ipre

Quick Start Guide and Reference Manual

BRYN MAWR COLLEGE

Microsoft Research

Robotics@GT & Intelligent Machines

ipre QuickStart Guide

IPRE Personal Robot

The IPRE personal robot for introductory computing is a Scribbler robot (manufactured by Parallax) augmented with the IPRE Fluke robot upgrade module.

Bluetooth

Name: IPRE-<Serial #>
PIN: 1234

Getting Started

0. Subscribe to the myro-users mailing list
1. Install Myro software on your PC (see Readme.html on CD)
2. Insert 6 AA batteries into the Scribbler (rechargeable batteries work well)
3. Plug Fluke into the Scribbler's serial port
4. Turn on power switch on Scribbler
5. Connect to the robot via bluetooth (see wiki for connection instructions)
6. Double click the "Start Python.py" icon on your desktop
7. Enter **from myro import *** in the Python window
8. Enter **upgrade('scribbler')** to install the IPRE firmware
9. Enter Bluetooth serial port (from step 5), e.g., COM40, when prompted
10. Enter **initialize( )**
11. You are now ready to explore computer science with robots!

More Information

wiki.roboteducation.org/Start

Additional Materials

Institute for Personal Robots in Education

The institute for Personal Robots in Education (IPRE) applies and evaluates robots as a context for computer science education. IPRE is hosted at Georgia Tech with Bryn Mawr College and initially supported for three years by seed funds from Microsoft Research and the schools themselves.

Desktop PC with Bluetooth

You will need a Bluetooth enabled computer. If you don't have blue tooth built in, or want to extend the range, we recommend the Class-1 Azio USB Bluetooth adapter.

Myro

To use the IPRE robot, you will need to run the installer for the programming language Python and the IPRE Myro Python library. Myro provides a student-friendly Python interface to the IPRE robot. Both are freely available on the web and can be installed from the CD.

Trouble with the Install?

We are very interested in improving our installation CD. If you have comments or questions about how the software install process works, or any other comments, please contact us:

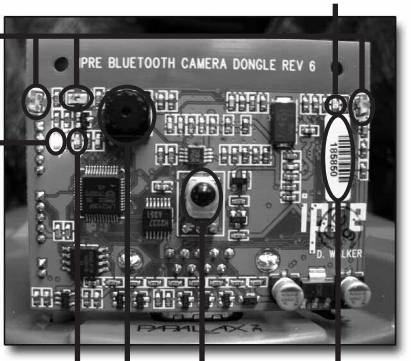
wiki.roboteducation.org/Start
Phone: (610) 526-5024

Getting to Know Your IPRE Fluke

Red Bluetooth LED
Indicates Bluetooth transmission

Infrared Emiters
right, center, left

Green Power LED



Bluetooth PIN 1234

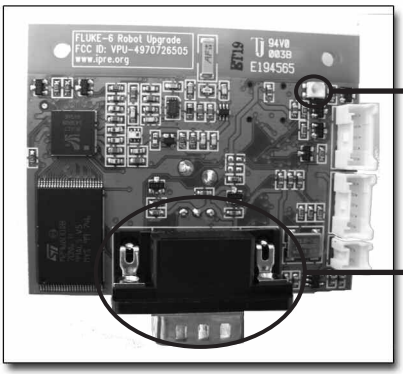
Red User LED

Infrared Receiver

Serial Number
Bluetooth name

Camera

Lens can be manually focused by turning the top of the lens gently in a clockwise or counter clockwise direction. Be sure to only turn the very top of the lens to prevent snapping the lens from the board.



Bright Red User LED
(blinking indicates low power)

Scribbler Serial Connection
Plug the connector into the serial port of the scribbler.

Manufactured under license from the Georgia Institute of Technology.

Myro Cheat Sheet

Institute for Personal Robots in Education
www.roboteducation.org

Getting started:

Double-click the "***Start Python***" icon on the Desktop.

```
>>> from myro import *
>>> init ( )
```

Manual drive:

```
>>> senses ( )
>>> gamepad ( )
>>> joyStick ( )
```

Movement Commands:

SPEED is between 0 and 1
SECONDS is float or integer and is optional

```
>>> forward (SPEED, SECONDS)
>>> forward (.5, 1.2)

>>> backward (SPEED, SECONDS)
>>> backward (1, 2)

>>> turnLeft (SPEED, SECONDS)
>>> turnLeft (1, .3)

>>> turnRight (SPEED, SECONDS)
>>> turnRight (1, .3)

>>> motors (LEFTSPEED, RIGHTSPEED)
>>> motors (-1, 1)

>>> move (TRANSLATE, ROTATE)
>>> move (.5, 0)
```

Myro Time functions:

```
>>> t = currentTime ( )
>>> while timeRemaining (SECONDS):
        doSomething ( )
>>> for t in timer (5):
        print t
```

