Raspberry Pi Robots & Imaging Processing

by Aviral Pandey, Rohan Wagle and Barry Brouillette

Latest Tech at the Robofest 2013 Competition

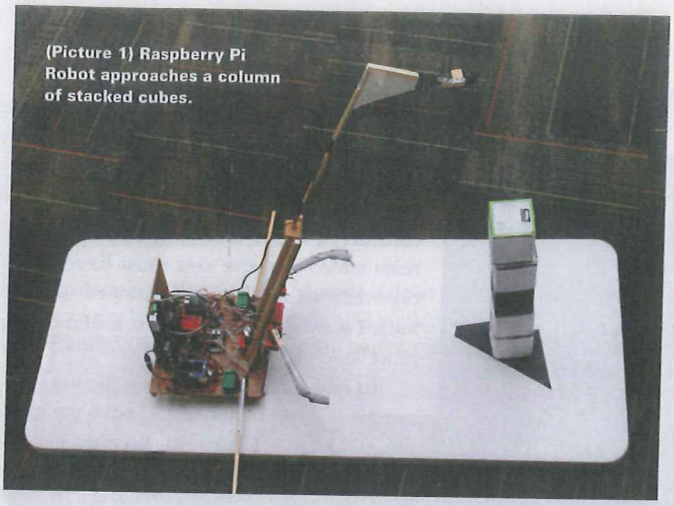
Robofest (www.robofest.net) is an annual robotics competition organized by Lawrence Tech University, Southfield, Michigan. It usually consists of three different events: the exhibition, VCRC (Vision Centric Robot Challenge) and the Game. The exhibition is for middle and high school students where they build and present robots to do a specific task or solve a problem of their choosing. The VCRC is an advanced level competition involving robots navigating an obstacle course largely through vision. The Game is for middle school and high school students, where robots complete a set of tasks autonomously. This year we participated in the Game but used some advanced image processing techniques using Raspberry PI.

2013 ROBOFEST GAME

This year's Game was called SRCC: Search, Rescue, Cleanup and Collect data. The playing field consisted of a set of boxes, one black and some white, placed somewhere on the table on top of a black right triangle. The challenge was to locate the boxes (search), grab the black box (rescue), remove the white boxes from the table (cleanup) and measure the area of the black triangle (collect data) in square millimeters.

In contrast to other robotics competitions and previous Robofest Game challenges, the discerning factor between two teams would be their accuracy, not their speed, which is the norm. As long as the challenge was completed within the allotted two minutes, the time which our robot took wouldn't matter. The accuracy of our measurement would account for a large part of our score. Therefore, a slow but extremely accurate robot was better than a fast but not-so-accurate robot.

(Picture 2) SRCC playing field.



(Picture 1) Raspberry Pi Robot approaches a column of stacked cubes.

This provided the perfect opportunity for us to use image processing. Although this process would be slower than other techniques of just having the robot drive around, its accuracy had the potential to be many times greater.

SOLUTION 1: NXT MEET RASPBERRY PI

The standard robot controller used in Robofest is the Mindstorms NXT. It's quite simple to program and provides many interfaces to motors and sensors. However connecting a webcam and using the NXT for image processing was impossible. We then came upon the Raspberry Pi (www.raspberypi.org). A $35 Linux box running an ARM processor, it was powerful enough to use for basic image processing but small and light enough to mount on a small robot. The Raspberry Pi's official language is Python, which is simple to program yet powerful. Our robot design contained both, communicating with each other.

The Raspberry Pi has a serial port and I2C capabilities but only as a master. The NXT lacks a simple serial interface, but has I2C with the master constraint. To get around this, we used a I2C-to-serial converter chip, the SC16IS750 (Sparkfun has a nice breakout board, part #BOB-09981). The Pi has a simple serial library, which only requires that you know the port of the serial connection. Figure 1 shows an example of serial connection. Figures 2-1 and 2-2 show it combination with an NXT.

```
#import the library
import serial

#initialize with the constructor
#The '/dev/ttyAMA0' changes based
#on your connection.
ser = serial.Serial('/dev/ttyAMA0', 9600, timeout=1)
#The commands work like simple
#console writes
ser.write("Hello World")
```

