ME 461 Laboratory #6
The Beaglebone, Embedded Linux and I^{2}C communication

Goals:

1. Become familiar with the Linux command prompt or terminal
2. Use I2C to communicate between Linux running on the Beaglebone development board and a MSP430F2272 microcontroller.
3. Learn how to command a RC servo motor.
4. Get a small taste of the powerful vision library, OpenCV, and implement a basic blob search algorithm.

Exercise 1:

The goal of this first exercise is to get you familiar with working in the Linux at the "terminal" or "command prompt" level. By the end of this exercise we will start up the X-windows interface for Linux but still the bulk of your work will be inside the terminal.

Probably the two most important terminal commands you need to know are "cd" (change directory) and "ls" (list files). If you type "ls" and then enter, Linux will list the files located in your current directory. Commands:

ls Lists files and directories in current directory

ls –la Lists files and directories and includes size of each file and its save date.

cd /home/root/djblock Changes current directory to /home/root/djblock

cd By itself takes you back to your home directory. On Beaglebone /home/root

pwd "Print Working Directory" indicates your current directory

Here are two websites that give information on the most important Linux terminal commands.
http://community.linuxmint.com/tutorial/view/100
http://faculty.ucr.edu/~tgirke/Documents/UNIX/linux_manual.html

First let's work with Linux "Headless." "Headless" means not connecting a keyboard, mouse or monitor to the Beaglebone but just communicating and commanding Linux through an RS-232 connection or an Ethernet connection. Have your TA show you how to power and connect everything to your Beaglebone. First use the RS-232 as our headless connection. Open Tera Term and select "COM1" as the serial port and set its baud rate to 115200. Then when everything is cabled, power on your Beaglebone. In a few seconds you should see a large number of messages printing to the terminal. There are boot messages indicating the status of the boot steps. If errors occur during boot they would

be printed here. (Side Note: After Linux has booted you can list off all these boot messages by typing the command "dmesg" at the terminal.) After about 20 seconds or so Linux will complete its boot and the login prompt "beaglebone login:" should be printed. Sometimes an additional debug message is printed to the terminal after the login prompt is displayed. Simply press enter once and the login prompt will reprint. Login:

Username: **root**

Password: **<There is NO password, just press Enter>**

Now perform the below steps to get you a bit more familiar with Linux and the Beaglebone.

- Check and Change the Date. If we had the Beaglebone connected to the internet instead of the lab's local network it would know to find the date from a NTP (Network Time Protocol) server. Instead we will manually set the date. Type the following:
  - **date**
  - "**date –help**" and scroll to the top to see the beginning of help
  - **date 1001131413**, but change it to today's date. (Note the format is MMDDhhmmYY.)
- Create a Directory
  - **mkdir <your netid>**
  - **cd ~/<your netid>** Note: ~/ is your home directory /home/root
- Create a simple C file so we can play around with the gcc C compiler
  - **nano myCfile.c** (nano is a simple text editor)
  - Write this simple C program:

```
#include <stdio.h>

void main(void) {
      int i = 0;
      int count = 0;
      for ( i=0; i<20; i++ )  {
            printf("Count = %d\n",count);
            count++;
      }
}
```

  - To save your file in nano: **Ctrl-O** then **Enter**. To exit nano: **Ctrl-X**
  - **ls** to see that your file was created.

- **gcc –lm myCfile.c –o myCfile**
- **./myCfile** to run your application.
- Play around with the arrow up and down keys to see you can scroll through your command history. Also try Tab completion by typing "**nano myC<Tab>.<Tab>**". Also if you type "**nano m<Tab><Tab><Tab>**" Linux should list all files starting with "m".
- Now reedit your myCfile.c file and add some errors to your program. Recompile your file and notice how gcc lists of the line numbers of your errors.
- To see line numbers displayed at the bottom of nano use the –c option so: **nano –c myCfile.c** and notice the line number display and fix your errors.
- If you know "vi" or "vim" you can use those text editors also. I think "nano" is easier to use because many of its commands are always listed at the bottom.
- To get ready for the next bullet item type the command "**ifconfig**" (interface config for the internet connection). Scroll up to the "eth0" item and find your beaglebone's "inet addr", internet address. Should start with 192.168.1.???.

