ENEE 699, Section 0301, Spring 2013

Autonomous Vehicle Control with Raspberry Pi

Pratik Mathur
University of Maryland, College Park
pratik@umd.edu

Introduction

Abstract

The objective of this paper is to (1) define the steps necessary for a Raspberry Pi to control a mobile robot and (2) describe the successes and failures encountered along the way. The steps outlined throughout the paper show what software was installed, how it was configured, and describe C/C++ and Arduino implementation needed to control an Arduino-based mobile robot from the Raspberry Pi.

Importance and Other Work

Autonomous vehicle control research is currently an important field of research and development in military and civilian context. The military can send robots into a hostile environment and safely go where humans cannot. Therefore autonomous vehicle control is the preferred method of fighting in the future. Autonomous vehicles can also be used in a civilian context. The following examples show how they can be used to track objects and guide vessels.

At this time of this writing there is continuing research on autonomous vehicle control. On December 9th 2012 a team of 4 sophomore students at Olin College of Engineering successfully built a cardboard quad-copter driven by the Raspberry Pi [1]. This team used OpenCV to process QR codes captured by a web camera. The team is currently working on tracking a moving QR Code that resides on a person or moving objects.

Another team known as FishPi is conducting ongoing research and development to create an autonomous boat driven by the Raspberry Pi that hopes to cross the Atlantic Ocean [2]. Both, the team at Olin College and FishPi team, are examples of research in the field of autonomous control but applied in unique ways. The team at Olin College must be careful to understand flight dynamics whereas the FishPi team must be aware of fluid dynamics.

My contribution

I was able to successfully use the Raspberry Pi to track a moving target and send control signals to a robotic land rover. I began this work in October 2012 under Dr. Blankenship's supervision and put together a simple proof-of-concept prototype in approximately 2 months. The engineering problems I ran into will be described in more detail throughout this paper.

Organization of this paper

The first section of this paper describes the technical problem. In order to ensure readers fully grasp the problem a short list of technical jargon is described followed by a general approach I took to solve this problem. The Raspberry Pi is formally introduced to the reader as a potential solution to the problem described in the previous section. Details on hardware and software setup follow. Once the reader knows how to get the Raspberry Pi functioning, the paper focuses on two methods of communication between the Raspberry Pi and Arduino. Both methods described in detail. The last major section discusses what code changes need to be made to control a mobile robot from the Raspberry Pi. This section describes the details of sample C/C++ code that was modified.

Problem

Problem Definition

Mobile robot object tracking requires a lightweight and power-efficient onboard computer processor that is capable of executing complex image-processing algorithms to guide a robot to follow a moving target. The ideal processor must be able to use these algorithms to identify a moving target and send commands to the mobile robot it is attached to. The importance of cost grows when the number mobile robots to be deployed increases, each carrying this processor. Therefore the ideal processor should be as inexpensive as possible. The power drawn by such a processor must also be carefully analyzed and cannot be ignored. The mobile robot must have enough power to carry the processor but also provide enough power for the processor to do useful work. Finding such a processor will make autonomous vehicle control a more practical problem to solve.

Technical Jargon

- Pixel – a sample of a continuous function and the smallest controllable element of a picture represented on a screen.
- Image – a 2-dimensional array of pixels
- Frame Rate – frequency at which an imaging device produces consecutive images
- Resolution – pixel height and width of a frame
- Processor speed - Clock cycles measured in MHz (Megahertz)
- Computer Memory – physical area on a computer used to store instructions or data. Measured in Megabytes (MB).
- SPI – Serial Peripheral Interface is a way of communication between physical devices.
- USB – Universal Serial Bus is a way of communication between physical devices that.
- GPU – Graphics Processing Unit is a specialized electronic circuit designed to rapidly manipulate and alter computer memory to accelerate the building of images in a frame buffer intended for output to a display.

Basics

In order to solve the defined problem above we must first experiment with computer processors available in the market today. One such candidate is the processor that comes with the Raspberry Pi [3]. If the Raspberry Pi's specification meets the requirements of the ideal processor, image-processing software must be installed and tested with an autonomous vehicle. In order to prove the Raspberry Pi is a strong candidate for autonomous vehicle control its performance must be tested. Computer algorithms must be implemented or modified in order to test computer vision on the Raspberry Pi.

The Raspberry Pi

Introduction

The Raspberry Pi is a credit-card sized computer that can plug into an HDMI-enabled TV or monitor. At $25 and 45 grams you get a computer running at 700 MHz and GPU capable of BluRay quality playback using H.264 decode. The graphics capabilities are roughly equivalent to an Xbox 1 performance [3].

