

Home Automation with the DragonBoard™ 410c Development Kit

Written by: Aswin Sivaraman, Matthew Kneiser, Spencer Williams

Qualcomm Technologies, Inc.

Now it is your turn to use the DragonBoard 410c! The board is currently flashed with a Linux-based operating system (Debian 15.04), and the LXDE (*Lightweight X11 Desktop Environment*). If you would like to learn how to switch your board's operating system, consult the Appendix.

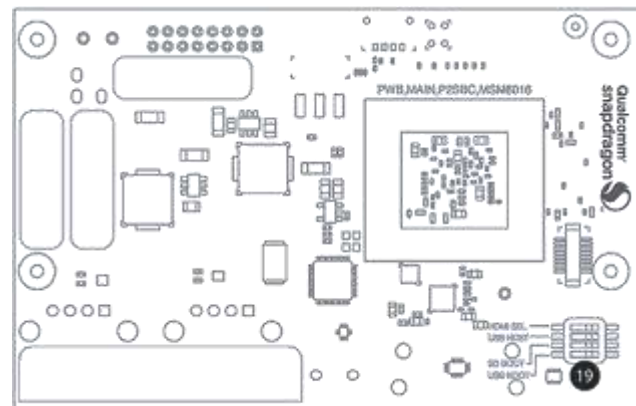
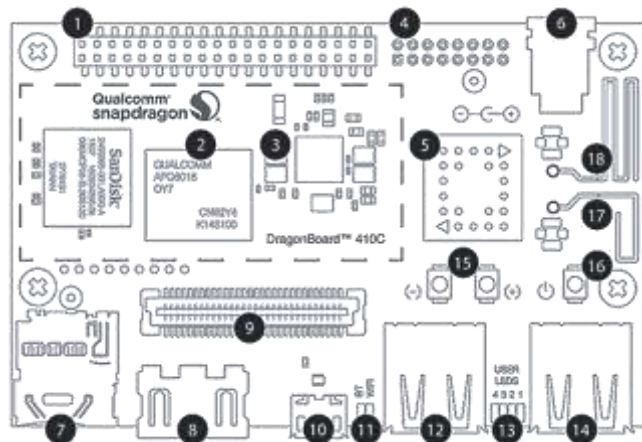
Goal

The aim of this workshop is to familiarize students with the Python scripting language, the OpenCV (*Open Source Computer Vision*) library, the Linux operating system, and the DragonBoard 410c. Participants will create a small Python program which utilizes a USB webcam for a variety of applications. These include: taking a picture, saving a picture, displaying a live video feed from the webcam, and performing facial recognition. We will be recreating the basic functions of a “smart security/surveillance camera”.

Phase 0: Overview

DragonBoard Overview

1. (J8) Low Speed Expansion Connector
2. APQ8016 Qualcomm® Snapdragon™ Processor
3. (U9) Power Management PMIC
4. (J7) Analog Expansion Connector
5. WLAN/Bluetooth/GPS
6. (J1) Power Jack
7. (J5) uSD Card Socket
8. (J6) HDMI Type-A Port
9. (J9) High Speed Connector
10. (J4) Micro USB Type B Connector
11. Bluetooth/WLAN LED's
12. (J3) USB Host2 Connector
13. User LED's 1-4
14. (J2) USB Host1 Connector
15. (S3-4) Vol+/Vol- Buttons
16. (S2) Power Button
17. Bluetooth/WLAN Antenna
18. GPS Antenna
19. (S6) Boot Switches



Power Button

There is a power button on the DragonBoard 410c. It is labeled S2 on the board. This button does not turn the power off. Rather, it operates in a similar fashion to a “lock button” on a smartphone. Keep in mind that the DragonBoard 410c is built on a smartphone platform, so many mechanisms and concepts are similar to smartphones.



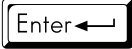
To turn off the device power, simply unplug the power adapter cable from the board. You will not need to use this button at any point during the workshop.



Review of UNIX Commands

The following commands will be used for this workshop. Within this handout, UNIX commands will be indicated through a terminal (command line) window, as shown below:





```
Terminal
sudo apt-get install -y gedit
gedit some_file.txt &
```

The above commands are an example of something you would type into the command line.

These are two, separate commands. To execute a command, press the  key. Please wait for a command to complete before entering the next command.

When you run a command in the terminal, it will instantiate a process, or program. To forcefully quit any program, you may press the  +  keys.