- The other "headless" connection to the Beaglebone is through an internet connection. Use Tera Term or Putty to connect to your Beaglebone over the network. For Tera Term select the menu item File->New Connection. Then in the window that opens select TCP/IP and type your Beaglebone's IP address in the Host Box. Leave all other options as the default and click OK. A SSH Authentication window will appear. Simply type the username "root" in the username box and leave the passphrase box empty because root does not have a password. Select OK and then a new terminal will appear and you are connected to your Beaglebone through a different interface. This terminal works identical to the RS-232 terminal connection except that you need to wait until the Beaglebone is fully booted before you try to connect over the internet. The Beaglebone needs to launch a SSH server before you can connect to it.

- To connect with Putty instead of Tera Term, open a Command Prompt (cmd) in Windows 7 and type "**putty root@192.168.1.???**" where ??? is your Beaglebone's remaining IP address. This then turns the command prompt into a terminal connected to your Beaglebone.

- From inside either of these internet terminals (Tera Term or Putty), use **cd** and **ls** to find the directories and files you created in the RS-232 terminal. Run the application that you compiled earlier. So with this internet terminal any computer/tablet/smartphone that is wired or wirelessly connected to the lab router can connect to your beaglebone.

- Now let's take a quick tour of the Linux running on the Beaglebone. The distribution of Linux running on the Beaglebone is Angstrom. It is a version of Linux specific for embedded systems. You can read more about Angstrom at http://www.angstrom-distribution.org/. Perform these steps to navigate through the Linux directory:
    - Type **pwd** to see what directory you are currently in.
    - **cd /** will take you to the start of the Linux file system. This is like typing cd c:\ in Windows.
    - **ls** to list all the files and directories at the start of the Linux file system. The bin and sbin directories have most of the common Linux commands. **cd /bin** and then **ls** and find the pwd, ls, ping, etc commands
    - **cd /sbin** and then **ls** and find the ifconfig, shutdown, etc commands
    - **cd /dev** This is where many of the devices connected to the beaglebone's processor are listed. For example find ttyS0. This is the serial port for the RS-232 terminal you used previously. Notice there are two i2c serial ports. We are going to use i2c-0. video0 is the USB camera connected to your beaglebone. Type **lsusb** to list all devices plugged into the USB.
    - **cd /etc** and **ls** This is where most of the boot setup scripts are located. For example **cat profile** lists the setup file for your terminal. You can change your default path for example by editing this file. Please do not change this file though!
    - **cd /usr** and **ls** This is where most of the installed programs are located. **ls /usr/bin** to see a large number of installed programs.
    - /sys has kernel related files. /mnt and /media are the normal locations for mount usb drives or any storage device. /lib stores most of the Linux runtime libraries. /var is where many Linux programs save program specific data during operation, like log data and printer queues.
    - **cd /proc** and **ls** The proc directory is full of virtual files. These are files that the linux kernel modifies and controls. You can list their contents and find out information about the system. **cat uptime** to see the amount of time Linux has been running. Run the command again to see that it changes. cat a few other items like cpuinfo or whatever you want.
- Practice copying files to/from Linux from/to Windows PC. First change back to your user ID directory. Couple ways to do that: Full path **cd /home/root/<yourDirectoryName>** or **cd ~/<yourDirectoryName>** or two commands **cd** to get back to /home/root and **cd <yourDirectoryName>** to change to your directory.

- **mkdir testcopying** and then **cd testcopying**
- Now switch to your Windows computer and open a terminal window. Type **n:** and then **cd n:\me461\Fall14\Lab6.** Type **dir** and you will see the file **create_a_file.c**. We want to copy this file down to the Beaglebone.
- To copy the file to your beaglebone you need to know its ip address again. (If you do not remember it from the above exercise, back at the Linux prompt type **ifconfig** and find its ip address under eth0. It will be 192.168.1.???.) Then go back to your Windows command prompt and type **pscp create_a_file.c root@192.168.1.???:/home/root/<yourDirectoryName>/testcopying/.** This takes a few seconds to perform the copy. It will also ask you for the Linux password which is no password so just press enter. Now go back to the Linux terminal and list the contents of your testcopying directory and you show see create_a_file.c there.
- Edit create_a_file.c with nano and you will see that it is a very simple C program that opens a file named "mydata.m" and creates and writes "M-file" formatted data file using the fprintf() function.
- Compile create_a_file.c at the Linux terminal by typing **gcc –lm –o create_a_file create_a_file.c** (the –lm option links in the math library) and then run the executable **./create_a_file**. When finished list the contents of the folder and a new file has been created mydata.m. Type **cat mydata.m** and you will notice that a two dimensional array of data has been written to this file.

