures proceed through the two books in parallel, often alternating chapters, and the laboratory exercises are coordinated with the lectures.

Example Programs and Uses (Labs)

In this section we present applications that are used in our Introduction to Computer Science course. Students implement these applications using Myro-C++, often with no prior programing experience. All of these labs and more, are available at `http://myro-cpp.sf.net`. The list of presented labs is not exhaustive; many of the labs and curriculum at `http://wiki.roboteducation.org/Educator_Resources` can be easily adapted to use Myro-C++.

Line Following

Using the line sensors located on the bottom of the robot, the students write a line following program to follow a path that is written on the ground. This is commonly implemented as a state machine, and students do a lot of hands on tweaking of the robot to try to get their robot to complete the course in the shortest amount of time. The line following task is often one of the students' favorite, as it provides a lot of freedom for them to try new approaches for completing the course quickly. The line following "brain" is implemented using conditional statements and loops, based on the state of the sensors.

Braitenberg Vehicles

Some of Valentino Braitenberg's most well-known work centers on thought experiments with what he calls "vehicles." These vehicles have simplistic controls, yet exhibit apparently complex behavior. Each experiment details a vehicle that has a small set of sensors, and how those sensors can be connected to the vehicle's motors such that the connections mirror the neurological connections in living creatures. The resulting vehicles seem to be capable of complex behaviors like fear, aggression, love, free will, etc. (Braitenberg 1986). Using the Myro-C++ library, the aggressive vehicle is implemented in the following function:

```
void aggressive(int seconds){
  while(timeRemaining(seconds)) {
    int l = robot.getLight("left");
    int r = robot.getLight("right");
    robot.motors(normalize(r), normalize(l));
  }
  robot.move(0,0);
}
```

The normalize function converts light sensor readings into the range of the motors. Different normalize functions can result in different behaviors of the robot. In this lab, students learn to write functions and use loops by implementing the aggressive, alive, etc., vehicles and creating a menu to choose which vehicle should run.

Light Following

While the Braitenberg vehicles follow light using the light sensors, by using simple image processing techniques, students can use the camera to follow light. Students write code to separate the image taken by the robot into regions, and determine which is the brightest region by using nested loops iterating through every pixel. Using what was calculated as the brightest region, students then have the robot execute the best appropriate action of turning left, turning right, or traveling forward.

Blob Following

Similar to the Light Following assignment, students train the robot to detect a blob in an image. Myro-C++ supports taking a "blob" type picture, which once trained, returns a black and white image where white is the detected color, and black is not (Figure 4). Using the blob picture and following the same approach from the light following assignment, the robot can be track a specific color. To train the blob detection, students must have their program present the user with an image that has had its training area outlined, and ask the user if the image is good.

Text Command Driving

For this example, students create a well-defined language in which they can tell the robot what to do. Examples of commands that are given are "turn left 90", causing the robot to turn left 90 degrees. Students implement a parsing function which parses the input string and makes the robot perform the correct action, given the command. Students implement two "modes" of operation: a manual drive mode, where a VideoStream window is opened, and the students tele-operate the robot by typing commands to the robot, and an auto-drive mode where the robot reads commands from a student provided "action" file. This lab requires students to

Figure 6: A panoramic image created using a least-squares approach to calculate overlap, and using the Myro-C++ interface to take the pictures and move the robot, as well as to access the pixel data of the images.

write robust code that can handle erroneous input, as well as write a string tokenizer for parsing the input strings.

Panoramic Creation

Students use the robot as a platform to take multiple images and stitch them together into a panoramic image as illustrated in Figure 6. This lab consists of two tasks to be accomplished by students (split into two separate laboratory assignments.) For the first task, students are provided a vector of images, along with a vector of overlap points. Students copy the non-overlapping portions of the individual images to form a final, panoramic image. In the second task, students write a function that calculates the overlap points between each pair of images, using a least squares error approach comparing the columns of two images to find the two most alike columns. Students must use nested loops, as in the Light/Blob following labs; however, this lab adds more complication, as they are not only accessing the data in the picture, but they must now copy it. Additionally, students are provided with a mathematical formula to calculate the least squared error, and must convert this to C++ code.

Student Response

It is difficult to provide information on the effectiveness of Myro-C++ as a teaching tool beyond anecdotal claims. Nevertheless, Myro-C++ appears to function effectively as a teaching tool, and has inspired several CS1 students to pursue independent studies so they can continue working with the robots beyond the CS1 course. A more through analysis of the effectiveness of using robots for CS1 using Myro-C++ is the topic of future work.

Conclusions and Future Work

In this paper we have presented Myro-C++, which is released open source at `http://myro-cpp.sf.net`. This software package allows users to program the Parallax Scribbler Robot with IPRE Fluke board in C++. Myro-C++ has additional functionality that is not available in the Python Myro, such as the VideoStream and Filter components. The library is actively maintained, and will be updated with additional functionality and bug fixes. Additionally, we have modified the the textbook *Learning Computing with Robots*, edited by Deepak Kumar, to use C++ examples, resulting in *Learning Computing with Robots in C++*. Finally, we presented laboratory assignments that are given to students in our Introduction to Computer Science courses. All of these materials are available in full at `http://myro-cpp.sf.net`.

The next version of Myro-C++ will include a graphics drawing library, similar to that in the Python Version of Myro. Furthermore, future versions of Myro-C++ will not have the requirement of using Cygwin, and will be supported in windows natively. We also hope to test and support the new Scribbler2 platform.

Acknowledgements

We would like to thank Georgia Tech and IPRE which provided the initial funding for this project. We would also like to thank the University of Tennessee Electrical Engineering and Computer Science Department for partially funding Myro-C++ development, and additional coursework. We would also like to thank Genevieve Walker and Allison Thompson for helping to develop the initial robotics projects used in our Introductory Computer Science course.

References

Blank, D. 2006. Robots make computer science personal. *Communications of the ACM* 49(12):25–27.

Braitenberg, V. 1986. *Vehicles: Experiments in synthetic psychology*. The MIT press.

IPRE. 2007. 2007 Annual Report. `http://www.roboteducation.org/Files/2007-AnnualReport.pdf`.

Kumar, D.; Blank, D.; Balch, T.; O'Hara, K.; Guzdial, M.; and Tansley, S. 2008. Engaging computing students with AI and robotics. In *AAAI Spring Symposium Series, presented at the Symposium on Using AI to Motivate Greater Participation in Computer Science, tech. report SS-08*, volume 8.

Kumar, D. 2008. Learning Computing With Robots in C++. Available at `http://myro-cpp.sourceforge.net`.

Myro-C++ Main Project Page. `http://myro-cpp.sourceforge.net`.

Summet, J.; Kumar, D.; O'Hara, K.; Walker, D.; Ni, L.; Blank, D.; and Balch, T. 2009. Personalizing CS1 with robots. In *Proceedings of the 40th ACM technical symposium on Computer science education*, 433–437. ACM.