(Figure 1) Example of Serial Connection code in Python.

```
// NXT code for NXT and RPi Communication
#include "RoboHawk_Drivers2.c"
// You can find the above file at www.robofest.net/2013/RoboHawksNXT.zip
      SpecifyRobohawkPort(S1);
      char *cp;
      int i;
      int c=65;
      char s[80];
      char command[65];
      UartInit(); // Initialize the SC16IS750 I2C to Serial chip
 SendCharsSerial("GO");
      while(1)   // Endless loop waiting for messages
      {
            cp = GetIncoming(command); // check for message.
                                       // If none, command will be empty string
            i=strlen(command);  // If we've got any characters
                                // then we have a complete message
            s[0] = c;     // build up a simple message like AA
            s[1] = c;
            s[2] = 0;     // put a terminating null character at the end of it
            c++;     // increment c so that next message will be 'BB' and then 'CC'
            if (c > 'Z') c='A'; // Once we've used 'ZZ' start over with 'AA' again
            nxtDisplayBigTextLine(4, "%s", s);
            SendCharsSerial(s );
            cp = GetIncoming(command); // check for message.
                                       // If none, command will be empty string
            i=strlen(command);  // If we've got any characters
                                // then we have a complete message
            while(i == 0) {
                  cp = GetIncoming(command);
                  i=strlen(command);
            }
            nxtDisplayBigTextLine(6, "%s", command);
            wait1Msec(500); // Wait so you can see what's happening
      }
```

(Figure 2-1) NXT code example for NXT and Raspberry Pi communication.

(Figure 2-2) Raspberry Pi code example for NXT and Raspberry Pi communication.

```
# Raspberry Pi code for NXT and Raspberry Pi communication
import serial
ser = serial.Serial('/dev/ttyAMA0', 9600, timeout = 1)
inbuffer = " "
def check_for_message():
     global inbuffer
     while 1:
          newchar = ser.read(1)
          if (newchar == ""):          # If nothing was received then return
               return ""               # return here if message is not yet complete
          if newchar == ">":           # If we get a ">" character
                                       # we know that this is
             inbuffer = ""             # the start of a new message so clear inbuffer
          if newchar == "\r":          # If a carriage return was received
                                       # send back message
               inbuffer_copy=inbuffer # copy inbuffer so that we can
                                       # clear out the global var
               inbuffer=""             # clear out inbuffer so it's ready
                                       # for next message
               return inbuffer_copy    # return here if message is complete
          else:
               if newchar != "\n":     # ignore linefeeds
                    inbuffer += newchar # If any other character add to inbuffer
while true:
     output = check_for_message()
     if(output != " "):
      ser.write(output)
```

Capturing and Converting Images

```
fswebcam --jpeg 95 -r 1280x720 --no-banner --save /path/image.jpg
convert /path/image.jpg -colorspace Grey /path/image_bw.jpg
```

(Figure 3) Capturing and converting images.

IMAGE PROCESSING

The image processing program consists of about 450 lines (or 14kb) of Python code. While an attempt was originally made to use standard image processing techniques such as edge detection or line detection, we found these functions inadequate and created our own functions to process the image using just a simple get pixel type command found in Scikit image (http://scikit-image.org/docs/dev/api/skimage.html). The program can be broken down to four parts: image capture, image scanning, data analysis and result report.

The image capture sequence uses fswebcam (www.sanslogic.co.uk/fswebcam/) and ImageMagick (www.imagemagick.org), both free. Using command line arguments within our Python program via the Commands library, we have fswebcam take an HD picture with the robot's attached webcam, and save it to a preset directory. ImageMagick then converts the picture from RGB to grayscale. Since the triangle is black on white, converting to grayscale decreases the data overhead and makes processing much simpler. Figure 3 shows how to capture and convert an image in the command line.

The image scanning code looks at the grayscale image and attempts to find the edges of the triangle. The basic orientation of the triangle was given within the rules. Knowing this, we could separate the search for each leg into a different "scan." Each scan occurs from a different edge of the image and locates a different edge of the triangle. Figure 4 shows a basic scanner

The main component of each of the three scans is a set of coordinates, which is the current pixel the scanner is focusing on. Manipulating the current coordinates, the scanner can get data on any pixel desired, using a sequence of loops to maneuver the coordinate values around the image.

An image with 1280 x 720 resolution has 921,600 pixels to process through. However if the object of interest covers a large number of pixels (in our case the triangle), the scan can move to columns at intervals rather than attempt to process every column. The example program uses an interval of five.

The scanner works by using two for-loops, one nested within the other. The first for-loop will scan every tenth pixel along the bottom edge of the image, while the inner for-loop will scan every pixel above the one the outer loop is scanning, effectively scanning every fifth column of pixels in the image.