- To practice copying files from Linux to Windows go back to your Windows command prompt and move to the C: drive by typing **c:** and then change directory to your directory on the C:\ drive. From inside your directory type the following: **pscp root@192.168.1.???:/home/root/<yourDirectoryName>/testcopying/mydata.m .** This will copy mydata.m to your Windows folder. See that the contents of the file are correct by typing "**type mydata.m"** at the Windows prompt.
- Just in case you have trouble copying files to/from the PC using the Ethernet connection, I would like to give you a backup method using a USB drive/stick.
  - Unfortunately our Angstrom Linux cannot (or does not have the right options selected) mount a USB drive automatically. Instead you will need to plug in your USB drive and then power off and on the Beaglebone. So plug in your USB drive (if you do not have one ask your TA) and cycle the Beaglebone's power.
  - Once Linux has booted, change directory to the /dev folder. In the /dev folder type **ls s***. This will list all devices that start with s. Most of the time your USB drive will

be called “sda” and its working partition called “sda1”. It could also be called sdb, sdc, etc. Remember this label.

- Change directory back to /home/root/<yourdirectoryname>. Create a directory like **mkdir USBstick**. Then type **sync** to make sure this new directory is permanently stored.

- To mount the USB drive for use type

  mount /dev/sda1 /home/root/<yourdirectoryname>/USBstick

- Now change directory into USBstick and list all the files in your USB drive. Practice with the “cp” command copying files to and from your USB drive.

- Now use the monitor that is attached to the Beaglebone through a HDMI micro to VGA adapter cable and the wireless keyboard and mouse. Hopefully all three of those things are located near your Beaglebone. When you turn on the monitor you should see another login prompt. Yet another way to log into your Beaglebone. A cool side note is that all three login methods we have used so far, RS-232 terminal, terminal over Ethernet, and this monitor terminal, can all be used in tandem. So you could be working with the wireless keyboard and monitor while your partner could be working in one of the other terminals. Login this terminal again with root and no password. List the contents of your root directory and run one of the programs you built above to see that this is another terminal interface able to command Linux. This would be called a terminal window, but one that now has a head.

- We can now go a step further and start up a Linux X-windows interface to give Linux the look and feel of an operating system like Windows 7. At the monitor’s terminal type **gdm** and wait 20 to 30 seconds for the GNOME Display Manager (gdm) to launch. Now the wireless mouse is useful and you can click away. Where the headless interface to Angstrom Linux is very solid (I have not seen too many bugs) the windows interface is a bit flaky. For example gimp is a program like paint for Windows 7. If you run the gimp program, gdm freezes every time and you need to power off and power back on the Beaglebone. Also if you run a number of programs, gdm will get very slow or again freeze. These are all issues I am sure someone is working on “out there” and all of this will get better in time.

- Press together the keys Ctrl-Alt-F3. You will see another monitor terminal. There are actually 6 total, F1 – F6. Ctrl-Alt-F2 takes you back to the GDM window.

- “gedit” is the default text editor for Linux. Open up a Terminal, and at the prompt type **gedit**. This is much easier and more familiar then the non-graphical text editors. Here you can cut and paste code from one window to another, etc.

- In the second week of Lab 6 you will be working with the USB camera connected through the Beaglebone's USB port. Test your camera by running a small program named "Cheese". Find it under the menu item "Applications->Sound & Video" If a camera image does not display, select the Cheese menu item Edit->preferences and change the camera's resolution to 160 X 120. You should be able to see your pretty face, add effects, and snap some pictures or video. To view the pictures you take on the beaglebone, copy them from the cheese directory to your desktop and then open the images using the "evince" command. The cheese picture files are automatically stored at "/home/root/.gnome2/cheese/media/" with a name based on the photo date. Type **evince <picture's file name>** to view any of the pictures you take with cheese. We can also send these pictures to the windows PC by using the "pscp" command from a Windows command line. Open a Windows command prompt and type **pscp root@192.168.1.???:/home/root/.gnome2/cheese/media/<filename> <destination directory>** to copy the photo to <destination directory> on your Windows computer.

