

# Myro-C++: A C++ Port of Myro

**John R. Hoare**, Richard E. Edwards,  
Bruce J. MacLennan and Lynne E. Parker

Electrical Engineering and Computer Science Department  
University of Tennessee, Knoxville TN, USA

January 8<sup>th</sup>: 2011 IPRE Winter Workshop

- Think robots and IPRE are great?
- Want or Need to teach C++ instead of Python?
  - Teach C++ syntax for future courses using C/C++.
  - Still providing a simplified interface to CS1 students.

# Myro-C++

- C++ Port of Myro
- Similar syntax and features
- Added feature of VideoStream element

## Availability: Myro-C++

Project URL: <http://launchpad.net/myro-c++>

- Released Open Source under GPLv3.
- Supports Linux, Mac OS X, and Windows (via cygwin)

This talk:

- I'll talk about Myro-Cpp Version 2.2.x
- Will be released very soon (in the next few days)
- If you want a preview before its official release, you can get the (development) code by checking it out using bazaar:

```
bzr branch lp:myro-c++
```

# Outline

- 1 Introduction
- 2 Software
  - Installation
  - General Usage
  - VideoStream
  - Compiling
- 3 Book
  - Learning Computing With Robots
- 4 Labs
  - Labs Overview

# Outline

- 1 Introduction
- 2 Software
  - Installation
  - General Usage
  - VideoStream
  - Compiling
- 3 Book
  - Learning Computing With Robots
- 4 Labs
  - Labs Overview

# Installation: Ubuntu Linux

Compiled Binaries released as PPA on Launchpad.

Installation: In Terminal:

```
sudo apt-add-repository lp:~myro-cpp-dev/myro-cpp
sudo apt-get update
sudo apt-get install myro-cpp myro-cpp-dev
```

# OS X Installation

- Installer provided as a .dmg file with an installer
- Available at <http://launchpad.net/myro-c++>

## Windows Installation (A Multi-Step Process)

- Install Cygwin (<http://www.cygwin.com>) with Myro-Cpp dependencies
- Install fltk-1.1 from cygwin that uses X-Server
- Compile Myro-Cpp from source

```
tar xzf myro-cpp-2.x.x.tar.gz
cd myro-cpp-2.x.x
mkdir build; cd build
cmake ..
make install
```

- Use X-Server whenever running myro-cpp
- We provide a Linux Virtual Machine for students

# Outline

- 1 Introduction
- 2 Software
  - Installation
  - General Usage
  - VideoStream
  - Compiling
- 3 Book
  - Learning Computing With Robots
- 4 Labs
  - Labs Overview

# Outline

- 1 Introduction
- 2 Software
  - Installation
  - **General Usage**
  - VideoStream
  - Compiling
- 3 Book
  - Learning Computing With Robots
- 4 Labs
  - Labs Overview

```
1  from myro import *
2  initialize()
3
4  # Define the new functions
5
6  def botDance(amount, seconds, speed, waitTime):
7      forward(speed)
8      wait(waitTime)
9      backward(speed)
10     wait(waitTime)
11     turnLeft(amount, seconds)
12     turnRight(amount, seconds)
13     stop()
14
15     def botSong(waitTime):
16         beep(3, 800)
17         wait(waitTime)
18         beep(2, 550)
19
20
21     # The main routine
22     def main():
23         speak("Lets dance")
24         botSong(1)
25         botDance(0.5, 2, 0.5, 1)
26         speak("I am the greatest entertainer alive")
27
28     main()
```

```
1  #include <Myro.h>
2  #include <iostream>
3
4  using namespace std;
5
6  void botDance(double amount, double seconds, double speed, double waitTime){
7      robot.forward(speed);
8      wait(waitTime);
9      robot.backward(speed);
10     wait(waitTime);
11     robot.turnLeft(amount, seconds);
12     robot.turnRight(amount, seconds);
13     robot.stop();
14 }
15
16 void botSong(double waitTime){
17     robot.beep(3, 800);
18     wait(waitTime);
19     robot.beep(2, 550);
20 }
21
22 int main(){
23     connect();
24     cout << "Let's Dance!" << endl;
25     botSong(1);
26     botDance(0.5, 2, 0.5, 1);
27     cout << "I am the greatest entertainer alive!" << endl;
28     disconnect();
29 }
```

# So...

- C++ API requires that robot functions are called off the robot object
  - Python API supports this, but doesn't require it.
- The Myro-C++ library is very similar to the Python version
  - We provide more functions for getting sensors than Python-Myro
  - This is so users don't have to cast to the correct return type depending on what sensor they are get()-ing. (although they can if they want!)

```
vector<int>* lineSensors = (vector<int>*)robot.get("line");
```

# Moving The Robot

Same functions as in Myro

- `Scribbler::motors(double leftSpeed, double rightSpeed);`
- `Scribbler::turnLeft(double speed, double time);`
- `Scribbler::turnRight(double speed, double time);`
- `Scribbler::forward(double speed, double time);`
- `Scribbler::backward(double speed, double time);`

All are member functions of the Scribbler Class. (so would call `robot.motors(ls,rs);`)

# Accessing Sensors

Every sensor has an associated `get()` function.