All code will be written and saved using text-editor **gedit**. To save any changes made to code written in this workshop, press the  +  keys, or use the mouse to click ‘Save’.

To switch between any open windows within the operating system, press the  +  keys. For example, if you have both gedit and the **UXTerm** (terminal window) open, you may alternate between these two windows using  + .

Phase 1: Setup

Hardware Setup

In this phase, you will configure the DragonBoard 410c for the purposes of this workshop by setting up a wireless Internet connection and installing software packages in the UNIX terminal. To get started with the DragonBoard 410c, follow these steps in order:

1. Connect the HDMI monitor, USB keyboard, and USB mouse. Refer to the diagram shown in Phase 0 if you need help identifying the right ports. **Do not** power the DragonBoard 410c yet.
2. Power on your HDMI monitor **while** the DragonBoard 410c is still unpowered.
3. On the backside of the board, verify that the four S6 switches are set to 0-0-1-0, as shown below. This will enable “USB Host” only. The other controls are not needed for the workshop, as the operating system is already loaded onto the board.



4. Only after all peripherals from Step 1 are connected, plug in the provided power adapter.
5. Wait for the onboard LED lights to flash, and you should see the booting up on the monitor after a few seconds.

Wireless Network Setup

Parts of this workshop will require access to the Internet, so now you will configure the wireless network connection. On first boot, you should see a wireless network indicator in the bottom right corner, as shown below (two squares with an x between them).



Click on the wireless network indicator and select the appropriate wireless network. Ask your workshop administrators to provide the network information.



Enter the network credentials (if applicable) at the prompt, and click 'Connect'.



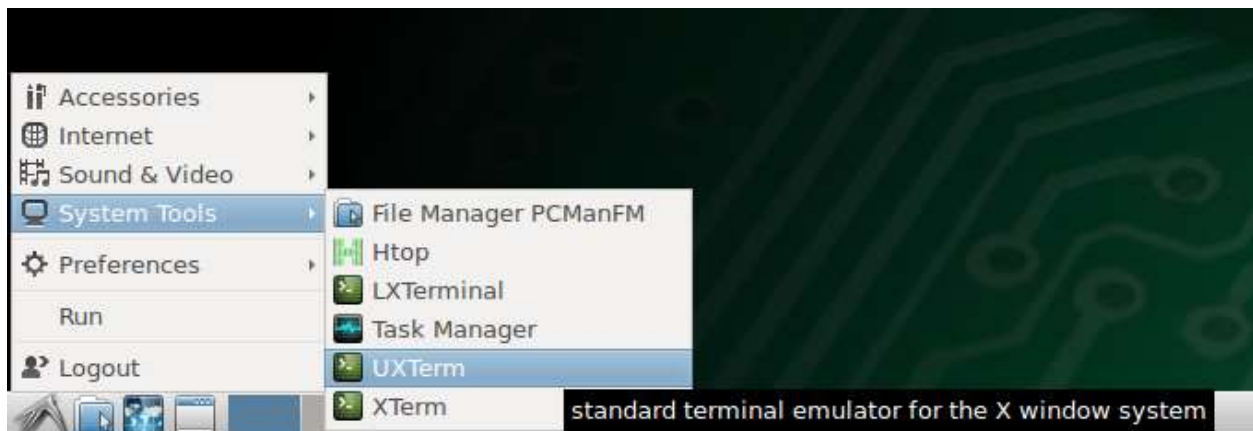
The network may require visiting a webpage for additional authentication. Click the browser icon in the bottom left corner (looks like a map), and try accessing a webpage.



If you run into issues configuring the wireless network, check out the ‘**Troubleshooting Wi-Fi Connection Issues**’ appendix at the end of this document, or ask for help.

Installing New Software

We’ll need to install a few software packages for this workshop. First, we will install **gedit**, a popular and easy-to-use Linux text editor. Open a terminal window by clicking the ‘Start’ button in the bottom-left corner, and select System Tools → UXTerm (or LXTerminal or XTerm).



When the terminal window comes up, type in the following commands in the exact order that follows:

```
Terminal

sudo apt-get update -y
sudo apt-get upgrade -y
sudo apt-get install -y gedit
```

Each of the above UNIX commands will take several minutes to complete. Wait for each command to complete before typing in the next command.

Logistics

The DragonBoard 410c comes with two USB ports on it. In this workshop, you will be using a USB mouse, USB keyboard, and USB webcam, for a total of three devices. The DragonBoard 410c only comes with two onboard USB ports, so you will need to alternate between the mouse and webcam on the same port.