Sound functions:

```
>>> speak ("Hello " + getName ( ))
>>> getVoices ( )
>>> setVoice (askQuestion ("Pick one",
                           getVoices( ) )
>>> beep (SECONDS, HERTZ)
>>> beep (1, 440)
>>> beep( 1, 440, 880)
>>> playSong (makeSong("a 1/8;"))
>>> makeSong ("1 1; b .25; c 1/4")
```

Picture Commands: *(based on Guzdial)*

```
>>> picture = takePicture ( )
>>> picture = takePicture ("gray")
>>> picture = takePicture ("blob")

>>> show (picture)
>>> show (picture,
        "New Window Name")

>>> for pixel in getPixels (picture):
        red = getRed (pixel)
        green = getGreen (pixel)
        blue = getBlue (pixel)
        setRed (pixel, 0)
        setBlue (pixel, 255 – blue)

>>> savePicture (picture, "name.jpg")
>>> savePicture ([p1, p2], "x.gif")
>>> p = makePicture (100, 100)
>>> p2 = copyPicture (p)
```

Miscellaneous:

```
>>> getName ( )
'Scribby'
>>> setName (“Barney”)
>>> getLight ( )
>>> getBright ( )
>>> getIR ( )
>>> getObstacle ( )
>>> flipCoin ( )
>>> setForwardness ('scribbler-forward')
>>> setForwardness ('fluke-forward')
```

Python:

```
def main (arg1, arg2):
    print “Hello”
    for i in range (10):
        print i,
value = randomNumber ( )
```

Communication:

```
>>> chat = Chat (“lulu”, “password”)
>>> chat2 = Chat (“bozo”, “password”)
>>> chat.send (“bozo”, “Hi, Bozo!”)
>>> chat2.receive ( )
'Hi, Bozo!'
```

Web interface:

```
sendPicture (p, "filename",
      "password", "bozo")
```

Images appear at
myro.roboteducation.org/myweb/

Graphics interface: *(based on Zelle)*

```
win = GraphWin ( )
image = Image (Point (0,0),
          makePixmap(pic) )
image.draw (win)
win [“height”] = 400
win [“height”] = getHeight(pic) + 10
image.move (10,10)
image.refresh (win)
```

Choices:

```
>>> pickAFile ( )
>>> pickAFolder ( )
>>> pickAColor ( )
>>> pickOne (5)
```

Myro Reference Manual

Institute for Personal Robots in Education

Myro Reference Manual

Myro is a framework for programming robots. It is written in the language Python and designed for use in Introductory Computing courses. It is being developed by the Institute for Personal Robots in Education. Please visit **www.roboteducation.org** for more information.

Getting Started

All the commands from Myro are available in Python by importing them. Here is a quick start list:

1. Double-click on the "Start Python" icon on the desktop
2. Enter: **from myro import ***
3. Turn on the robot, and connect via Bluetooth
4. Now, you can enter **init ( )** and type the COM port of your robot

To test that everything is working, try:

```
>>> from myro import *
```

If you have a gamepad:

```
>>> gamepad ( )
```

Otherwise:

```
>>> joyStick ( )
```

Manual Drive

gamepad ( ): runs a program to control the robot through a gamepad. Optionally takes four string parameters that are items for the computer to speak.

senses ( ): Opens a window to see the robot's sensor values.

joyStick(showSensors = 0): opens a joystick window; click and drag to move robot. Pass a 1 to joystick ( ) to see sensor values, too.

Output Functions

beep(self, duration, frequency, frequency2 = None): make a tone. If two tones are given, the robot will combine them.

speak(message, async = 0) - text-to-speech, turns message into spoken words

stopSpeaking ( ) - stop the speaking, if speaking asynchronously

getVoice ( ) - get the voice of the current speaker

getVoices ( ) - get a list of all of the possible voices

setVoice(name) - set the voice to a known voice by name

Input Functions

ask (item) - will ask one time for item (s)

```
>>> ask ("Name")
(window pops up, user enters Sarah, presses Ok button)
'Sarah'
```

raw_input (prompt) - will prompt user for input. Like ask, but no window.

```
>>> raw_input ("How old are you? ")
How old are you? 19
'19'
```

askQuestion (question, [answerList]) - prompt a question and return answer.

```
>>> askQuestion ("Are you ready?")
(window pops up, users selects Yes)
'Yes'
>>> askQuestion ("How many lumps would you like?", ["One", "Two", "Three"])
(window pops up, user selects Three)
'Three'
```