- `vector<int> Scribbler::getLine()`
- `vector<int> Scribbler::getLights()`
- `bool Scribbler::getStall()`
- Etc...

# Features Overview

**Table:** Comparison of Features

<b>Feature</b>	<b>Python</b>	<b>C++</b>
Text-To-Speech	Yes	No
GUI Ask/Selection Functions	Yes	Future
Robot Movement	Yes	Yes
Robot Sensors	Yes	Yes
Image Manipulation	Yes	Yes
Graphics Drawing	Yes	Future
Song Playing	Yes	Future
Web Development	Yes	No
Gamepad	Yes	Future
Video Stream	No	Yes

# API Documentation

- Entire API documentation is available online, and with Myro-C++
- <http://web.eecs.utk.edu/~jhoare/doc/myro-cpp>
- Accessible from Project Webpage:  
<http://launchpad.net/myro-c++>

# Outline

- 1 Introduction
- 2 Software
  - Installation
  - General Usage
  - VideoStream
  - Compiling
- 3 Book
  - Learning Computing With Robots
- 4 Labs
  - Labs Overview

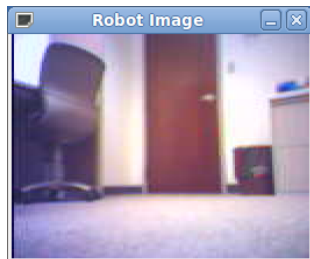
# Outline

- 1 Introduction
- 2 Software
  - Installation
  - General Usage
  - **VideoStream**
  - Compiling
- 3 Book
  - Learning Computing With Robots
- 4 Labs
  - Labs Overview

# VideoStream Element

- The VideoStream provides a way to turn the robot into a streaming web camera.
- The VideoStream continuously takes pictures from the robot, and displays it on the screen.
- Very simple to use:

```
// 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
```

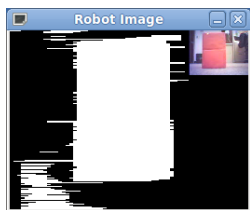


# Filters

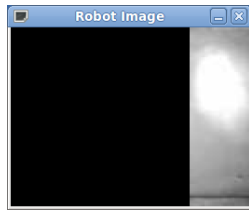
- Additionally, the VideoStream supports "Filters" which allows for real-time video processing



Invert Filter



Picture-in-Picture



Brightest Region

# Implementation

```
1  #include <Myro.h>
2  #include <VideoStream.h>
3  #include <Filter.h>
4  #include <iostream>
5  using namespace std;
6
7  class InvertFilter: public Filter {
8  public:
9      virtual void filter(Picture * image) {
10         int height = image->getHeight();
11         int width = image->getWidth();
12
13         Pixel temp;
14         for(int h = 0; h < height/2; h++) {
15             for(int w = 0; w < width; w++) {
16                 temp = image->getPixel(w,h);
17                 image->setPixel(w,h, image->getPixel(w,(height-1)-h));
18                 image->setPixel(w,(height-1)-h,temp);
19             }
20         }
21     }
22 };
23
24 int main(){
25     connect();
26     VideoStream vs(robot,1);
27     InvertFilter * invert = new InvertFilter();
28     vs.addFilter(invert);
29     vs.startStream();
30     cout << "Press enter to continue" << endl;
31     cin.ignore();
32     disconnect();
33 }
```



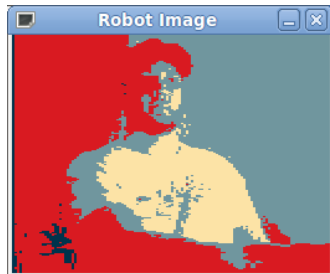
Douglass Blank's webpage has his portrait, with the code:

```
1 from myro import *
2
3 black = (0, 51, 76)
4 red = (217, 26, 33)
5 blue = (112, 150, 158)
6 yellow = (252, 227, 166)
7
8 def obamicon(picture):
9     for pixel in getPixels(picture):
10         gray = getGray(pixel)
11         if gray > 182:
12             setRGB(pixel, yellow)
13         elif gray > 121:
14             setRGB(pixel, blue)
15         elif gray > 60:
16             setRGB(pixel, red)
17         else:
18             setRGB(pixel, black)
19     return picture
20
21 # from obamicon import *
22 # pic = makePicture(pickAFile())
23 # show(obamicon(pic))
```