Raspberry Pi Model A specification

Table 1 This information was provided from Raspberry Pi's Wikipedia page [5]

Processor	Broadcom BCM2835
CPU	700 MHz ARM1176JZF-S core (ARM11 Family)
GPU	Broadcom VideoCore IV
Memory	256 MB
Power Ratings	300 mA (1.5 W)
Weight	45 g (1.6 oz)
Operating Systems	Debian, Raspbian, Fedora, Arch Linux ARM, RISC OS, FreeBSD, Plan 9, Robot OS
Low-level peripherals	GPIO, UART, SPI

Installation

Operating System

Installation of the operating system is an important step when working with the Raspberry Pi. Installing an operating system on the Raspberry Pi is very different from installing one on a desktop or personal computer. Determining which operating system to install is equally important depends on what your goals are. The Raspbian Wheezy operating system is recommended as a good start by the Raspberry Pi foundation [5]. The following are the steps I followed to install this operating system on the Raspberry Pi.

Steps

1. Download the OS image file to a desktop or personal computer other than the Raspberry Pi and take note of its file path [6]
2. Copy this image to an SD Card of at least 8 GB. Please note, the following steps are what I followed running OS X Lion 10.7.1. Windows has different steps.
    a. Open a terminal and enter the following to become a super user: `su –`
    b. Insert your empty SD card into your card reader and enter: `diskutil list`
    c. Determine the identifier of your SD card drive (i.e. mine was /dev/disk1)
    d. Now that you know the identifier, unmount the driver: `diskutil unmountDisk /dev/disk1`
    e. Now copy the contents of the operating system image into the empty SD card: `dd if=<PATH TO YOUR OS IMAGE FILE> of=/dev/<YOUR SD CARD DRIVE IDENTIFIER>`
    f. Wait. This could take a while depending on the speed of your SD Card Reader (mine took 45 minutes).
    g. Once this copy completes the SD card should have a fully functional linux distro installed. The next step is to set up the hardware in order to boot up the Raspberry Pi

Hardware / Firmware setup

The Raspberry Pi must be properly powered and connected to physical interfaces such as a monitor, keyboard, and mouse in order to use it effectively. The following are steps I followed to get the Raspberry Pi up and running with an operating system.

Steps

1. Connect an HDMI-enabled monitor to the HDMI port of the Raspberry Pi
2. Connect a standard keyboard and mouse into the two USB ports on the Raspberry Pi.
3. Connect a USB-enabled web camera to a USB port on the pi.
4. If possible, connect an Ethernet cable to the network port (you will need Internet when updating the Raspbian Wheezy operating system).
5. Insert the SD card you installed Raspbian Wheezy on into the Raspberry Pi's SD Card drive (opposite side of the board).
6. Connect a micro-USB power supply to the Raspberry Pi's power supply port.
7. Once you are sure all wires are securely fastened into their ports plug the other end of the micro-USB power supply into a wall socket. This powers on the Pi.
8. If you completed steps 1-6 above correctly you should see a Raspberry image display on your monitor and you should boot into a terminal
9. Using your keyboard, enter: sudo raspi-config

10. You will see a number of configuration options such as configuring your time zone, enabling ssh, configuring your keyboard, etc. I would recommend selecting all configuration options hwoever for this paper configuring the keyboard and expanding the root partition to fill the SD card are most important.
11. Reboot by unplugging the power and plugging back in.
12. Once you see a terminal enter, update your operating system to ensure it is the latest version:
    a. Sudo apt-get install git-core
    b. sudo wget http://goo.gl/1BOfJ -O /usr/bin/rpi-update && sudo chmod +x /usr/bin/rpi-update
    c. sudo rpi-update
13. Update the firmware
    a. sudo rpi-update fab7796df0cf29f9563b507a59ce5b17d93e0390

Note on web camera drivers

In order for your web camera to properly function on the Raspberry Pi you must ensure the linux drivers are correctly installed. I used was the Logitech HD Pro C910. Fortunately, the UVC drivers came bundled with the Raspbian Wheezy operating system [7].

OpenCV

The next step is to install the image-processing package used to track a moving object: OpenCV. At the time this paper was written the latest version of OpenCV is 2.4.4a [8]. Due to memory constraints, the Raspberry Pi could not handle compilation of latest version of OpenCV from source code and threw performance error as shown in Figure 1.

```
[61% built target opencv_perf_gpu]

[61% building CXX object modules/gpu/CMakeFiles/opencv_test_gpu.dir/test/test_core.cpp.o]

c++: internal compiler error: Killed (program cclplus)

Please submit a full bug report,

with preprocessed source if appropriate.

See <file:///usr/share/doc/gcc-4.6/README.Bugs> for instructions.

make[2]: *** [modules/gpu/CMakeFiles/opencv_test_gpu.dir/test/test_core.cpp.o] Error 4

make[1]: *** [modules/gpu/CMakeFiles/opencv_test_gpu.dir/all] Error 2

make: *** [all] Error 2
```