Movement Functions

forward (amount, seconds): move forward, stop any rotation, for number of seconds

```
>>> forward (1, .5)
```

backward (amount, seconds): move backward, stop any rotation, for number of seconds

```
>>> backward (.9, 2)
```

turnLeft (amount, seconds): turn left, stop any forward movement, for number of seconds

```
>>> turnLeft (.4, 1)
```

turnRight (amount, seconds): turn right, stop any forward movement, for number of seconds

```
>>> turnRight (.5, 1)
```

stop ( ): stop all movement

```
>>> stop ( )
```

translate (amount): move forward and backwards. 0 to 1 moves forward; 0 to -1 moves backwards **rotate (amount)**: turn left or right. 0 to 1 turns left, 0 to -1 turns right

NOTE: translate and rotate are independent, although they may both effect each the wheel. That means that a translate ( ) and a rotate ( ) will blend into a meaningful combination.

```
>>> translate (1)  # full speed ahead
>>> translate (-1) # full speed backwards
>>> translate (0)  # stop in the translate direction
>>> rotate (.5)  # half-speed to the left
>>> rotate (-1)  # full speed to the right
```

move (translate, rotate): rotate and translate

```
>>> move (0, 1)  # turn full speed to left
>>> move (0, -1) # turn full speed to right
>>> move (1, 1)  # turn full speed to left while moving full speed ahead
>>> move (.5, 0) # go forward half speed
```

motors (left, right): control the left and right motors

Reading Sensors

getLight (pos): read a light sensor on the scribbler; defaults to "all"

getIR (pos): read an IR sensor on the scribbler; defaults to "all"

getLine (pos): read line sensor on the scribbler; defaults to "all"

getStall ( ): read stall sensor on the scribbler

NOTE: *Every time you issue a move command, the stall sensor resets, and it needs to wait a short time to see whether the motors are stalled. This means that the sensor won't give accurate results if you test it too soon after the robot starts to move.*

getName ( ): read the robot's name

getPassword ( ): read the robot's password

getAll ( ): read all positions of all of the major sensors; returns a dictionary

getVolume ( ): returns 0 or 1

getData ( ): get some bytes stored in the robots memory

getInfo ( ): retrieve information about the robot

getBright ("left" | "middle" | "center" | "right" | 0 | 1 | 2): read one of the Fluke's virtual light sensors. The Fluke's virtual light sensors report the total intensity in the left, center, and right sides of the Fluke's camera.

getObstacle ("left" | "middle" | "right" | 0 | 1 | 2): read one of the Fluke's IR obstacle sensors (see setIRPower below). Higher values mean that IR light is being reflected (e.g an obstacle is detected), a low value means IR is not being reflected and there seems to be open space in that direction. The value ranges from 0 to 6400.

getBattery ( ) gets the voltage of the battery (**note:** If the battery drops below ~6.1V the Fluke's back LED will flash to alert you to change or preferably recharge your batteries)

get (sensor): read the sensor "stall"; or get all readings of "ir", "light", or "line"; or get "all" which is all of those. Also can get "config".

get ("config"): returns meta data about the robot's hardware

get (sensor, pos): read any of the following sensors or items by name and position

```
>>> get ("stall")
0
>>> get ("light", 0)
128
>>> get ("line")
[0, 0]
>>> get ("all")
{'light': [235, 13], 'line': [1, 0], 'ir': [0, 0], 'stall': 0}
>>> get ("all") # using a fluke
{'battery': 6.2436550642715174, 'light': [13176, 3058, 1848], 'ir': [1, 1], 'obstacle': [0, 0, 0],
'bright': [193536, 193536,  193536], 'stall': 0, 'blob': (0, 0, 0), 'line': [1, 1]}
```

Setting Values

setLED (position, value): set a LED

```
>>> setLED ("left", "on")
>>> setLED ("right", "off")
```

setName (name): set the robot's name (limit of 16 characters)

```
>>> setName ("Howie")
```

setVolume (level): set the speaker's volume (0/"off" or 1/"on")

```
>>> setVolume ("off")
>>> setVolume (0)
>>> setVolume ("on")
>>> setVolume (1)
```

setData (position, value): set a byte of data in the robot's memory to a value between 0 and 255.

setLEDFront (value): turn on the led on the front of the Fluke (0 for off and or 1 for on)

```
>>> setLEDFront (1) # turn on the Fluke's front LED
>>> setLEDFront (0) # turn off the Fluke's front LED
>>> set ("led", "front", 0) # turn off the Fluke's front LED
```

setLEDBack (value): turn on the LED on the back of the Fluke. The brightness of this LED is configurable between 0-1.


```
>>> setLEDBack (0.5) # turn on the Fluke's back LED at 1/2 brightness
>>> setLEDBack (1.0) # turn on the Fluke's back LED at full brightness
>>> set ("led", "back", 0) #turn off the Fluke's back LED
```

setIRPower (power): set the output power of the Fluke's IR obstacle sensors (defaults to 135). If getObstacle ( ) always reports high values try lowering the IR output power. If you always receive a zero value, try increasing the power. The power value should be between 0 and 255.

```
>>> setIRPower (135)
>>> setIRPower (140)
```

darkenCamera (level): turn off the camera's auto-exposure, auto-gain, and lower the gain to "level:. This is useful when using the getBright ( ) virtual light sensors.

