ME 461 Laboratory #6
The C.H.I.P., 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 C.H.I.P. 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 your C.H.I.P. installed with the needed software and get you familiar with working in Linux at the "terminal" or "command prompt."

First you will focus on setting your C.H.I.P. board. The company that makes the C.H.I.P., "Next Thing Co," has made it very easy to install new Linux images on the C.H.I.P. board. For this class we are only going to work with the C.H.I.P. board using the "headless" Linux image. "Headless" means that an X windows environment will not be used. We will only be able to work with the C.H.I.P. through a command prompt or in Linux terms it is call a "terminal."

Go to the website http://flash.getchip.com/ and follow the instructions to "flash" your C.H.I.P. with the headless installation of Debian. Debian is a version of Linux. As of this writing the flash tools only work with the C.H.I.P. plugged into a USB 2.0 port. The top port on the lab PCs is a USB 2.0 port. You may have to close and open Chrome a few times if the flash does not complete successfully.

Once you have your C.H.I.P. flashed, plug it on to your break out board. Make sure the USB port is pointing off the break out board. Power the breakout board with 6 to 12V, we will use 12V in lab, and plug COM1's serial cable into the 9 pin connector. Pull up Tera Term and select Serial, COM1, 115200 Baud. Once you have everything connected, power on the 12 volt supply. At Tera Term you should see a bunch of debug text showing the boot state of the C.H.I.P. Once you see a login prompt, login with "chip" and password "chip".

We need to setup your C.H.I.P. so it can wirelessly connect to "Illinois_guest" when you want to download packages and other web files. In addition connect to the lab room's wireless router when you want to transfer files to/from the PC. To setup connecting to "Illinois_guest" you need the MAC address of your chip. Type "sudo ifconfig" and then type the password "chip". The MAC address, or also called the physical address, is a 12 digit hex number. Find and jot down the MAC address for the "wlan0" listing. Then you need to go to a U of I website to allow this device to connect to Illinois_guest. In Chrome go to http://go.illinois.edu/IllinoisNetDevices and login with your active directory account. Click

on the item “Add Device”. Fill in the given boxes with your C.H.I.P.’s MAC address and a name you make up for you C.H.I.P. In the Notes box you could make a note that this device is for ME461. Select the terms of use and create.

Connect to Illinois_guest by running the following command “sudo nmtui”. “nmtui” stand for Network Manager Text User Interface. In nmtui use the arrow keys and enter to select “Activate New Connection” and then select Illinois_guest. After a little time it should indicate that the C.H.I.P. is connected. Exit nmtui again using arrow keys and enter.

Check that your C.H.I.P.’s WIFI is connected to outside building web access by typing “ping www.google.com”. You should see replies if everything is working correctly. Then run these commands, which will take a bit, at your C.H.I.P.’s terminal:

sudo apt-get update *This syncs your C.H.I.P. with the current Debian repository*

sudo apt-get install git build-essential *This installs git and C/C++ Compiler*

sudo apt-get install libopencv-dev python-opencv *This installs OPENCV*

Once these install commands are done you have all the needed packages to complete the rest of this lab. For your final project, if you decide to use the C.H.I.P., you may need to install other packages.

Play a bit with Linux

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/chip/<yourdirectory> Changes current directory to /home/chip/<yourdirectory>

cd By itself takes you back to your home directory. On C.H.I.P. /home/chip

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

We will be working with Linux “Headless.” “Headless” means not connecting a keyboard, mouse or monitor to the C.H.I.P. 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 C.H.I.P.. 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 C.H.I.P.. 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 logging in.) After about 20 seconds or so Linux will complete its boot and the login prompt "C.H.I.P. 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: **chip**

Password: **chip**

Now perform the below steps to get you a bit more familiar with Linux and the C.H.I.P..

- Check and Change the Date. If the C.H.I.P. is connected to the internet, instead of the lab's local network, it will find the date from a NTP (Network Time Protocol) server. If you want to view data and/or change it, 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/chip
- 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 "**sudo ifconfig**" (interface config for the internet connection). Scroll up to the "wlan0" item and find your C.H.I.P.'s "inet addr", internet address.

- The Mechatronics lab has a local network that connects all the PCs to a wireless router call "Mechnight". Connect to this router using the command "sudo nmtui". Use your keyboard's arrow keys to navigate this application and activate a connection with "mechnight". Ask your TA for the password to "mechnight." Now your PC and can communicate and transfer files back and forth to the C.H.I.P.