# Obamicon - VideoStream

```
1  #include <Myro.h>
2  #include <VideoStream.h>
3  #include <Filter.h>
4  #include <iostream>
5  using namespace std;
6
7  Pixel black = {0,51,76};
8  Pixel red = {217,26,33};
9  Pixel blue = {112,150,158};
10 Pixel yellow = {252,227,166};
11
12 class ObamaIconFilter : public Filter {
13     public:
14     virtual void filter(Picture * image) {
15         int height = image->getHeight();
16         int width = image->getWidth();
17
18         Pixel temp;
19         int intensity=0;
20         for(int h = 0; h < height; h++) {
21             for(int w = 0; w < width; w++) {
22                 temp = image->getPixel(w,h);
23                 intensity = (temp.R+temp.G+temp.B)/3;
24                 if ( intensity > 182 )
25                     image->setPixel(w,h,yellow);
26                 else if ( intensity > 121 )
27                     image->setPixel(w,h,blue);
28                 else if ( intensity > 60 )
29                     image->setPixel(w,h,red);
30                 else
31                     image->setPixel(w,h,black);
32             }
33         }
34     }
```



# Outline

- 1 Introduction
- 2 Software
  - Installation
  - General Usage
  - VideoStream
  - Compiling
- 3 Book
  - Learning Computing With Robots
- 4 Labs
  - Labs Overview

# Outline

- 1 Introduction
- 2 Software
  - Installation
  - General Usage
  - VideoStream
  - **Compiling**
- 3 Book
  - Learning Computing With Robots
- 4 Labs
  - Labs Overview

# We Have to Compile Again

- As we all know, C++ is not interpreted like Python, it is compiled.
- So, we have to compile our code before running it.
- Two aspects:
  - 1 Pointing the compiler to the Header files (Myro.h, VideoStream.h, etc)
  - 2 Linking the code with the libmyro-cpp library.

## pkg-config and myro-cpp-config

To deal with problems (1) and (2), we provide both:

- Configuration file for the pkg-config program:  
`pkg-config --cflags --libs myro-cpp`
- myro-cpp-config shell script (to remove pkg-config program dependency)

```
myro-cpp-config --cflags --libs myro-cpp
```

# Makefiles

myro-cpp-config (and pkg-config) allow us to create portable makefiles

- Don't need to hard code paths
- Directories can change from system to system, without changing the makefile

```
1 CC=g++
2 CFLAGS='myro-cpp-config --cflags'
3 LIBS='myro-cpp-config --libs'
4 EXECUTABLES=dance invert obamicon
5
6 all: $(EXECUTABLES)
7
8 dance: dance.cpp
9     $(CC) $(CFLAGS) $(LIBS) $< -o $@
10
11 invert: invert.cpp
12     $(CC) $(CFLAGS) $(LIBS) $< -o $@
13
14 obamicon: obamicon.cpp
15     $(CC) $(CFLAGS) $(LIBS) $< -o $@
16
17 clean:
18     rm -f $(EXECUTABLES) *.o core
```

# Outline

- 1 Introduction
- 2 Software
  - Installation
  - General Usage
  - VideoStream
  - Compiling
- 3 Book
  - Learning Computing With Robots
- 4 Labs
  - Labs Overview

# Outline

- 1 Introduction
- 2 Software
  - Installation
  - General Usage
  - VideoStream
  - Compiling
- 3 Book
  - Learning Computing With Robots
- 4 Labs
  - Labs Overview

# Learning Computing With Robots in C++

Book was modified to use C++, but keeps the same Philosophy as the Original Python version:

- Modified to use C++ examples instead of python examples.
- C++ topics are introduced in a need-to-know fashion
- A few chapters required heavier modification.
- Works along-side *How To Think Like A Computer Scientist: Learning with C++*

# Availability: Learning Computing with Robots in C++

Available free on-line in PDF form:

<http://web.eecs.utk.edu/~mclennan/Classes/102/LCRcpp>

Also linked to from the IPRE wiki.