autoCamera ( ): turn on the auto-exposure, auto-gain, and auto-color-balance.

set (item, value): set a value (for "name", and "volume")

set (item, position, value): set a value (for "led")

```
>>> set ("name", "Duckman")
>>> set ("led", "center", "off")
>>> set ("volume", "off")
```

Flow of Control

If you wished to perform a loop for 5 seconds, you could use one of the two following idioms:

```
while timeRemaining (5):
   print "running..."
for seconds in timer (5):
   print "running for", seconds, "..."
```

Image processing

The following objects and functions are related to the camera functions on the Scribbler. There are two different interfaces: the Multimedia interface (largely based on Mark Guzdial's introductory book: *Introduction to Computing and Programming in Python, A Multimedia Approach*), and the Graphics Object interface (largely based on John Zelle's introductory book: *Python Programming: An Introduction to Computer Science*). These are independent; however, there are methods to move between the two. The first library is built on the second.

Creating a picture, manually, from a file, or from the robot:

```
picture = takePicture ("color" | "gray" | "blob")
    # gets image from robot
picture = makePicture (filename)
    # reads in a image file, examples: PNG, JPG, GIF
picture = makePicture (columns, rows)
    # creates a blank picture
picture = makePicture (columns, rows, array)
    # creates a new picture from an array (a column-major sequence of
    0-255 values that represent Red, Green, Blue (in that order))
picture = makePicture (columns, rows, array, mode)
```

```
    # creates a new picture from an array, where mode = "color", or "gray"
pictureCopy = copyPicture (picture)
    # creates a copy of picture
```

Output:

```
show (picture)
repaint (picture=None)
    # pixel changes don't appear until you repaint ( )
savePicture (picture, filename)
savePicture ([picture, picture, ...], filename)
    # creates an animated GIF
```

The show (picture) function has a couple of mouse functions. You can click with the left mouse on a pixel and you can see "(x,y): (r,g,b)" in the window's status bar. If you click and drag a rectangle in the window, you will set the blob-tracking colors to the ranges of colors inside the bounding box.

show ( ) can also take an optional name:

```
show (pic2, "name2")
```

so that you can show more than one image at a time.

Image dimensions:

```
int_value = getWidth (picture)
int_value = getHeight (picture)
```

Predefined colors:

```
black     = makeColor (  0,   0,   0)
white     = makeColor (255, 255, 255)
blue      = makeColor (  0,   0, 255)
red       = makeColor (255,   0,   0)
green     = makeColor (  0, 255,   0)
gray      = makeColor (128, 128, 128)
darkGray  = makeColor ( 64,  64,  64)
lightGray = makeColor (192, 192, 192)
yellow    = makeColor (255, 255,   0)
pink      = makeColor (255, 175, 175)
magenta   = makeColor (255,   0, 255)
cyan      = makeColor (  0, 255, 255)
```

Creating and manipulating color objects:

```
color = makeColor (r, g, b)
color = pickAColor ( )
color = getColor (pixel)
setColor ([pixel | color], color)
makeDarker (color)  # takes a color and makes it slightly "darker"
makeLighter (color) # takes a color and makes it slightly "lighter"
```

You can also determine the *distance* between two colors using:

```
distance (color1, color2)
```

Pixel manipulation:

```
pixel = getPixel (picture, x, y)
pixels = getPixels (picture)
int_value = getRed (pixel)
int_value = getGreen (pixel)
int_value = getBlue (pixel)
int_value = setRed (pixel, color)
int_value = setGreen (pixel, color)
int_value = setBlue (pixel, color)
int_value = getX (pixel)
int_value = getY (pixel)
r, g, b = getRGB ([color | pixel])
```

Red, green, and blue int_values are between 0 and 255, inclusive.

The getPixels ( ) function is designed to be used with a for-statement, like so:

```
for pixel in getPixels (picture):
   # do something with each pixel
```

You can also make a Pixmap object that allows integration into other graphics objects:

```
pixmap = makePixmap (picture)
      # turns a picture into a pixmap
```

This can be used by the object-oriented graphics system.

```
win = GraphWin (title)
p = Point (x, y)
image = Image (Point (x, y), pixmap)
image.draw (win)
```

See John Zelle's book for more details.

All types of images have an RGB value, even if an image type actually has a palette (such as gif files), or grayscale images. Images that have a palette will use the palette entry for getColor ( ), and will look-up the closest color in the palette when using setColor ( ). For example, if you try to set a pixel to Color (255, 255, 255) but there is no such color in the palette, then the color will be set to a nearby color. Gray scale images will have identical RGB values for each pixel. For example, getColor (getPixel (p, x, y)) might give Color (125, 125, 125) for one position and Color (65, 65, 65) for another. When setting a color in a grayscale image, the single value is actually taken as the average of the 3 components. For example, if you tried:

```
graypic = takePicture ("gray")
setColor (getPixel (graypic, 0, 0), Color (50, 100, 150))
# Doesn't do what you expect!
```

The color of that pixel would end up at 100, (50 + 100 + 150)/3 = 100.