The keyboard is the most important peripheral. Almost everything in this workshop can be done using only the keyboard, so it is recommended that you use the same port for both the

mouse and webcam. In the next step, we will see how to use the keyboard for everything and remove the need for a mouse altogether.

You will now create and enter a new directory, or file folder, for this workshop. With the terminal window you had previous opened, enter the following command:

Terminal
<pre>mkdir ~/workshop cd ~/workshop</pre>

Installing the OpenCV Library

Before proceeding to write Python code, you will need to install the OpenCV framework, which will allow us to interact with the USB webcam and manipulate the image. To begin, run this command in your terminal window to install OpenCV for Python:

Terminal
<pre>sudo apt-get install -y python-opencv</pre>

To ensure that it installed successfully, open a Python interpreter and try to import the OpenCV library. To open a Python interpreter, simply enter in the terminal window:

Terminal
<pre>python</pre>

You will now see an output similar to the following, succeeded by three arrows pointing to the right (>>>).



Example Output
<pre>Python 2.7.9 (default Mar 1 2015, 12:52:23) [GCC 4.9.2] on linux2 Type "help", "copyright", "credits" or "license" for more information.</pre>

Whenever the three arrows (>>>) are present, your terminal window is operating within the Python interpreter. To try importing OpenCV, enter the following command inside the interpreter:

Terminal > Python Interpreter
<pre>>>> import cv2</pre>

If the import fails, an error message will appear stating “ImportError: No module named cv2”. If you see this, please contact your workshop administrators to help with installing OpenCV.

You will need to exit the Python interpreter to return to the UNIX terminal. To do so, press the

 +  keys, or simply type in:

```
Terminal > Python Interpreter
```

```
>>> exit()
```

Cascade Classifier Theory

We will be using the OpenCV cascade classifier to get facial recognition working. This classifier will break down the task of detecting faces into various stages. For each stage, it does a very rough and quick test. If that passes, it does a slightly more detailed test, and so on. The algorithm may have 30-50 of these stages or cascades, and it will only detect a face if all stages pass. The advantage is that the majority of the pictures will return negative during the first few stages, which means the algorithm won't waste time testing all 6,000 features on it. Instead of taking hours, face detection can now be done in real time.

Though the theory may sound complicated, in practice it is quite easy. The cascades themselves are just XML files that contain OpenCV data used to detect objects. You initialize your code with the cascade you want, and then it does the work for you. Since face detection is such a common case, OpenCV comes with a number of built-in cascades for detecting everything from faces to eyes to hands and legs. There are even cascades for non-human things.

Obtaining the Cascade Classifier

We will be using the Haar Cascade Frontal Face Classifier for this workshop. To download the XML file for this workshop, enter the following command:

```
Terminal
```

```
wget http://pulse.ece.illinois.edu/2016/frontalface.xml
```

This will download the XML file which will be used later for facial recognition. To verify that the file is downloaded into your folder, run the “ls” utility, which lists the contents of your current folder:

```
Terminal
```

```
ls -lh
```

The contents of the directory will be shown as follows:

Example Output

```
frontalface.xml
```

Phase 2: Playing with OpenCV

Take a webcam snapshot

To begin, we will write a Python program which will read in a single image frame from the USB webcam and display this frame on a new window. You will need to create a Python source file using your text editor, **gedit**. To do so, enter the following command in your terminal:

Terminal

```
gedit take_picture.py &
```

This should now open the **gedit** text editor with the name of your new file shown in the top pane, like so:



In Python, code blocks (aka functions) are defined by their indentation. It is important that you properly and consistently indent your code. Any differences in spacing will result in a program error.

Also in Python, if a line begins with the “#” symbol, it is a comment. Comments are merely included in code to make readability easier, and are therefore optional. In other words, you *can skip* typing out the bolded blue text if you’d like. Enter the following code into your text editor, line by line:

gedit take_picture.py

```
#!/usr/bin/env python
import cv2, sys

# Define constants
DEVICE_NUMBER = 0

# Initialize webcam
vc = cv2.VideoCapture(DEVICE_NUMBER)

# Check if the webcam works
if vc.isOpened():
    # Try to get the first frame
    retval, frame = vc.read()
else:
    # Exit the program
    sys.exit(1)

# Read in the next frame
retval, frame = vc.read()

# Show the frame to the user
cv2.imshow("DragonBoard 410c Workshop", frame)

# Exit program after waiting indefinitely for a pressed key
cv2.waitKey(0)
```

Save your code (**Ctrl** + **S** or click ‘Save’), then exit the text editor program (**Ctrl** + **W** or click the “X” on the top-right of the window).