Figure 1 – Error message when compiling OpenCV 2.4.4a on Raspberry Pi

I attempted to follow steps to overclock the Raspberry Pi to 800MHz [5] hoping it would give enough clock cycles to gcc compiler. Unfortunately I received the same error message shown in Figure 1 despite the clock speed being increased. The only other option was to install a version of OpenCV documented to work: OpenCV 2.3.1[10]. The following is are steps I followed to install OpenCV on the Raspberry Pi:

1. Open the terminal on the Raspberry Pi
2. `sudo apt-get -y install build-essential cmake cmake-qt-gui pkg-config libpng12-0 libpng12-dev libpng++-dev libpng3 libpnglite-dev zlib1g-dbg zlib1g zlib1g-dev pngtools libtiff4-dev libtiff4 libtiffxx0c2 libtiff-tools`
3. `sudo apt-get -y install libjpeg8 libjpeg8-dev libjpeg8-dbg libjpeg-progs ffmpeg libavcodec-dev libavcodec53 libavformat53 libavformat-dev libgstreamer0.10-0-dbg libgstreamer0.10-0 libgstreamer0.10-dev libxine1-ffmpeg libxine-dev libxine1-bin libunicap2 libunicap2-dev libdc1394-22-dev libdc1394-22 libdc1394-utils swig libv4l-0 libv4l-dev python-numpy libpython2.6 python-dev python2.6-dev libgtk2.0-dev pkg-config`
4. `wget http://sourceforge.net/projects/opencvlibrary/files/opencv-unix/2.3.1/OpenCV-2.3.1a.tar.bz2/download`
5. `tar -xvjpf OpenCV-2.3.1a.tar.bz2; rm OpenCV-2.3.1.a.tar.bz2; cd OpenCV-2.3.1/; mkdir build; cd build;`
6. `cmake -D CMAKE_BUILD_TYPE=RELEASE -D CMAKE_INSTALL_PREFIX=/usr/local -D BUILD_PYTHON_SUPPORT=ON -D BUILD_EXAMPLES=ON ..`
7. `make; make install`
8. `Wait about 4 hours`
9. `sudo nano /etc/ld.so.conf.d/opencv.conf`
10. `Add "/usr/local/lib" the at the end of the file and save.`
11. `Sudo nano /etc/bash.bashrc`
12. `Add "PKG_CONFIG_PATH=$PKG_CONFIG_PATH:/usr/local/lib/pkgconfig export PKG_CONFIG_PATH" at the end of the file`
13. `cd ~/opencv/OpenCV-2.3.1/build/bin`
14. `./convexhull`
15. If you are seeing a GUI appear with colored lines being drawn then you have installed OpenCV correctly.

Once the operating system and image-processing software has been installed the next step is to prove the Raspberry Pi can communicate with a microcontroller.

Communication between Raspberry Pi and Arduino

The Arduino platform was the best microcontroller to start testing with because it has been around for a long time and we know how to control robots with it. The goal is to show the Raspberry Pi can send commands to an Arduino board that controls a land rover. I experimented with two methods of communication between the Raspberry Pi and Arduino:

1. SPI – unsuccessful
2. USB – successful

In this section I will describe in detail the experiments I conducted in order to get the Raspberry Pi and an Arduino Mega 2560 to communicate.

SPI Communication

Serial Peripheral Interface (SPI) is one method of communication between devices. Devices communicate in master-slave mode where the master initiates the process of sending a data frame and the slave devices receive it. The following steps outline how the Raspberry Pi can be wired to an Arduino Mega 2560 over SPI:

1. Find 4 standard jumper wires to be used in the following steps
2. Read through Arduino Mega 2560 documentation [11] and look for the SPI pins:
    a. 50 – MISO – Master-In-Slave-Out
    b. 51 – MOSI – Master-Out-Save-In
    c. 52 – SCK - Serial Clock
    d. 53 - SS – Slave Select
3. Using the diagram on the MitchTech website [11] I constructed a new diagram using the Arduino Mega 2560 board as shown in Figure 2 below:

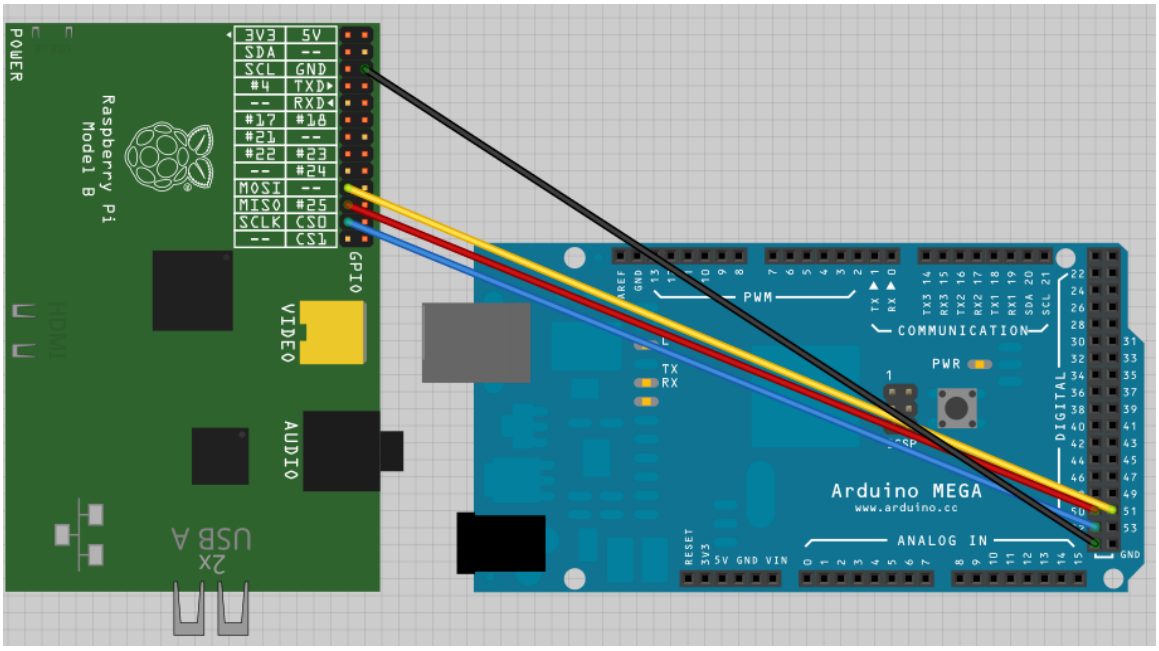

Figure 2 – SPI Communication between Raspberry Pi and Arduino Mega 2560 [Diagram created by Pratik Mathur using Fritzing [13] software tool

4. Confirm the wires are properly fastened to all ports
5. Navigate to the MitchTech website [11] and copy the code marked as "Arduino code" provided by Nick Gammon and flash it to the Arduino 2560
6. Power On the Raspberry Pi and open the terminal
7. Ensure the operating system has the required drivers that speak SPI [11]
    a. `modprobe spi_bcm2708`
    b. `modprobe spidev`

```
c. lsmod; // ensures SPI modules are loaded by the operating
   system
```

8. From the MitchTech website [11] copy the code marked as "Raspberry Pi Code" into the Raspberry Pi and create a C program called "spidev_test.c".
9. Compile this program using gcc in the terminal on your Raspberry Pi: gcc spidev_test.c –o spidev_test
10. Open the Serial Console on your Arudino (from a PC) that is connected to your Arudino
11. Ensure the baud rate is set to 115200.
12. On the raspberry pi terminal navigate back to where you compiled the spidev_test.c program and enter: ./spidev_test
13. If you did everything correctly you should see "HELLO WORLD" characters print out.

Troubleshooting SPI Communication

Instead of "HELLO WORLD" output, the Arduino's Serial Console received garbage data. Possible problems during communication are outlined:

1. Baud Rate mismatch between Serial Console and Arduino Code – I verified both rates were initially set to 115200
2. Baud Rate could be too high and needs to be lowered – I tried all possible baud rates ranging between 9600 – 115200. Testing this required modifying the Arudino code and reflashing. The only difference was the garbage data looked different but was still not the "HELLO WORLD" string I was looking for.
3. The MISO-MOSI connection were inverted – I tried to swap the MISO and MOSI connections however running the "spidev_test" program generated no output at all whereas previously the Arduino Serial Console was at least receiving garbage data. Since the Arduino was receiving no data it is unlikely that MISO-MOSI connections were inverted.
4. There is a protocol synchronization problem between the Arduino and the PC it is connected to. It was unclear how exactly to debug a protocol synchronization problem.

After considering the possible problems described above, I was unable to pinpoint where the problem was with SPI communication.