- As our last "Linux play time" exercise, we will mess a bit with python. Python can be thought of as both a scripting language and a programming language. Open another terminal window and type **python** and the prompt. This will open a python prompt. Here you can type in commands similarly as in Matlab. Type in the following commands to get a very small taste of python.

```
import numpy as np

A = np.matrix( [[4,9,2],[23,5,12],[95,120,23]]) # Creates a matrix.
x = np.matrix( [[6],[3],[9]] )                   # Creates a matrix
(like a column vector).
y = np.matrix( [[1,2,3]] )                       # Creates a matrix
(like a row vector).
b = np.matrix( [[4], [7.5], [13]] )
A.T                                     # Transpose of A.
A*x                                      # Matrix multiplication of A
and x.
A.I                                     # Inverse of A.
solution = np.linalg.solve(A, b)        # Solve the linear equation
system. A*solution = b
```

  You can exit the python command prompt by typing **quit() <return>**

  Had you typed these lines of code into a text file with extension .py, they could be executed in sequence by typing **python <filename>** in the Linux terminal.

  Python does not need the GDM display manager running. It also works from the "headless" terminals.

Exercise 2:

The goal of this exercise is to develop code for both Linux and the MSP430 to allow the Beaglebone (I2C Master) to communicate data to/from your MSP430F2272 processor (I2C slave) using the I2C serial port. The MSP430F2272's USCIB0 can be setup as an I2C serial port and the Beaglebone has an I2C serial port with a given Linux device driver. To get you started you are given a Linux program that every 200ms issues an I2C write of an 8 bit number to the I2C address 0x25 (your microcontroller's address will be set to this number). After the write command is complete it issues an I2C read asking the device at address 0x25 (your microcontroller) for an 8 bit number back. Both the value transmitted and the value received are printed to the terminal. First you need to make sure you have connected the I2C pins of the Beaglebone to the I2C pins of the MSP430F2272 and, like all I2C serial ports, both wires need to be pulled to Vcc through a 10Kohm resistor. Using the demo board as a guide:

- Solder a wire from the SCL pin of the Beaglebone SV8 three pin connector to MSP430F2272 pin P3.2. P3.2 was connected to the DAC so you will need to cut that wire.
- Solder a wire from the SDA pin of the Beaglebone SV8 three pin connector to MSP430F2272 pin 3.1. P3.1 was also connected to the DAC so cut that wire.
- With two 10Kohm resistors pull up to Vcc both P3.1 and P3.2.
- Solder a wire from the "REG 3.3V" 2X3 connector (top row is 3.3V) to any Vcc pin on the right of the board. This will power the MSP430F2272 when only the Beaglebone's power is connected.
- Solder a push button next to the MSP430F2272. This is a "Reset" button for the MSP430F2272 that is useful to restart your program when the debugger is not connected to the MSP430F2272.

At the end of this exercise you are going to be asked to command the position of a RC servo motor. The steps below explain how to wire for the RC servo motor.

- At the top of your board you will find three columns painted black, red and yellow. These colors are on your board because a labeling mistake was made at that connector. First solder a three pin header to the top most pads of those three columns.
- The black column is already connected to ground through the traces of the board so you do not have to connect a wire to that column.
- The red column needs to be connected to 5V. Solder a wire from the red column to one of the 5V pins of SV5 (also labeled BB_5V) down in the Beaglebone area.

- The yellow column needs to be connected to a PWM output. Solder a wire from the yellow column to MSP430F2272 P4.1/TB1. P4.1/TB1 may already be soldered to the 5 pin header for Lab 5. You do not need to disconnect that wire because nothing is connected to the 5 pin header.

Now that you are done with the soldering, boot back up your Beaglebone and copy and compile the given Linux program's code. Steps below:

- From a Beaglebone terminal, (take your pick, headless or monitor) change directory to your /home/root/<yournetid> directory. In that directory create another directory and name it "i2c_singlebyte". **mkdir i2c_singlebyte**
- On your Windows PC change directory to: **cd n:\me461\fall14\Lab6**.
- Copy all the i2c files from this directory to your Beaglebone.

  pscp i2c* root@192.168.1.<your_ip>:/home/root/<yournetid>/i2c_singlebyte/.
- At the Beaglebone terminal compile your i2c program

  Gcc –lm i2c.c i2c_test_byte.c –o single_byte
- You can run the executable if you would like but your MSP4302272 has not been programmed to receive the i2c data so timeout errors will be printed.
- Before you go onto the next steps view the given source files to understand even better what the executable is performing. What are the four parameters of the functions "i2c_write_bytes" and "i2c_read_bytes"?