If you wanted to actually make part of a grayscale image a color, then you would need to turn the grayscale into a color image, via something like:

```
# take a grayscale picture:
p = takePicture ("gray")
# make a new color picture the same size:
colorPic = makePicture (getWidth (p), getHeight (p))
# go through each pixel and copy it to new color picture:
for p in getPixels (picture):
    setColor (getPixel (colorPic, getX (p), getY (p)), getColor (p))
```

You can also save any picture to a file:

```
writePictureTo (picture, filename) # for compatibility with Mark Guzdial's book
savePicture (picture, filename) # does the same as above, but has intuitive name
```

copyPicture ( ) is effectively defined as:

```
def copyPicture (picture): newPicture = makePicture (getWidth (picture),
getHeight (picture))
       for x in range (getWidth (picture)):
 for y in range (getHeight (picture)):
   setColor (getPixel (newPicture, x, y),
 getColor (getPixel (picture, x, y)))
       return newPicture
```

Sound:

```
sound = makeSound (filename) # opens a file and returns a sound object
play (sound) # plays the sound file through the computer's speakers
```

Examples

Processing by rows and cols:

```
from myro import *
picture = makePicture (pickAFile ( ))
show (picture)
for i in range (getWidth (picture)):
    for j in range (getHeight (picture)):
        pixel = getPixel (picture, i, j)
        setGreen (pixel, 255)
    repaint ( ) # not every time, just once per col
```

Processing by each pixel:

```
from myro import *
picture = makePicture (pickAFile ( ))
show (picture)
for pixel in getPixels (picture):
  setGreen (pixel, 255)
repaint ( )
```

Advanced Vision Functions

The Fluke can not only take images, but it can do some very simple image processing. In particular, a simple form of on-board image segmentation. If you want to track something in an image based on its color or brightness, the on-board color segmentation can be very useful since its faster than doing it in python. By default, the Fluke is set to track pink objects. You can also graphically select an object in the image to track. First use the show (picture) function and then drag a box around the object you want to track, the software will automatically determine the color bounding box.

We can use the Fluke's computer vision in two ways. First, we can grab a "blob" image from the Fluke. This image is a black and white image with the white pixels being part of the object of interest. Blob images can be transmitted faster than a full color picture.

```
p = takePicture ( )
show (p)
# select object in the image
b = takePicture ("blob")
show (b)
```

For instance, here we see two images of a dog and her toy. The first is a regular color picture and the second is a blob image with the pink dog toy selected.

Another useful function is getBlob ( ) that will return three items, the total number of pixels that fell inside the bounding box, and the average x and y locations of those pixels.

```
pixel_count, average_x, average_y = getBlob ( )
```

Rather than using the mouse to select the bounding box for segmentation, you can manually configure the Fluke using the configureBlob ( ) function call:

```
configureBlob (y_low = 0, y_high = 255, u_low = 0, u_high = 255, v_low = 0, v_high = 255)
```

The parameters to configureBlob ( ) create a bounding box in YUV space. Instead of using RGB which stands for Red/Green/Blue, YUV is an alternate way to describe color. The Y component contains the brightness or intensity information of the pixel, the U/V components contain the color.

Finally, you can also use getBlob ( ) to locate bright pixels. We do this by segmenting bright pixels, meaning the Y components of the pixels are large :

```
configureBlob (y_low=100, y_high=255)
b = takePicture ('blob')
show (b)
pxs, avg_x, avg_y = getBlob ( )
```

Graphics Objects Interface

```
picture = Picture (columns, rows)
pixmap  = makePixmap (picture)
image = Image (Point (x, y), pixmap)
```

Example

```
win = GraphWin ( )
image = Image (Point (0, 0), makePixmap (picture))
image.draw (win)
win["width"] = 451
win["height"] = getHeight (picture) + 10
image.move (10,10)
image.refresh (win)
```

Miscellaneous commands

wait (seconds) - Pause for the given amount of seconds. Seconds can be a decimal number.

```
>>> wait (5)
>>>
```

currentTime ( ) - the current time, in seconds from an arbitrary starting point in time, many years ago. Can you figure out the date of the start time?

```
>>> currentTime ( )
1164956956.2690001
```

flipCoin ( ) - Returns "heads" or "tails" randomly.

```
>>> flipCoin ( )
'tails'
>>> flipCoin ( )
'tails'
>>> flipCoin ( )
'heads'
```

pickOne (value) or **pickOne (value1, value2, ...)** - Returns a number or element randomly.

```
>>> pickOne (5)
0 # randomly returns 0 through 4 evenly over time
>>> pickOne (2, 4, 6, 8)
6 # randomly returns 2, 4, 6, or 8 evenly over time
>>> pickOne ("red", "white", "blue", "green")
6 # randomly returns "red", "white", "blue", or "green" evenly over time
```

randomNumber ( ) - Returns a random number between 0 (inclusive) and 1 (exclusive).


```
>>> randomNumber ( )
0.65218366496862357
```