After returning to your terminal window, try executing your new program by running this command:

Terminal

```
python take_picture.py
```

If it works correctly, you should see a picture taken from the webcam appear in a new window. To close the window, simply press any button on the keyboard. You now have your first functional Python program!

Save a webcam snapshot to an image file

We can extend the program we wrote in the last step to save the image that the webcam took to a file. Copy your previous file to a new one, and open the new file up in your text editor. Enter the following two commands:

Terminal

```
cp take_picture.py save_picture.py
gedit save_picture.py &
```

You have now opened a second Python source file, whose contents are exactly that of your first program. Edit this file in **gedit** to include some new lines of code. Note that:

- New lines of code will be indicated with a green background.
- Modified lines of code will be indicated with a yellow background.
- Removed lines of code will be indicated with a red background and a strikethrough.

gedit save_picture.py

```
#!/usr/bin/env python
import cv2, sys

# Define constants
DEVICE_NUMBER = 0
IMAGE_FILE = "output.jpg"

# Initialize webcam
vc = cv2.VideoCapture(DEVICE_NUMBER)

# Check if the webcam works
if vc.isOpened():
    # Try to get the first frame
    retval, frame = vc.read()
else:
    # Exit the program
    sys.exit(1)

# Read in the next frame
retval, frame = vc.read()

# Save the frame as an image file
cv2.imwrite(IMAGE_FILE, frame)

# Read the output file
img = cv2.imread(IMAGE_FILE)
```

```
# Show the saved image on the screen
cv2.imshow("DragonBoard 410c Workshop", img)

# Exit program after waiting indefinitely for a pressed key
cv2.waitKey(0)
```

You have now added three new lines of code and changed one existing line. The three added lines will write the frame image into an external file (as defined by `IMAGE_FILE`, set to "output.jpg"), and then open the file. The modified line now shows the opened image file to the user.

Save your changes and exit the text editor. Return to the terminal and run your new program by entering the command:

```
Terminal
python save_picture.py
```

You should see a window popup with a still image taken from the webcam. Recall that pressing any keyboard key will dismiss the new window with the picture. Your new program has now saved the file as "output.jpg". You can check the contents of your current directory and locate this file. Run the "ls" utility to see the contents of your current directory:


```
Terminal
ls -lh
```

The contents of the directory will show as follows:

```
Example Output
frontalface.xml
output.jpg
save_picture.py
take_picture.py
```

You can open this picture with a Linux built-in picture viewer like so:

```
Terminal
gpicview output.jpg
```

Press the  key to close the gpicview window.

Write text onto an image

We can extend the program we wrote in the last step to save an image that the webcam took with the filename written on it to a file. Copy your previous file to a new one, and open the new file up in your text editor.

Terminal

```
cp save_picture.py write_text.py
gedit write_text.py &
```

You have now opened a third Python source file, whose contents are exactly that of your second program. Edit this file in **gedit** to include some new lines of code.

gedit write_text.py

```
#!/usr/bin/env python
import cv2, sys

# Define constants
DEVICE_NUMBER = 0
IMAGE_FILE = "output_with_text.jpg"
FONT_FACES = [
    cv2.FONT_HERSHEY_SIMPLEX,
    cv2.FONT_HERSHEY_PLAIN,
    cv2.FONT_HERSHEY_DUPLEX,
    cv2.FONT_HERSHEY_COMPLEX,
    cv2.FONT_HERSHEY_TRIPLEX,
    cv2.FONT_HERSHEY_COMPLEX_SMALL,
    cv2.FONT_HERSHEY_SCRIPT_SIMPLEX,
    cv2.FONT_HERSHEY_SCRIPT_COMPLEX
]

# Initialize webcam
vc = cv2.VideoCapture(DEVICE_NUMBER)

# Check if the webcam works
if vc.isOpened():
    # Try to get the first frame
    retval, frame = vc.read()
else:
    # Exit the program
    sys.exit(1)

# Read in the next frame
retval, frame = vc.read()
```

```

# Write the filename onto the frame using every font
for i in xrange(len(FONT_FACES)):
    font_typeface = FONT_FACES[i]
    font_scale = 2
    font_color = (255,255,255)
    x = 0
    y = (i+1)*50
    cv2.putText(frame, IMAGE_FILE, (x,y), font_typeface, font_scale, font_color)

# Save the frame as an image file
cv2.imwrite(IMAGE_FILE, frame)

# Read the output file
img = cv2.imread(IMAGE_FILE)

# Show the saved image on the screen
cv2.imshow("DragonBoard 410c Workshop", img)

# Exit program after waiting indefinitely for a pressed key
cv2.waitKey(0)

```

You have just added a new constant and a for-loop. The new constant, “FONT_FACES”, is an array which contains the names of all the fonts used by the OpenCV library. There are 8 fonts in total, so the length of the array is 8.