It is also necessary to check that you have a minimum number of pixels of interest. For example, shadows cast on the game field would confuse the program into thinking that there are more "black" areas than really exist.

If the scanner reaches the end of the column without finding any "interesting" (black) pixels, then it will just move on to the next column. If ten black pixels are found though, the program then stores, in a list, the coordinates where it first saw black pixels. This first list contains the "raw points" of the edges.

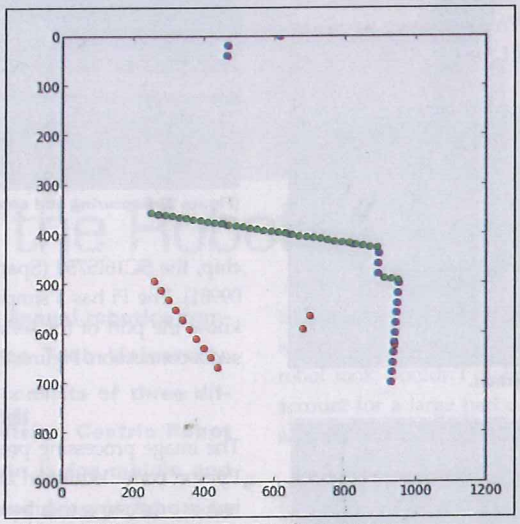
This scan can report edge points of a shape on a different background as long as the entire background is almost homogeneous. If not, this algorithm can be adapted to first search for the background then the shape of interest, as is shown in example program 4. We used a set of three scans to find the different sides of the triangle. In reality you can do a scan from any side, and even at angles with a little bit of modification.

The list returned by the scans is raw points, no lines yet. In order to create a line we used a point-to-line function. The idea was to select two start points that can be considered relatively likely to be accurate. Using these two points, a line can be extended. This line is a base to compare all other points of the list to. Points too far away from the line are discarded. Now we have lists of collinear points.

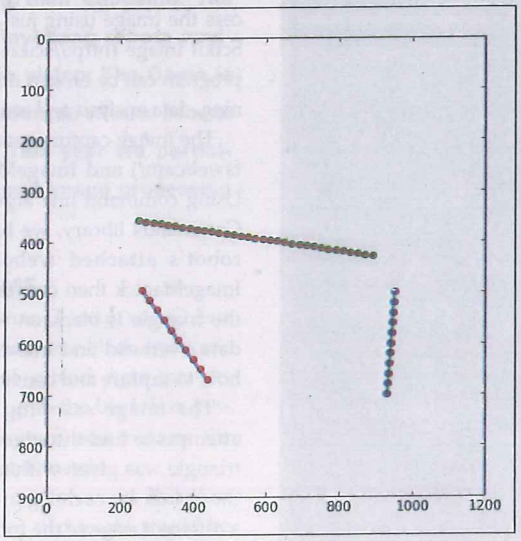
The first step in the data analysis sequence is a linear regression formula, which will take each of the three lists and formulate three linear equations. The points of intersection of these equations are calculated locating the corners of the triangle (or any other shape). Having the coordinates of the corners and using the distance formula, the program calculates the lengths of the sides. The largest line is assumed to be the hypotenuse and will be temporarily discarded while the smaller two legs will be considered legs x and y. The competition rules stated that the area must have been reported in square millimeters which meant a ratio from pixels to millimeters must be found.

```
# The following program shows how do a basic scan and
# reports the points in a list called points

points[]
for y in range(0, YMax-1, 10):
    x = XMax
    count = 0
    inBlack = False
    StartPoint = [0,0]
    x -= 1
    while(x>0):
        pix = image[x,y]
        if pix < BLACK_THRESH:
            if not inBlack:
                StartPoint = [x,y]
                inBlack = True
                count++
            else:
                count++
        else:
            if inBlack:
                inBlack = False
                count = 0
        if count > 5:
            points.append(StartPoint)
            break
        x -= 1

# note that maX and maxY are the x,y dimensions
# of the image respectively.
# Also note that the top of the image is y = 0,
# and the bottom is y = maxY
```

(Figure 4) A basic scanner.



(Figure 5) Raw output of points from a scan.