Switch to creating a program for your MSP430F2272 that will communicate through the I2C serial port to the Beaglebone. Your initial program needs to wait for an 8 bit value from the beaglebone and then echo that 8 bit value back to the beaglebone. Follow these steps:

- Use the project creator to create a new project and import this project into Code Composer Studio.
- I would like you to start with a new "starter shell" of code that has a few changes to help you with the I2C serial port. This new starter shell is located at N:\me461\fall14\Lab6\user_msp430i2cshell.c. Open this file and copy its entire contents. Then open the main C file of your new project and paste the i2c shell contents over the top of the new project's main C file contents. (Or in other words replace the text of your new project C-file with the text of user_msp430i2cshell.c.)

- In your main() function's initialization section set the USCIB0 peripheral's registers to setup the USCIB0 as an I2C slave.
    - Set P3.1 and P3.2 as USCIB0 pins
    - Put USCIB0 into reset and set clock source to SMCLK
    - Set USCIB0 to synchronous mode and I2C mode.
    - Set the USCIB0's "Own Address" to 0x25. This is the address your Linux program will need to communicate with.
    - Pull the USCIB0 out of reset.
    - Finally enable only the UCB0RXIE interrupt. The I2C receive interrupt.
- Besides printing the I2C received value in the main() function's while loop, the remainder of the code you need to develop will be placed in the USCI0TX_ISR interrupt service routine. Scroll down to the USCI0TX_ISR function. The first 20 or so lines of that function have not changed. This code is for transmitting the "printf" character strings through USCIA0 to Tera Term. But below that code you may notice some differences. With I2C mode enabled there are now two more interrupt sources that cause the USCI0TX_ISR to be called. Both the I2C TX interrupt and the I2C RX interrupt use the USCIAB0TX_VECTOR (interrupt function). This is a bit confusing and hence the reason I wanted to give you a new starter shell. Follow the below steps to develop the I2C receive and transmit code:
    - Initially the UCB0RXIE interrupt is enabled in main(). When the beaglebone writes its first byte to the MSP430, the UCB0RXIE interrupt source will cause USCI0TX_ISR to be called. Inside UCB0RXIE's if statement:
        - Read the value received into a global variable, i.e. RXData.
        - Copy this value to another global variable to be used to send this received value back to the beaglebone, i.e. TXData.
        - Toggle an LED.
        - Set "newprint" to 1 telling the main() function to print this new receive value. (Since the Beaglebone is only sending us a character every 200ms you can print out each value received).
        - Last step is to disable the UCB0RXIE interrupt source and enable the UCB0TXIE interrupt source. (This is part of the protocol we are creating

here. The Beaglebone knows not to send any more information until it receives a byte back from the MSP430).

- This is all that is needed for the receive portion so the code falls out of the if statement and USCI0TX_ISR is exited.

- When the USCIB0 is ready to transmit, it will signal the UCB0TXIE interrupt. Since you just enabled that interrupt source before finishing the I2C receive code, the USCI0TX_ISR function will again be called. This time it will enter the UCB0TXIE if statement. In this if statement:

  - Write the transmit global variable's value to the USCIB0 transmit buffer.

  - Toggle a second LED.

  - Get ready for the next time the Beaglebone sends another byte by disabling the UCB0TXIE interrupt source and re-enabling the UCB0RXIE interrupt source.

- Up in the main() function's while loop print this value whenever "newprint" is set to 1.

- That's it. Now the MSP430 will wait for a byte from the Beaglebone, echo it back and print the value received.

- Use the oscilloscope to watch the output/input of the I2C SDA and SCL lines. Load your MSP430 with the program you just created and run it. Then in Linux run the single_byte application, **./single_byte**. single_byte should print what it transmitted and what it received back and your MSP430 should be printing what it received to its USB serial port.