- The other "headless" connection to the C.H.I.P. is through this internet connection. Use Putty to connect to your C.H.I.P. over the network. To connect with Putty, open a Command Prompt (cmd) in Windows 7 and type "**putty chip@192.168.1.???**" where ??? is your C.H.I.P.'s remaining IP address. (Run sudo ifconfig again to find your IP address.) This then turns the command prompt into a terminal connected to your C.H.I.P.. This terminal works identical to the RS-232 terminal connection except that you need to wait until the C.H.I.P. is fully booted before you try to connect over the internet. The C.H.I.P. needs to launch a SSH server before you can connect to it.

- From inside this internet terminal, 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 C.H.I.P.

- Now let's take a quick tour of the Linux running on the C.H.I.P. The distribution of Linux running on the C.H.I.P. is Debian. 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 C.H.I.P.'s processor are listed. For example find ttyS0. This is the serial port for the RS-232 terminal you used previously. Notice there are three i2c serial ports. We are going to use i2c-1. video0 is the USB camera connected to your C.H.I.P.. 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/chip/<yourDirectoryName>** or **cd**

~/<yourDirectoryName> or two commands **cd** to get back to /home/chip 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\Fall17\Lab6.** Type **dir** and you will see the file **create_a_file.c**. We want to copy this file down to the C.H.I.P.
- To copy the file to your C.H.I.P. you need to know its ip address again. (If you do not remember it from the above exercise, back at the Linux prompt type **sudo ifconfig** and find its ip address under wlan0. It will be 192.168.1.???.) Then go back to your Windows command prompt and type "**pscp create_a_file.c chip@192.168.1.???:/home/chip/<yourDirectoryName>/testcopying/."** (Don't forget the "." Period) This takes a few seconds to perform the copy. It will also ask you for the Linux password which is "chip." Now go back to the Linux terminal and list the contents of your testcopying directory and you should 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 chip@192.168.1.???:/home/chip/<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 Linux setup does not mount a USB drive automatically. It does recognize the USB drive but you need to mount it. Plug in the USB drive your TA gives you.

- 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/chip/<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 (change sda1 to your sd?)

  sudo mount /dev/sda1 /home/chip/<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. Remember to add sudo to your commands when you what to write something to the disk.

- A cool side note is that each terminal you use to connect to the C.H.I.P., Tera Term over serial and multiple Putty sessions over Ethernet can all be used in tandem. So you could be working with the serial interface and Tera Term while your partner is on another PC connected to your C.H.I.P. through Putty. Show this to yourself by creating both a Tera Term session and a Putty session on your PC.

- 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. At your Linux 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.

Exercise 2:

The goal of this exercise is to develop code for both Linux and the MSP430 to allow the C.H.I.P. (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 C.H.I.P. 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-1 pins of the C.H.I.P. 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. These pullup resistors are already solder on the C.H.I.P. board so we do not have to solder them on our break board. Using the demo board as a guide and using the C.H.I.P. pinout diagram you can find online by searching for "C.H.I.P. pinout".

- Solder a wire from the TWI1-SCK pin of the C.H.I.P. 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 TWI1-SDA pin of the C.H.I.P. to MSP430F2272 pin 3.1. P3.1 was also connected to the DAC so cut that wire.
- Solder a wire from the "+3.3V" 1X5 connector to any Vcc pin on the right of the board. This will power the MSP430F2272 when only the C.H.I.P.'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 two sets of three columns labeled OUT, PWR, GND. First solder a three pin header to the top most pads of one of those three column sets.
- The GND 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 1X5 connector labeled +5V down in the C.H.I.P. area.

- The OUT column needs to be connected to a PWM output. Solder a wire from the OUT 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 C.H.I.P. and copy and compile the given Linux program's code. Steps below:

- From a C.H.I.P. terminal change directory to your /home/chip/<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\fall17\Lab6**.

- Copy all the i2c files from this directory to your C.H.I.P..

  "pscp i2c* chip@192.168.1.<your_ip>:/home/chip/<yournetid>/i2c_singlebyte/."