USB Communication

Universal Serial Bus (USB) is a standard used for communication between and powering of devices. Since the Raspberry Pi and Arduino Mega 2560 both support USB this was a good alternative to SPI communication. I initially experimented with a Python library to initiate a transfer of a simple command from the Raspberry Pi to the Arduino. The following are steps I took during the experiment and are based off of Dr. Monk's DIY Electronics Blog website [14]:

Python Library

1. Connect a USB cable from the Arduino to your desktop PC (not the Raspberry Pi).
2. Navigate to Dr. Monk's blog and flash the code under the "Arduino" section to your Arudino
3. Unplug the Arduino from your PC and connect it to one of the USB ports on the Raspberry Pi
4. Power on the Raspberry Pi (this should also give power to your Arduino).
5. Once you have logged into the operating system open the terminal
6. wget http://sourceforge.net/projects/pyserial/files/pyserial/2.5/pyserial-2.5.tar.gz/download
7. Navigate to the directory where you downloaded the Py-Serial library.
8. Gunzip pyserial-2.5.tar.gz; tar –xvf pyserial-2.5.tar;
9. cd pyserial-2.5; sudo python setup.py install; // this install Py Serial library on the Raspberry Pi
10. ls /dev/tty*
11. You should see a name for your Raspberry Pi's USB interface (i.e. something like /dev/ttyACM0). Keep a note of this interface name.
12. Open the Python2 console from the Raspberry Pi's menu (it is also referred to as ScratchPad)
13. From within the Python2 / ScratchPad console enter: import serial
14. ser = serial.Serial('/dev/ttyACM0', 9600);
15. while 1 : ser.readLine()
16. As soon as you enter step 15 in the Python console you will start to see messages from the Arduino. This proves the Arduino can send messages to the Raspberr Pi. However, we need to show the Raspberry Pi can send messages back to the Arduino in order to control a robot.
17. Hit CRTL-C and break out of the infinite loop you started in step 15. You will see an error message, which is expected and normal.
18. From within the Python console enter "ser.write(5)".
19. You should see the Arduino LED light blink 5 times.
20. This shows the Raspberry Pi can send commands to the Arduino.

Note

The experiment above proved two-way communication works between the Raspberry Pi and Arduino over USB. The next step is experiment with object-tracking algorithms that come bundled with the OpenCV package. One of these algorithms was written in C. Invoking Python commands from a C program requires taking extra steps that allow a C program to communicate with Python libraries. This can become more complicated that necessary. Instead, I decided to experiment with the Serial Programming Guide for POSIX Operating Systems.

Serial Programming in C

The Serial Programming Guide for POSIX Operating systems provided information on how a C program can send signals over the USB protocol [15]. I wrote a simple C program to verify this. Lines 1-58 shown in the sample code below proves a Raspberry Pi C program can communicate with an Arduino.

```
1   //  Created by Pratik Mathur
2   //  Copyright (c) 2012 University of Maryland, College Park. All rights
3   reserved.
4   // THIS PROGRAM IS A PROOF OF CONCEPT OF USB COMMUNICATION BETWEEN THE
5   RASPBERRY PI AND AN ARDUINO.
6   //
7   #include <termios.h> /* POSIX terminal control definitions */
8   #include <string.h>  /* String function definitions */
9   #include <unistd.h>  /* UNIX standard function definitions */
10  #include <fcntl.h>   /* File control definitions */
11  #include <errno.h>   /* Error number definitions */
12  #include <ctype.h>
13  #include <stdio.h>
14
15  /*
16   * 'open_port()' - Open serial port 1.
17   *
18   * Returns the file descriptor on success or -1 on error.
19   */
20
21  int open_port(void) {
22      int fd; /* File descriptor for the port */
23
24      /* The first string in open() represents a device ID specific to your
25  machine.  Make sure replace it with your USB device's ID! */
26      fd = open("/dev/tty.usbmodem1a21", O_RDWR | O_NOCTTY | O_NDELAY);
27      if (fd == -1) {
28          /*
29           * Could not open the port.
30           */
31
32          perror("open_port: Unable to open /dev/cu.usbmodem1d11 - ");
33      } else
34          fcntl(fd, F_SETFL, 0);
35
36      return (fd);
37  }
38
39
40  int main(int argc, const char * argv[])
41  {
42      // open a connection to the arduino over USB
43      int fd=-1;
44
45      // store the file descriptor for use when writing
46      fd = open_port();
47
48      // write 1 byte to the USB port of the Arudino
49      int n = write(fd, "R",1);
50
51      // if the write was unsuccessful it will return a value less than 0
52      if (n < 0)
53          fputs("write() of 1 byte (R) failed!\n", stderr);
54
55      // return success
56      return 0;
57  }
58
```

This program successfully wrote the character "R" to the USB port of the Arduino. The next step is to understand algorithms that track a moving object.

Object-tracking Algorithms that use Open CV

There are two algorithms I inspected that used OpenCV to performed object-tracking

1. cvBlob [16]
2. camshiftdemo [17]

cvBlob

The cvBlob algorithm is composed of a number of modules shown below in Figure 3.