File and Folder Functions

pickAFolder ( ) - allows you to select a folder

```
>>> pickAFolder ( )
'C:/Python24'
```

pickAFile ( ) - allows you to select a file

```
>>> pickAFile ( )
'C:/Python24/README.txt'
```

Media Functions

readSong (filename) - reads a file in the Song File Format and returns a list of tuples.

makeSong (text) - make a song in the Song File Format where lines are separated by semicolons.

```
>>> makeSong ("c 1; g 1/4; a 1/2; e 3/4;")
[(523.29999999999995, 1.0), (784.0, 0.25), (880.0, 0.5), (659.29999999999995, 0.75)]
>>> singsong = makeSong ("c 1; g 1/4; a 1/2; e 3/4;")
```

saveSong (text or song, filename) - saves a song in tuple format, or in text format

playSong (song, wholeNoteDuration = 0.545) – plays a song on the robot

playSpeech (filename) - play a WAV file

saveSpeech (message, filename) - save the message as a WAV file

Web development

In order to use the web development you first need to run the register ( ) function. You will need to enter the fields as below:

Your email address	:	(your email address)
Your robot's name	:	(any name up to 16 characters, no spaces or other punctuation)
Course keyword	:	owls OR yjackets OR yellowj
Create a Myro password	:	(any password, don't use a good one, but it is encrypted)

The keyword is a code that determines school and section number.
For example, here are some valid entries:

Your email address	:	dblank@brynmawr.edu
Your robot's name	:	Theodore
Course keyword	:	owls
Create a Myro password	:	*********
Your email address	:	someone@mailinator.com
Your robot's name	:	Jester
Course keyword	:	yjackets
Create a Myro password	:	*****

```
Your email address      : someoneelse@mailinator.com
Your robot's name       : Sally
Course keyword          : yellowj
Create a Myro password  : *****
```

If you have a robot connected, it will set the robot's name. Otherwise, when you connect your robot, you should name your robot the same name:

```
setName ("Theodore")
setName ("Jester")
```

You can now surf to your page and edit your HTML section if you wish: http://myro.roboteducation.org/myweb/

To login in, use all lowercase for the robot name, even if you registered it with uppercase letters. The password is case sensitive.

To send a picture to your website, use the sendPicture ( ) function:
sendPicture (picture, photoname, password, robotname = None)

If you leave robotname off, sendPicture ( ) will ask the robot what its name is.

There is currently a limit on the size of a picture that can be sent. Only pictures about the size from the camera will work. You'll get a message if it is too large.

Here is a full example:

```
from myro import *
register ( ) # opens up window
initialize ( )
picture = takePicture ( )
sendPicture (picture, "dormroom3", "bash72hg")
# robot's name is "theodore" so that will get sent
```

If you need to change your password, use the command:

```
setPassword (robotName, emailAddress, newPassword)
takePicture ("color" | "blob" | "gray")
setWhiteBalance ("on" | "off" | 0 | 1)
set ("whitebalance", value)
```

Gamepad

There are two commands: getGamepad ( ) and getGamepadNow ( )

1. getGamePad ( ) waits for an event before returning (ie, is blocking),
2. getGamepadNow ( ) will immediately return the current state of the gamepad.

Gamepad Examples

```
>>> getGamepad ("count")
2
>>> getGamepad ("button") # waits till you press at least one button
[0, 0, 1, 0, 1, 0, 0, 0]
>>> getGamepad (1, "button") # waits till ID 1 presses at least one button
[1, 0, 0, 0, 0, 0, 0, 0]
>>> getGamepad (range (getGamepad ("count")), "button")
# waits till someone presses a button
[[0, [1, 0, 0, 0, 0, 0, 0, 0]],
 [1, [0, 0, 0, 0, 0, 0, 0, 0]]]
>>> getGamepad ("button", "axis", wait=.01)
{"button": [1, 0, 0, 0, 0, 0, 0, 0],
 "axis": [-0.999969482421875, 0.0]}
(sometimes axis doesn't return exactly 1 or -1).
```

Here is a short, useful program:

```
>>> while 1: move (*getGamepad("robot"))
```

Here is a more functional one:

```
done = False
while not done:
    results = getGamepad ("button", "robot")
    move (*results["robot"])
    if results["button"][1]: beep (.5, 440)
    if results["button"][2]: beep (.5, 880)
    done =  (results["button"][0] == 1)
```