Now that you have working programs on both and Beaglebone and the MSP430 that allow for I2C communication of one byte between the processors, we will take it one step further and transfer 8 bytes to/from the processors. Use these tips/hints to modify your communication scheme:

- First make a backup of your single byte transfer code of both your Beaglebone and MSP430.

- On the Beaglebone:

  - Change the variables rx and tx to arrays of 8 unsigned char.

- Now that rx and tx are arrays, C sees them as pointers. When you pass them to "i2c_write_bytes" or "i2c_read_bytes" you should remove the &.
- With the same timing, every 200ms, send the 8 values of your array to the I2C port. Then after the write function returns call the i2c_read_bytes function to read 8 bytes send back from the MSP430.
- With two print lines print all 8 bytes that were sent and all 8 bytes that were received.
- To make the display of RX and TX data a bit more interesting, change the data to transmitted (maybe add a constant) every new transmission.
- Compile your code, but wait to test until the MSP430 code has been developed.

- On the MSP430F2272:
  - Change RXData and TXData to 8 element unsigned chars.
  - Change the I2C RX interrupt function so that 8 bytes are received and stored to RXData before the TX interrupt is enabled. Remember that the RX interrupt is called whenever 1 byte has been received. So 8 interrupt calls will occur to receive the 8 bytes of data.
  - Once 8 bytes have been received from the Beaglebone, blink an LED and disable the RX interrupt and enable the TX interrupt. For now just echo back the 8 bytes just received so assign TXData's elements to RXData's elements
  - Change the I2C TX interrupt function so that 8 bytes of TXData are transferred back to the Beaglebone program. Again remember that only one byte is transferred across the I2C serial port per interrupt. So 8 interrupt calls will occur to send the 8 bytes of data.
  - Once the 8 bytes have been transmitted blink a LED, print at least 4 of the bytes recieved and disable the TX interrupt and enable the RX interrupt to get the I2C ready of the next transfer.
  - Try out your new programs.

Take the transmission of 8 bytes one step future. Now instead of transferring eight random bytes of data, use the 8 byte transfer to transfer two long integers (32bits/4bytes). So two long int transferred from the Beaglebone to the MSP430F2272 and then transfer two long int from the MSP430F2272 to the Beaglebone. Also don't echo the same data received back to the Beaglebone. Send something else back like a sampled ADC value in millivolts and the elapsed time of the microcontroller. Tips/hints:

- Here is an example of extracting the second byte of a long integer.

```
Bytetosend = (unsigned char)(mylong>>8);
```

- To put a long back together

```
Newlong = (((long)b3)<<24) +  (((long)b2)<<16) + (((long)b1)<<8) +  ((long)b0);
```

Depending on how much time you have, you may want to get started working with the RC servo. In Exercise 3 you will be asked to command an RC servo to move to different positions depending on the position of a bright color seen by a USB camera connected to the Beaglebone. The RC servo will be discussed in lecture. You will drive the RC servo with a 50 HZ carrier frequency PWM signal. The RC servo's PWM input is connected to P4.1/TB1. To command an RC servo, the PWM duty cycle is varied between approximately 3% duty cycle to 13% duty cycle.

Exercise 3:

In this exercise you are going to start communicating more meaningful messages from the Beaglebone to the MSP430 to control a RC servo. Eventually, we want to be able control a RC servo based on the location of a neon hat in the view of USB camera that is connected to the Beaglebone. This is going to require the Beaglebone to perform some image processing and transmit the location of the hat to the MSP430 over i2c. To do the image processing, we are going to rely on an open-source computer vision library called OpenCV. Originally developed by Intel, this library is a powerful tool that gives us access to a large variety of data types and functions useful for image processing.

Before we start thinking about programming anything for the Beaglebone, we need to first set up the MSP430 to output PWM signals capable of controlling our RC servo.

Controlling a RC servo from the MSP430 means we need to set up Timer B0 to output a PWM signal with varying duty cycle. We want our PWM signal to have a carrier frequency of 50 Hz and a varying duty cycle from 3%-13%. This range of duty cycles will command the RC servo to move to a position between -90^{o} to 90^{o}. (The 3%-13% guideline, which correlates to a pulse length of .6ms to 3ms, can vary slightly depending on which brand and type of RC servo you are using.)