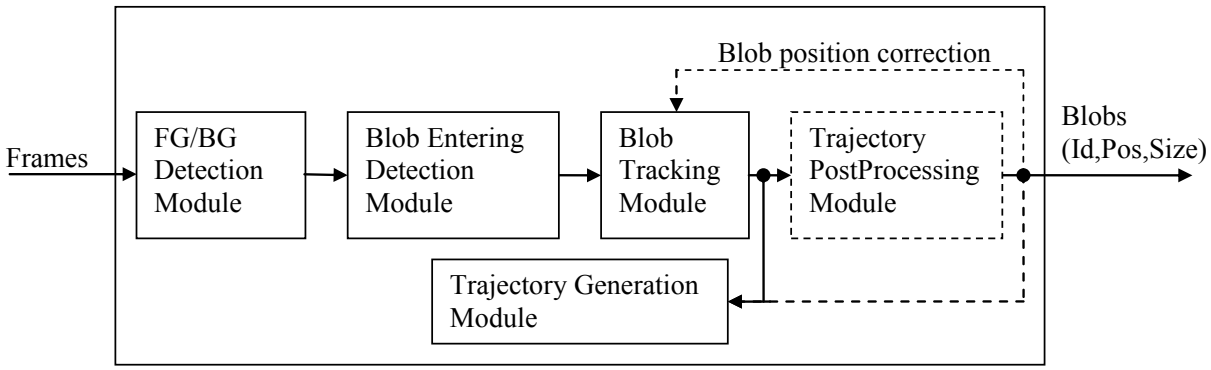

Figure 3 – flow diagram of cvBlob algorithm

Each module performs a specific function on camera frames passed in and processes them. The frames are modified and passed along the module chain. Once reaching the end of the chain, the blob's position and size are outputted. More details about how the algorithm works can be found on the cvBlob website. I download the source code available from the Google Code website for cvBlob and compiled "red_object_tracking.cpp" on the Raspberry Pi [18]. As soon as the compiled program is executed it displays a capture feed of the web camera attached to the Raspberry Pi. Within the first 5-10 seconds of execution the video capture froze. The frames were no longer updating on the screen. If there were a software bug in the program I would have expected a segmentation fault (or Core Dumped file message). However there was no such error or core file and the program continued to execute (i.e. it did not crash).

Investigation of Cause of "freeze" in Red Object Tracking algorithm

I decided to investigate what caused the "red_object_tracking" program to freeze on the Raspberry Pi. I was initially suspicious of two possible problems:

1. There is a software bug in the cvBlob source code

2. The cvBlob source code was written in such a way that is optimized for an Intel-based architecture, not ARM (which is the architecture used by Raspberry Pi).

To see if the if #1 was the problem I downloaded and compiled the cvBlob source code on my Machintosh computer. When I executed the red_object_tracking program on the Macintosh it ran without any problems. The video frames continued to capture updates and a red object I was holding in my hand was tracked accurately as I moved it around. This proved the problem was not in the algorithm or code.

There was a possibility of #2 above if the cvBlob C++ code was written and optimized for an Intel-based machine rather than ARM so I was suspicious there was something in the code that caused it to freeze on the Raspberry Pi. I decided to put print statements in different areas of the "red_object_tracking.cpp" code. I noticed the problem happened as soon as it invoked the "cvLabel(segmentated, labelImg, blobs);" on line 67. This function was defined in another C++ program called "cvLabel.cpp" [19]. I proceeded to place print statements in "cvLabel.cpp" in order to pinpoint exactly where in the function it froze on the Raspberry Pi. After much debugging, it revealed the problem occurred on line 97 of "cvLabel.cpp". More specifically, the code was attempting to loop image_height x image_width times (i.e. for a frame that a 480 x 640 it would loop 307, 200 times) just for one frame. The next incoming frame would require another 307, 200 of looping again. This was not a problem for my Macintosh running at 3.2GHz with 6GB of RAM. However the Raspberry Pi appears to be struggling with this kind of processing since it is only 700 MHz with 256MB of RAM (almost 5 times less processing power and 23 times less main memory). I knew this was the problem because the Raspberry Pi appeared to be stuck in the first loop iterating through each pixel in every frame.

After pinpointing where the problem was in the code I had two options: fix the problem or find another algorithm that works out of box. I went with the second option due to time constraints. As it turns out, this problem has been identified on the cvBlob website as Issue #23 – "Problem with signed char, cvLabel hangs" [22].