# Outline

- 1 Introduction
- 2 Software
  - Installation
  - General Usage
  - VideoStream
  - Compiling
- 3 Book
  - Learning Computing With Robots
- 4 Labs
  - Labs Overview

# Outline

- 1 Introduction
- 2 Software
  - Installation
  - General Usage
  - VideoStream
  - Compiling
- 3 Book
  - Learning Computing With Robots
- 4 Labs
  - Labs Overview

# Labs Overview

- 9 Labs
- 7 Use the robot/Myro-C++
- Presented in the order that the students do them

# Movement and Input

Students run experiments with their robot to determine:

- How long it takes to turn a certain amount
- How long it takes to travel a certain amount

Students then use these equations to turn and travel, by reading input from stdin in a loop.

# Navigator Functions

Using the equations from the previous lab, students:

- Create a `turn()` function to turn left or right some number of degrees
- Create a `travel()` function to move forward or backward some distance

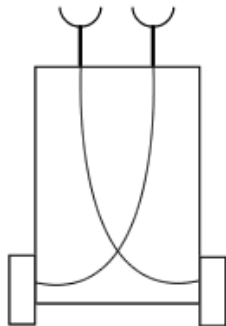
Students write functions to “decode” a message stream.

- The decoded message is then written by the robot, using the `turn()` and `travel()` functions written by the student.

# Braitenberg Vehicles

Vehicles with simplistic controls, yet exhibit apparently complex behaviors.

- Vehicles respond to changes in light.
- Students write numerous different vehicles, and provide a menu to select which vehicle to run.
  - Alive
  - Coward
  - Aggressive
  - Love
  - Explorer



# Light and Blob Following

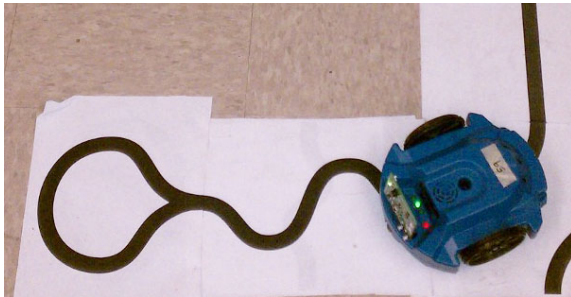
First lab using Pictures and Image Processing.

- `findLight()`: Gets brightest region of the image (left, right, or center)
- `followLight()`: Move robot to face the brightest region of the image (from `findLight()`)
- `trainBlob()`: Provide an interface for the user to train the blob tracking of the robot.
- `followBlob()`: Move robot to face the blob, using `findLight()`.

# Line Following

Use the Line sensors underneath the robot to follow a track.

- Students work to test their code and make it more robust
- Bonus to students with fastest times



# Navigator Class

Create a Navigator Object with:

- `turn()` and `travel()` functions from Lab2
- `manualDrive()` and `autoDrive()`
  - Parse string command
  - `manualDrive()`: Can be tele-operated by the student by typing commands, with VideoStream on screen
  - `autoDrive()`: Can drive automatically by providing a command file

# Panoramic

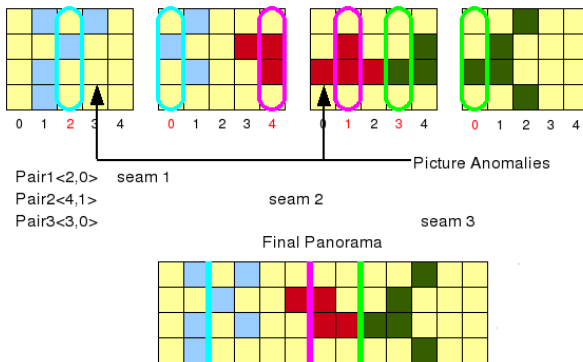


- Take multiple pictures from the robot's camera and stitch them together.
- Given two students in two parts:
  - 1 Image Stitching
  - 2 Calculating Image Overlap

# Panoramic: Image Stitching

Write a function that creates a panoramic, given the overlap between each picture:

```
Picture * buildImage(vector<Picture*> imageArray,  
                    vector< pair<int,int> > overLap,  
                    int& final_width);
```



# Panoramic: Compute Overlap

- Function to compute Overlap between two images:

```
pair<int,int> computeOverlap(Picture * image1 ,  
                             Picture * image2,  
                             int col_offset);
```

- Function to compute for each pair of images:

```
vector< pair<int,int> > buildMergeList(  
    vector<Picture*> imageArray);
```

- Usually this lab is assigned as an “extra credit” assignment.

# Questions?

## Software:

`http://launchpad.net/myro-c++`

## Learning Computing With Robots:

`http://web.eecs.utk.edu/~mclennan/Classes/102/LCRcpp`

## John Hoare:

`http://web.eecs.utk.edu/~jhoare`