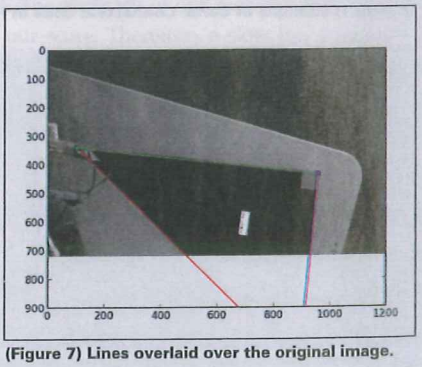


(Figure 6) A filtered list of points.

To find the ratio, we take a picture of a sample triangle which would have the millimeter lengths already written on it. We then run the program and have it output the leg lengths in pixels. Then we divide the known millimeter lengths by the calculated pixel lengths to find our ratio, which usually comes out to be .85 mm/pixel. This ratio can be stored as a constant and multiplied by the pixel lengths to find their millimeter lengths. Then the area formula for a triangle is completed using these lengths, and returned as a variable.

The result report sequence finishes off the image processing program by returning the calculated area to the program that called it. Originally the area would be sent to the NXT via serial connection, but later was switched to send the area back to a subroutine in the main program once we switched to using only a Raspberry Pi. Additionally, for debugging purposes the program can output each step it took on a graph, so we could visually see where the program may have gone wrong while we were writing it.

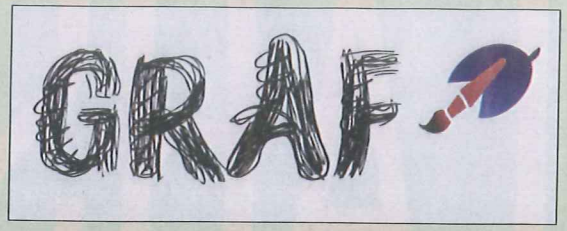
RASPBERRY PI ONLY SOLUTION:



(Figure 7) Lines overlaid over the original image.

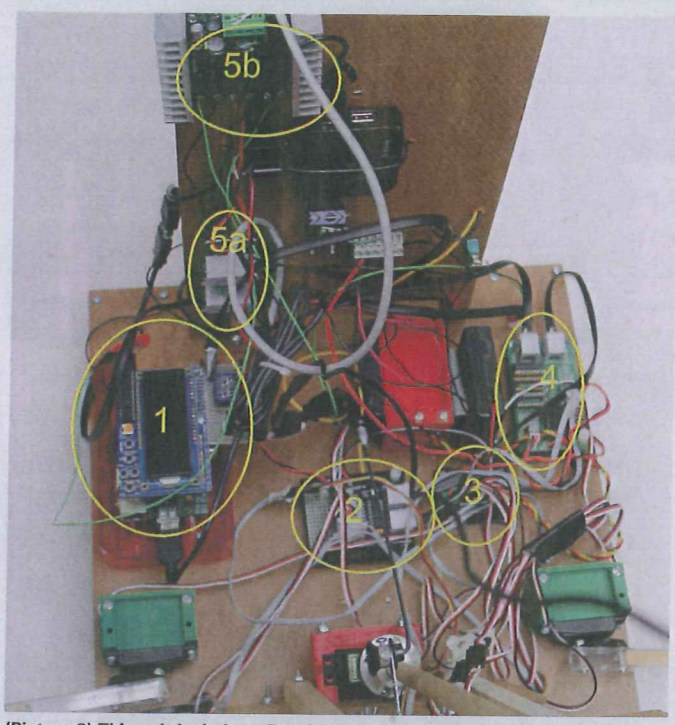
Some other teams complained that our original robot, having both a Mindstorms NXT and a Raspberry Pi, was against the contest rule of only one robot controller. While the Raspberry Pi wasn't really a robot controller, we were in a gray area. So for the world level competition we decided to use only the Raspberry Pi. Picture 3 shows the collection of boards that we used to give the Raspberry Pi comparable abilities to the NXT.

For use in robots, Raspberry Pi lacks many things. It does not have an ADC (ana-



GRAF, Lawrence Technological University
www.robofest.net

As a unique effort put forth by LTU Robofest, a new way to get students involved with STEAM (Science, Technology, Engineering, Arts and Mathematics) is with GRAF: Global Robotic Arts Festival. With the integration of arts and creativity, students are able to express themselves through intelligent and interactive robots with categories including robotics, dance, music and kinetic arts and sculptures. This new type of event is open to student teams from the 4th grade through college, and is open to hobbyists, enthusiasts and robotics companies throughout the world. The first GRAF event will be on November 23, 2013 at the Macomb Community College Expo Center in Michigan. Registration begins mid-August, 2013 at www.robofest.net.