Camshiftdemo

I determined the "camshiftdemo" program did exactly what was needed [17]. I downloaded and compiled the source code on the Raspberry Pi. The frame rate of camshiftdemo is slower than cvBlob, however, the program was successfully able to track an object of my choosing. As I moved the object around in front of the web camera the red-colored ellipse followed it a few seconds later. Figure 4 shows an image taken while the program is running as it tracks an orange-colored bottle cap:

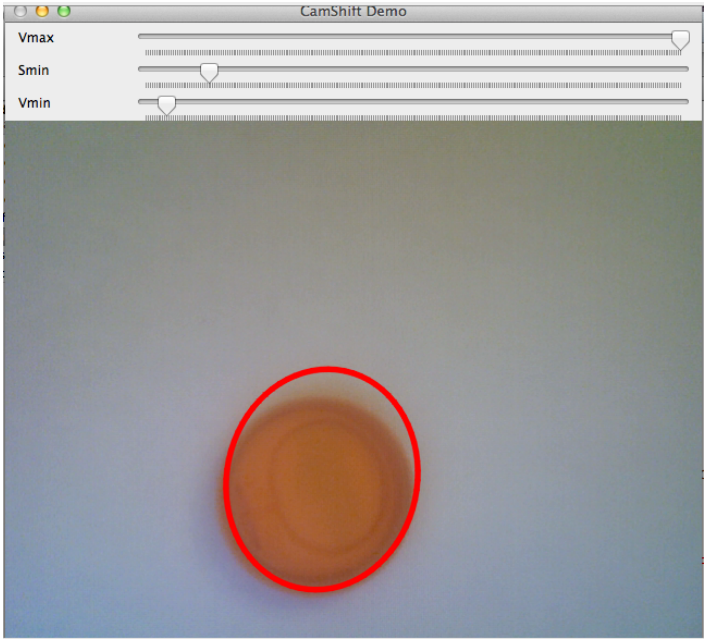

Figure 4 – orange bottle cap discovered by OpenCV "camshiftdemo" program

In Figure 5 below you can see the object is followed when moved:

Figure 5 – orange bottle cap moved to a different position and the camshiftdemo followed its movement

Notice the ellipse widens after the move. This shows the algorithm is not very accurate even under well-lit setting. Compared to cvBlob algorithm the camshiftdemo algorithm requires much less processing power and is a good candidate the Raspberry Pi. I decided to understand how this algorithm works so it can be modified to send a robot command strings.

Camshiftdemo Pseudocode

The following pseudocode is to give the reader an idea of what camshiftdemo code is doing out of box [17]. I have intentionally left out details that are not relevant to this project in order to help facilitate understanding the program will need to be made.

```
1   // loop infinitely
2   while (true)
3
4       // capture web camera frame
5       cap = getNextFrame();
6
7       // if frame has data and an object has been selected to be tracked
8       if (!cap.isEmpty() AND trackObject==TRUE)
9
10          // convert the selected region to be tracked into a rectangle
11          Rect selection = getRectangle (TopLeftPoint, BottomRightPoint);
12
13          // mask everything beyond the selected region and //return as matrix
14          Matrix maskedSelection = maskSelectedRegion(zerosMatrix, selection);
15
16          // convert the selection into a rotated rectangle //
17          //using the CAMSHIFT algorithm
18          RotatedRect rRect = CAMSHIFT(maskedSelection);
19
20          // rRect from now on represents where the algorithm
21          //thinks the tracked object is.
22
23          // determine the center of the tracked object
24          Point center = rRect.getCenter();
25
26          // draw ellipse around object to-be-tracked
27          ellipse(center, rRect);
28
29      endif
30
31  end while
```

Figure 6 - General algorithm of camshiftdemo

Changes To Existing Camshiftdemo Code

Looking at the pseudocode above you will notice line 8 checks if a frame is captured and if the user has selected an object to track. From lines 11 – 18 the algorithm determines the current location of the tracked object. By the time we get to line 24 the algorithm has a good idea where the tracked object is located. After line 27 it is the perfect place to add code that creates instructions for the robot to follow and send the instructions to the robot via a USB connection. The following pseudocode is inserted after line 27:

```
1   // determine the area of ellipse drawn
2   area = ellipse.getArea();
3
4   // if the object has moved further away
5
6   if (area > IDEAL_AREA)
7       MOVE_FORWARD();
8   end if
9
10  if (area < IDEAL_AREA)
11      MOVE_BACKWARD();
12  end if
13
14  // get x coordinate of object's center
15  x = center.x;
16
17  // if object's center has moved left
18  if (x < screen width / 2)
19      TURN_LEFT();
20  end if
21
22  // if object's center has moved right
23  If (x > screen width / 2)
24      TURN_RIGHT();
25  end if
```

Figure 7 – Pseudocode constructing a simple control signal to be sent to the mobile robot. To view C++ code look a reference [20].