Gamepad Details

1. getGamepad ("count") - returns (immediately) the number of gamepads connected
2. getGamepad (ID, ITEM, ...) - return the ITEMs for gamepad ID. ID can be left out and will default to 0, the first one. If you request more than one ITEM, then they come back in a dictionary. Just request one ITEM and you'll get the value (as a list, string, or number).
3. getGamepad ([ID1, ID2...], ITEM, ...) - return the ITEMs for gamepad IDs as a list of lists of ID, RESULTS. For example:

   ```
   >>> getGamepad ([0, 1], "button", "axis")
   [[0, {'button': [0, 0, 0, 0, 0, 0, 0, 0], 'axis':
   [0.0, 1.0]}], [1, {'button': [1, 1, 0, 0, 0, 0, 0, 0],
   'axis': [-1.0, -1.0]}]]
   >>> getGamepad ([0, 1], "axis")
   [[0, [0.0, 1.0]], [1, [-1.0, -1.0]]]
   ```

4. getGamepad ( ) has one keyword argument, "wait" for setting a sleep value between gamepad polls. Default is 0.05 seconds. Setting to zero will have small latency, but may eat up your CPU.

ITEM can be:

1. "count" - returns (immediately) number of gamepads plugged in
2. "robot" - returns axis values (floats) ready to be used by move ( ) Myro command as [translate, rotate]
3. empty - if nothing is passed in, it returns all of the following as a dictionary
4. "init" - has this gamepad been initialized? as boolean
5. "name" - name of gamepad, as string
6. "axis" - returns values of axis controller in a list (as floats)
7. "ball" - returns values of ball controller in a list
8. "button" - returns values of buttons in a list (as integers)
9. "hat" - returns values of hat in a list

Here is a short little demo of how you could write a multi-player "game" where each player has a gamepad controller and appears as a circle on the screen.

```
from myro import *
def game ( ):
    win = GraphWin ("My Game!", 500, 500)
    numplayers = getGamepad ("count")
    colors = ['red', 'blue', 'green', 'yellow', 'orange']
    players = []
    # Create the players:
    for p in range (numplayers):
        circle = Circle (Point(randomNumber ( ) * 500,
                               randomNumber ( ) * 500), 10)
        circle.setFill (colors[p])
        players.append (circle)
        players[-1].draw (win)
    # Let's play!
    speak ("Red is it! Don't let red touch you!")
    while True:
        for  (id, data) in
getGamepadNow(range(numplayers),"axis","button"):
            players[id].move(data["axis"][0] * 20,
                             data["axis"][1] * 20)
            if data["button"][0]: # fire a missile
                computer.beep(.1, 440 * 2 ** id)
        wait(.1)
```

The window produced with the show (picture) command allows mouse clicks. The clicks will display the (x,y) and (r,g,b) in a status bar. Drag a rectangle to set YUV ranges for blob-tracking.

getPixels ( ) returns a generator object. Therefore, you cannot get a pixel by index.

The Myweb webpages give some notes. The images that a user has will be available when he/she edits their page. Images are placed in /myweb/data/robotname/*.jpg. Therefore one can:

```
sendPicture (takePicture ( ), "my-house", "PassWoRDz")
# assumes robot is connected, to take picture and get name
sendPicture (takePicture ( ), "my-house", "PassWoRDz",
"kitty") # you can provide the name explicitly, also
```

and refer to it picture with:

```
<img src="/myweb/data/kitty/my-house.jpg">
```

Arrays

The Array ( ) class, and makeArray ( ) function:

```
a = Array ( ) # zero dimensions
a = Array (10) # one dimension, 10 elements long
>>> a[9]
0
>>> a = Array (2, 3, 4) # 3 dimensions
>>> a[0][0][0]
0
>>> a[0][0][0] = 8
>>> a[0][0][0]
8
>>> a
<myro.Array object at 0x0160AC10>
```

makeArray can also take a Picture:

```
>>> p = makePicture (10, 10)
>>> array = makeArray (p)
>>> array[0]
<myro.Column object at 0x0161BB90>
>>> array[0][0]
<Pixel instance (r=0, g=0, b=0) at (0, 0)>
>>> array[0][1]
<Pixel instance (r=0, g=0, b=0) at (0, 1)>
>>> array[1][1]
<Pixel instance (r=0, g=0, b=0) at (1, 1)>
>>> setColor (array[1][1], Color (100, 50, 25))
>>> array[1][1]
<Pixel instance (r=100, g=50, b=25) at (1, 1)>
```

System Commands

To upgrade your Myro program files, you can issue the command:

```
>>> upgrade ("myro")
```

You must have write permissions for your Python installation (and other areas) for this to work. Afterwards, please restart Python.

```
>>> upgrade ("fluke")
>>> upgrade ("scribbler")
```