- Set up Timer B0's CCR1 (TB1) compare registers on the MSP430 to output a PWM wave with a 50Hz frequency and a duty cycle that is linked to TB0CCR1's value.
- Keeping the same communication time period (200 ms) use one of the long integers you are already transmitting from the Beaglebone to change the position of the RC servo motor. In the

Beaglebone code, gradually increase (by say 5 or 10) the TBCCR1 value commanding the RC servo. When you have increased the PWM value to the point where the RC servo motor has been commanded to approximately 90^{o}, start decrementing the PWM value until approximately -90^{o} has been reached. Your code should then continue to repeat this sequence. Show this working to your TA.

Now we are going to start playing with image processing code on the Beaglebone. As a first step, we are going to compile and run some given image processing code that displays an image in the Beaglebone's windowed environment.

- On the Beaglebone, make a directory in **/home/root/<your net id>/** called "**lab6_openCV**". This is where you will put the image processing code we are about to play with.
- From the windows command line, use the **pscp** command to copy the file **Lab6_student** from **n:\me461\fall14\Lab6** to the Beaglebone. (**pscp n:\me461\fall14\Lab6\Lab6_student.c root@192.168.1.<your 3-digits>:/home/root/<yournetid>/lab6_openCV/.**).
- Because the code displays an image, we need to run it from the windowed environment in the Beaglebone. Log into the Beaglebone via the "headed" connection. After logging in as root, type **gdm** at the command window to start the windowed environment. (The Beaglebone will take a little time to start the gdm.) Your wireless mouse/keyboard and USB camera should have been plugged into the Beaglebone's USB hub before powered on.
- Once you are in the windowed environment, open a terminal window and **cd** to the file you just copied from the windows machine. Compile the code by typing :

  "**gcc -O2 –lm Lab6_student.c -o Lab6_student `pkg-config --cflags --libs opencv`** ". (Note that the " ` " character is found on the same key as the tilde(~), and is NOT an apostrophe. The reason we need to use **`pkg-config --cflags --libs opencv`** when we compile the code is to tell Linux to include all the OpenCV libraries. If you want to see what libraries are being included, you can just type **pkg-config --cflags --libs opencv** at the command prompt and see what is returned.)
- Run the code you just compiled. The video from your USB camera will appear on the Beaglebone monitor in the upper left corner. The program is taking RGB images from your camera, changing pixels that match closely "blue jean blue" to white, and displaying the image in a window on the Beaglebone. Once you start the code running, you won't be able to halt its execution by pressing "ctl+c" because the Beaglebone processor is too busy. So when you are done looking at yourself and blue items you find around the lab, use the

mouse to close the terminal window that you started this application in by selecting **File->Close Window.**

- Open up another command prompt and re-navigate to the directory with the compiled code. Use gedit to open up the "Lab6_student.c" by typing "**gedit Lab6_student.c**". Look at the given code and try to make sense of it. Note the following:

  - After #include statements, notice the "#define DISPLAYIMAGE" statement. By commenting out this #define statement, and recompiling the code, you can stop the code from displaying an image on screen. This happens to work because all the lines of code that are related to displaying an image are written in blocks that look like this:

```
#ifdef DISPLAYIMAGE
//do stuff related to displaying images
#endif
```

    These code blocks make the compiler disregard certain lines of code if "DISPLAYIMAGE" has not been defined.

  - We set the width and height of an image capture object to be 120rows X 160columns pixels and initialize some of the variables we will use to store images captured from the camera.

  - We enter a while loop with the condition "`(key !=27)`" (27 is the ASCII code for the ESC key). This loop is where all the image processing is done. On each iteration, we do the following:

    1) Capture an image from the camera

    2) Use two nested for loops to go through every pixel. For each pixel we pass the red, green, and blue values to the rgb2hsv function, which determines the hue, saturation, and value for each pixel and stores the values in "h," "s," and "v" variables, respectively. If the hue, saturation, and value of a pixel happens to fit in a particular range (color blue jean blue), that pixel is changed to white.

    3) Display the image.