(Picture 3) This unit includes a Raspberry Pi with Adafruit LCD char plate shield, a digital expansion board, servo controller board, analog expansion board, a motor board to read encoders and a motor board to apply PWMs.

(Figure 8) Reading values of an ADC in Python

```
# The following example uses the standard linux I2C library smbus
and an ADC connected to the raspberry pi via I2C using the PCF8591

#import and initialize smbus
import smbus
bus = smbus.SMBus(1)

#contains the I2C address of the PCF8591 chip
I2C_ADDRESS = 0x49

for x in range(0,3):
    #reads the ADC value from the specified channel
    output = bus.read_byte_data(I2C_ADDRESS, x)
    print str(output)
```

log to digital converter) to attach sensors to, nor does it have hardware PWM (pulse-width-modulation) for use with motors. It does however have a Serial, I2C and SPI (serial-peripheral interface). This allows us to use other boards which can read sensors and control motors and have said boards communicate with the Raspberry Pi. The example program in Figure 7 shows a basic I2C connection to an ADC (the PCF8591).

Even in a case such as this however, the Raspberry Pi cannot send signals at very accurate intervals. The reason for this is the Pi's official language: Python. The Python garbage collector makes it close to impossible to make sure that I2C messages occur at the right time. Although a way exists to write code at a lower level, such methods make using python for the image processing (a necessity) much more

(Picture 4) Innovation Award Trophy we received at the World Championship (Left to right: Dr. Lewis Walker, Chancellor of Lawrence Tech, Aviral Pandey and Rohan Wagle)

difficult. For our purposes, the difference didn't matter, but for high-speed applications where it is critical that the robot stops instantly, this can cause a problem.

The Raspberry Pi combined with our vision technique would usually report errors of about 0-5 percent in the area of the triangle in millimeters. Such a degree of accuracy shows just how versatile and powerful the Raspberry Pi is. Its simplicity to program and the relative ease with which it can be added to other controllers makes it a great tool to use in robotic competitions. Source codes of this project can be found at www.robofest.net/2013/RubberDuck.zip. NXT sources and the original Raspberry Pi source can be found at www.robofest.net/2013/RoboHawksNXT.zip. We received the Innovation Award at the World Championship (picture 4). ◎

Sponsors of the team
Mr. Ray Okonski
Mr. Barry Brouillette
Farmington Public schools

Acknowledgements
- *Harrison High School, Farmington Hills, Michigan, provided resources for the team*
- *Steve Dail, Harrison High School teacher*
- *The Brouillette family provided basement space to work on this project*
- *Math and Science Department, Lawrence Technological University (www.LTU.edu), provided summer work opportunities to wrap-up this project.*
- *Dr. CJ Chung, Professor of Computer Science at Lawrence Technological University and founder of Robofest, encouraged us to write and reviewed this article.*

About the authors: ***Barry Brouillette*** *is a Mentor, Harrison High School RoboHawks Robotics team.* ***Aviral Pandey*** *is a RoboHawks Robotics team member, Harrison High School, Farmington Hills, Michigan, aviral0607@gmail.com.* ***Rohan Wagle*** *is a RoboHawks Robotics team member, Harrison High School, Farmington Hills, Michigan, rohan.wagle@hotmail.com*

Links
Image Magick, imagemagick.org
Python, python.org
RaspberryPi, raspberrypi.org
RobotC, robotc.net
Robofest, robofest.net
Scikit, scikit-image.org
Sanslogic, sanslogic.co.uk/fswebcam/
Sparkfun, sparkfun.com

For more information, please see our source guide on page 81.

INSIDE CARNEGIE MELLON'S ROBOTICS LAB

BOTS LEARN TO HELP HUMANS!

ROBOT

THE LATEST IN HOBBY, SCIENCE [illegible] TICS

RAPIRO PERSONAL ROBOT

VEX WALL-E

ROBOFEST 2013

WILL ROBOTS SAVE US?

Page 32

ROBOTICS MAJOR[illegible] FOR [illegible]

[illegible] BRE[illegible]

FREE

To receive a FRE[illegible] complet[illegible] HOWIFY See yo[illegible]

AUTONOMOUS ROBOT CHALLENGE

Page 28

24 NEW ROBO PRODUCTS

Page 74

WWW.BOTMAG.COM
NOVEMBER/DECEMBER 2013
$5.99
12
0 71486 04748 3
DISPLAY UNTIL NOVEMBER 18, 2013