Lines 1-25 shown in the left in Figure 7 represent the general logic that was inserted into the camshiftdemo program. Line 2 determines the mathematical area of the ellipse drawn on line 27 in Figure 6. Using the area of the ellipse we can determine if the tracked object has moved closer to or further from the mobile robot. Lines 6-13 implement this logic. Lines 15 – 25 determine whether the mobile robot should turn left or turn right. The modified camshiftdemo code is checked into a subversion server [20].

It is important to discuss Arduino code that received commands from the Raspberry Pi [21]. The following pseudocode shows what the Arduino did when receiving data from the Rasperry Pi.

```
// if the Raspberry Pi has sent information over USB
if (Serial.available())
   // read in 1 byte
    char command = Serial.read()

   // if left signal
    if (command=='L')
        // invoke the left turn function
        turn_left()

    // if right signal
    else if (command=='R')
       // invoke the right turn function
        turn_right()

    // if move backward signal
    else if (command=='B')
        // invoke the move backward function
        move_backward()

    // if move forward signal
    else if (command=='F')
       // invoke the move forward function
        move_foward()
```

The algorithm shown above is divided into 4 simple cases:

1. turn left
2. turn right
3. move backward
4. move forward

Depending on what the Raspberry Pi sends the Arduino will invoke the appropriate function that causes a servo attached to it to move as desired.

Recommendations to improve performance

The Raspberry Pi does a good job tracking a slow-moving target. However, if the target moves too quickly there aren't enough cycles in the CPU to keep track of it. The following is a list of recommendations to improve performances of the raspberry pi when tracking objects:

1. Reduce the image to a monochrome sample
2. Reduce the range of samples (i.e. 8-bit monochrome to 4-bit monochrome)
3. Reduce the size of the image (i.e. 1024 x 768 to 64 x 64)
4. Reduce the frame rate (60 fps to 5 fps)
5. Turn off the web camera feed entirely
6. Check cvBlob website for changes to code baseline for any posts regarding a fix for ARMv6 architecture.

I used the Model A version of the Raspberry Pi however Model B has double the RAM of the Model A. This could give significant improvement in performance.

Conclusion

The work described in this paper is a proof-of-concept to show that a Raspberry Pi can be used to control mobile robots. As with any emerging technology there are many challenges that need to be addressed. However, I feel like the Raspberry Pi is a competitive player and has an advantage over its competition due to its low cost. As more operating systems and software packages start releasing optimized versions for the ARM architecture we will see significant improvements in performance.

References

1. http://www.instructables.com/id/Autonomous-Cardboard-Rasberry-Pi-Controlled-Quad/
2. http://www.instructables.com/id/Making-an-autonomous-boat-with-a-Raspberry-Pi-a-/
3. http://www.raspberrypi.org/
4. http://en.wikipedia.org/wiki/Raspberry_Pi#Specifications
5. http://www.raspberrypi.org/downloads
6. http://downloads.raspberrypi.org/images/raspbian/2013-02-09-wheezy-raspbian/2013-02-09-wheezy-raspbian.zip
7. https://github.com/raspberrypi/linux/tree/rpi-3.2.27/drivers/media/video/uvc
8. http://sourceforge.net/projects/opencvlibrary/files/
9. http://www.jeremymorgan.com/tutorials/raspberry-pi/how-to-overclock-raspberry-pi/
10. http://mitchtech.net/raspberry-pi-opencv/
11. http://mitchtech.net/raspberry-pi-arduino-spi/
12. http://www.jameco.com/Jameco/Products/ProdDS/2121113.pdf
13. http://fritzing.org
14. http://www.doctormonk.com/2012/04/raspberry-pi-and-arduino.html
15. http://www.easysw.com/~mike/serial/serial.html#2_5_3
16. http://code.google.com/p/cvblob/
17. https://code.ros.org/trac/opencv/browser/trunk/opencv/samples/c/camshiftdemo.c?rev=1429
18. http://cvblob.googlecode.com/svn/trunk/samples/red_object_tracking.cpp
19. http://cvblob.googlecode.com/svn/trunk/cvBlob/cvlabel.cpp
20. https://enee699-mathurpratik.svn.beanstalkapp.com/raspberrypi/trunk/camshiftdemo.cpp
21. https://enee699-mathurpratik.svn.beanstalkapp.com/raspberrypi/trunk/raspi_arduino/raspi_arduino.ino
22. http://code.google.com/p/cvblob/issues/detail?can=2&start=0&num=100&q=&colspec=ID%20Type%20Status%20Priority%20Milestone%20Owner%20Summary&groupby=&sort=&id=23