- At the C.H.I.P. 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 C.H.I.P.. Your initial program needs to wait for an 8 bit value from the C.H.I.P. and then echo that 8 bit value back to the C.H.I.P. 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\fall17\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 C.H.I.P. 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 C.H.I.P., i.e. TXData.
        - Toggle an LED.
        - Set "newprint" to 1 telling the main() function to print this new receive value. (Since the C.H.I.P. 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 C.H.I.P. 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. 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 C.H.I.P. 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 C.H.I.P., 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 C.H.I.P. 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 C.H.I.P. and MSP430.
- On the C.H.I.P.:
  - 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 C.H.I.P., 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 C.H.I.P. 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 C.H.I.P. to the MSP430F2272 and then transfer two long int from the MSP430F2272 to the C.H.I.P. Also don't echo the same data received back to the C.H.I.P. 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 (NOTICE where the parenthesis are at)

  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 C.H.I.P. 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 C.H.I.P. 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 C.H.I.P. This is going to require the C.H.I.P. 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 C.H.I.P., 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 C.H.I.P. to change the position of the RC servo motor. In the C.H.I.P. 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.

In this next part you will be working with a USB camera connected to the C.H.I.P. and you will be able to view the camera's image on the second VGA monitor on the bench. There is a small issue with the default Linux image we have running on the C.H.I.P. in that after a number of minutes of inactivity the video stream to the monitor shuts off (screen save mode). This wouldn't be an issue if we had a USB keyboard attached to the C.H.I.P. because we could just hit a key to wake up the screen. Since we prefer not to attach a keyboard, we need to tell Linux not to turn off the screen. Perform the following steps:

- sudo nano /root/screenNoblank.sh *Then add to the file:*
  TERM=linux
  /usr/bin/setterm –blank 0 > /dev/tty1
  Ctrl-O to save
  Ctrl-X to exit

- sudo chmod +x /root/screenNoblank.sh *makes file executable*

- sudo crontab –e *you may have to select nano as the editor and then scroll to the bottom of the file and add the line*
  @reboot sudo sleep 20;sudo /root/screenNoblank.sh
  Ctrl-O to save
  Ctrl-X to exit

- *Now reboot by typing*:
  sync *to make sure everything is saved to disk*
  sudo shutdown now *wait for it to stop sending messages to Tera Term*

- *Power off and on your C.H.I.P. and wait for it to boot. Then perform this check to make sure* consoleblank *is equal to* 0.
  cat /sys/module/kernel/parameters/consoleblank

Now we are going to start playing with image processing code on the C.H.I.P. As a first step, we are going to compile and run some given image processing code that displays an image to the C.H.I.P.'s monitor.

- On the C.H.I.P., make a directory in **/home/chip/<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\fall17\Lab6** to the C.H.I.P. ("**pscp n:\me461\fall17\Lab6\Lab6_student.c chip@192.168.1.<your 3-digits>:/home/chip/<yournetid>/lab6_openCV/.**").

- 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.)

- Plug the Monitor's 3.5 mm connector into your C.H.I.P.'s connector and run the code you just compiled. The video from your USB camera will appear on the C.H.I.P. monitor. The program is taking RGB images from your camera, changing pixels that match closely "blue jean blue" to green, and displaying the image to the C.H.I.P.'s frame buffer. Play around viewing different blue objects in the lab. Press any key to stop the Lab6_student application.

- Use nano to open up the "Lab6_student.c" by typing "**nano Lab6_student.c**". Look at the given code and try to make sense of it. (You can also edit the file on the Windows PC and copy it down to your C.H.I.P. when you are done.) Note the following:

  - 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 and will keep on looping until any key is pressed on the keyboard. 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 green.

    3) Display the image to the framebuffer.

By default, the code is looking for pixels that appear blue jean blue or close to it. 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 or another brightly colored object 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\fall17\Lab6** called **bmpCapture.c**. Copy this file to your C.H.I.P. 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**". After running, there should be a new .bmp file called "mybmp.bmp".

- Transfer the image from your C.H.I.P. to **C:\<yourLabDirectory>** on the windows machine using pscp from the command prompt. Open the .bmp file to see how it looks. If the bitmap doesn't look good for whatever reason take another image with the same procedure above and recopy it to windows.

- 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 in your C.H.I.P. program 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. Display the X and Y coordinates and the number of pixels found. Show it working to your TA.

- Uncomment the line close to the top of your C file "#define I2C". When uncommented, this adds the I2C initialization and closing code to your C file. You still need to add similar code you wrote previously sending the position of the RCservo to the MSP430. Send the column (x) centroid location and number of pixels 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 C.H.I.P. in nano or on your Windows machine and copy the source down to the C.H.I.P. when you want to compile and try it out.) To compile your code you need to add i2c.c to the command: "**gcc -O2 –lm i2c.c Lab6_student.c -o Lab6_student `pkg-config --cflags --libs opencv`** "

- Compile and run your code on the C.H.I.P. 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.