We have already seen that displaying the image on the Beaglebone is hard work for the processor. So we will take advantage of Linux cross compiling capability and develop our C code first on the Linux laptop where we have more processing power and better display capabilities. Then when the vision code is finished we will move back to the Beaglebone where we will add

I2C communication with the MSP430. Transfer the C file to your Linux laptop. To communicate between the laptop and the Beaglebone through the network connection you first need connect the laptop wirelessly to the "MECHOMAP" router. (Right click on the wireless symbol in the top right of the Linux display and find "MECHOMAP") Then once connected you can use scp to transfer files back and forth. Use "ifconfig" to find the laptop's IP address and "pwd" to find the full path to locations on both the Beaglebone and the Linux laptop. The laptop user is "ge423user" with password "f33dback5". (Note: the user ge423user's home directory is /home/jeff. My TA Jeff Arena setup the first laptop that was imaged for each of the laptops in lab. We were able to change the user to ge423user but we have not figured out yet how to change the name of the home folder, sorry.)

By default, the code is looking for pixels that appear blue jean blue or close to it. Developing code first on the laptop, try to modify the code to look for a different color. To do this, you will need to estimate a range in hue, saturation, and value that corresponds to your color of choice. (An HSV color-wheel might be useful for finding suitable values. Note that, in the code HSV values are all in the range of 0-255 so you will have to scale the hue value from 0^o to 360^o to 0 to 255.) Once you think you have suitable limits in hue, saturation, and value, go into the code and change it to look for pixels that fit in this new range. Recompile and rerun the code. You should see that it no longer targets the same dark-blue pixels as before, and instead targets your color (or close to it). Show this working to your TA.

Now we are going to start trying to target the neon yellow hats with our OpenCV code. Estimating HSV ranges without any help from a computer can be difficult to do accurately, so we are going to enlist a more powerful tool, Matlab. We will analyze an image taken from the camera to determine precisely the best HSV range to detect neon yellow hats.

- First, we need an image of the hat. There is a c file in **n:\me461\fall14\Lab6** called **bmpCapture.c**. Copy this file to the Linux laptop and compile with the command:

  " **gcc –O2 -lm bmpCapture.c –o bmpCapture `pkg-config --cflags --libs opencv`** ". (Again the " ` " characters are not apostrophes.) Run the program you just compiled with the command "**./bmpCapture mybmp**". (Note: there is a web camera built into the laptop's monitor.) After running, there should be a new .bmp file on your desktop called "mybmp.bmp". If the bitmap doesn't look good for whatever reason (maybe "the front" is your bad side) feel free to take another image by the same procedure and make sure the bright yellow hat is in the picture.

- Transfer the image from your Linux laptop to **C:\<yourLabDirectory>** on the windows machine using pscp from the command prompt.

- Open Matlab and type **ME461_ColorThreshold C:\<yourLabDirectory\mybmp.bmp** at the command line. This will open up a user interface designed to find RGB or HSV ranges based on an image. The program will ask you to select a region of interest in the image, and then let you click on pixels out of that region. Left click adds pixels and right click removes pixels. The program will then give you the smallest possible HSV (or RGB) ranges necessary to capture all the pixels you choose.

- Take the HSV range values you just determined and once again modify the C code on the Linux laptop to target those pixels.

Now we are going to start calculating the location of the center of area of the neon hat, and using these values to control the RC servo on the MSP430.

- Add code to Lab6_student.c to calculate the total number of neon yellow pixels in the image and the centroid of these pixels (in both x(=columns) and y(=rows). (Note: in your centroid calculations, make sure never to divide by zero. This will cause runtime errors.)

- To check your center of area calculations, paint some sort of crosshair on the image that is displayed. (Note, make sure when you are coloring pixels for the cross hair that you do not write outside the bounds of the 120X160 image. Run Lab6_student on the Linux laptop and make sure it behaves as expected before moving on. Display the X and Y coordinates and the number of pixels found. Show it working to your TA.

- Use **scp** to copy code from the Linux laptop to the Beaglebone. Suppress the display output by commenting out the line "#define DISPLAYIMAGE" at the beginning of the code. Unsuppress the I2C and use I2C to send the column (x) centroid location and pixel number information to the MSP430. Make it so that if the number of neon yellow pixels in the image exceeds some realistic threshold, the RC servo motor is commanded to go to a location based on the x location of the hat in the image. (You can develop your C code on the Beaglebone in nano or gedit. Remember that to start gedit, you need to be in the gdm window environment of the Beaglebone.)

- Compile and run your code on the Beaglebone. Make the RC servo move by moving the hat around in front of the USB camera. What happens when you take the hat off screen? Show this all working to your TA.