Robot Object Interface

robot = **Scribbler (port)** - robot constructor for the Scribbler; may ask for a port. You may also provide a name, which causes Myro to search the ports for such a robot.

```
>>> robot = Scribbler ( )
>>> robot = Scribbler ("Scribby")
>>> robot = Scribbler ("com4")
```

robot = **Surveyor (port)** - real robot constructor for the SRV-1; may ask for a port (this is under development)

robot = **Create (port)** - real robot constructor for iRobot's Create; may ask for a port (this is under development)

robot = **Roomba (port)** - real robot constructor for iRobot's Roomba; may ask for a port (this is under development)

robot = **SimScribbler ( )** - simulator constructor (this is currently under development)

Computer Object

The computer object is automatically created. In addition to the robot, the computer can also make sounds.

computer.speak (message) - computer will say the text message

computer.stopSpeaking ( ) - stop talking

computer.setVoice (name) - set the voice to a named voice

computer.getVoice ( ) - get the name of the current voice

computer.getVoices ( ) - get alternative set of named voices

computer.playSpeech (filename) - play a speech file (wav file)

computer.saveSpeech (message, filename) - save speech to a wav file

computer.beep (duration, frequency, [frequency2]) - makes a sound from computer. Duration is given in seconds (floats are ok).

computer.playSong (song) - play a song in the Song File Format

Instant Messaging Interface

You can send and receive messages from other Myro users.

```
>>> chat = Chat ("myname", "mypassword")
>>> chat.send ("somebodyelse", "Hi, how are you?")
>>> chat.receive ( )
[("sombodyelse@myro.roboteducation.org", "I'm fine, thanks!")
```

Remote Robot Control

The robot that will be controlled:

```
>>> robot.initializeRemoteControl ("mypassword")
>>> robot.processRemoteControl ( )
>>> []
>>> robot.processRemoteControlLoop ( ) # threaded, infinite loop
>>>
```

The computer that will be the controller:

```
>>> chat = Chat ("myname", "mypassword")
>>> chat.send ("remoterobotname", "robot.turnLeft (.4)")
>>> chat.receive ( )
```

There is also a RemoteRobot constructor which acts like a regular robot, but sends the commands to the other robot.

```
>>> robot = RemoteRobot ("remoterobotname")
>>> robot.turnLeft (.4)
```

Robot's Orientation

If you are using a bluetooth-serial adapter or a serial cable to control your robot, then the scribbler's normal forward direction is used. When using the Fluke the forward direction is flipped. "Forward" is in the direction of the Fluke's camera. However, the orientation of the scribbler can be manually changed using the setForwardness ( ) function. This is particularly useful if you want to use the Scribbler's IR and light sensors.

setForwardness (orientation): orientation can be 0/"scribbler-forward" or 1/"fluke-forward"

getForwardness ( ): Returns the orientation of the robot "scribbler-forward" or "fluke-forward"

Regulatory Information

FCC ID: VPU-4970726505

THIS DEVICE COMPLIES WITH PART 15 OF THE FCC RULES.
OPERATION IS SUBJECT TO THE FOLLOWING TWO CONDITIONS.
(1) THE DEVICE MAY NOT CAUSE HARMFUL INTERFERENCE, AND
(2) THIS DEVICE MUST ACCEPT ANY INTERFERENCE RECEIVED,
INCLUDING INTERFERENCE THAT MAY CAUSE UNDESIRED OPERATION.

NOTE: This equipment has been tested and found to comply with the limits for a Class B digital device, pursuant to Part 15 of the FCC Rules. These limits are designed to provide reasonable protection against harmful interference in a residential installation. This equipment generates, uses, and can radiate frequency energy and, if not installed and used in accordance with the instructions, may cause harmful interference to radio communications. However, there is no guarantee that interference will not occur in a particular installation. If this equipment does cause harmful interference to radio or television reception, which can be determined by turning the equipment off and on, the user is encouraged to try to correct the interference by one or more of the following measures:

- Reorient or relocate the receiving antenna.
- Increase the separation between the equipment and receiver.
- Connect the equipment into an outlet on a circuit different from that to which the receiver is connected.
- Consult the dealer or an experienced radio/TV technician for help.

RF Exposure

This equipment complies with FCC radiation exposure limits set forth for an uncontrolled environment. This equipment should be installed and operated with minimum distance 20cm between the radiator and your body. This transmitter must not be co-located or operating in conjunction with any other antenna or transmitter.

NOTE: THE MANUFACTURER IS NOT RESPONSIBLE FOR ANY RADIO OR TV INTERFERENCE CAUSED BY UNAUTHORIZED MODIFICATIONS TO THIS EQUIPMENT. SUCH MODIFICATIONS COULD VOID THE USER'S AUTHORITY TO OPERATE THE EQUIPMENT.