In the newly added for-loop, you use the variable “i” to iterate through all of the “FONT_FACES” fonts. The indented section defines the font properties, such as color, scale, and position (using coordinates x and y).

You then use the OpenCV library to put the text onto the frame.

Save your changes and exit the text editor. Return to the terminal and run your new program by entering the command:

```

Terminal
python write_text.py

```

You should see a window popup with a still image taken from the webcam. This new still should contain 8 different texts with differing fonts, all of which say the filename (IMAGE_FILE, which is now “output_with_text.jpg”).

Recall that pressing any keyboard key will dismiss the new window with the picture.

Your new program has now saved the file as “output_with_text.jpg”. You can check the contents of your current directory and locate this file. Run the “ls” utility to see the contents of your current directory:

Terminal
<pre>ls -lh</pre>

The contents of the directory will show as follows:

Example Output
<pre>frontalface.xml output.jpg output_with_text.jpg save_picture.py take_picture.py write_text.py</pre>

Display a live video from the webcam

We can extend the program we wrote in the last step to show a live feed of video from the webcam with text written on it. Copy your previous file to a new one, and open the new file up in your text editor.

Terminal

```
cp write_text.py show_video.py
gedit show_video.py &
```

You have now opened a fourth Python source file, whose contents are exactly that of your third program. Edit this file in **gedit** to include some new lines of code.

gedit show_video.py

```
#!/usr/bin/env python
import cv2, sys

# Define constants
DEVICE_NUMBER = 0
IMAGE_FILE = "output_with_text.jpg"
FONT_FACES = [
    cv2.FONT_HERSHEY_SIMPLEX,
    cv2.FONT_HERSHEY_PLAIN,
    cv2.FONT_HERSHEY_DUPLEX,
    cv2.FONT_HERSHEY_COMPLEX,
    cv2.FONT_HERSHEY_TRIPLEX,
    cv2.FONT_HERSHEY_COMPLEX_SMALL,
    cv2.FONT_HERSHEY_SCRIPT_SIMPLEX,
    cv2.FONT_HERSHEY_SCRIPT_COMPLEX
]

# Initialize webcam
vc = cv2.VideoCapture(DEVICE_NUMBER)

# Check if the webcam works
if vc.isOpened():
    # Try to get the first frame
    retval, frame = vc.read()
else:
    # Exit the program
    sys.exit(1)

# If the webcam read is successful, loop indefinitely
while retval:
```



```

# Write text onto the frame using a single font
for i in xrange(len(FONT_FACES)):
    font_typeface = FONT_FACES[5]
    font_scale = 2
    font_color = (0,0,255)
    font_weight = 5
    x = 0
    y = 50
    cv2.putText(frame, "[LIVE]", (x,y),
                font_typeface, font_scale, font_color, font_weight)

# Save the frame as an image file
cv2.imwrite(IMAGE_FILE, frame)

# Read the output file
img = cv2.imread(IMAGE_FILE)

# Show the frame on the screen
cv2.imshow("DragonBoard 410c Workshop", frame)

# Read in the next frame
retval, frame = vc.read()

# Exit program if the ESCAPE key is pressed
if cv2.waitKey(1) == 27:
    break


```

Make sure that all of the code underneath the “while” statement is properly indented!

There are significant changes to our program now, which we will explore in detail:

Our program now uses a “while” statement to loop indefinitely so long as the webcam constantly gives the program a new frame. Looping through these frames is what gives us live video!







We have removed the “for”-loop because we no longer want to loop through all of the fonts. Instead, we have picked one font to use (in this case, the fifth font). Additionally, we have added a new parameter “font_weight”, which gives thickness to the text. We have changed the “font_color” from white to red. We have changed the text from the file name to “[LIVE]”. Additionally, we have removed the two lines of code pertaining to saving the frame, as we no longer want to save a still image.

Lastly, we have modified the “waitKey” call to now look and see if the returned key code is equal to 27. This is the key code for the  key. If that key is pressed, we want to “break” out of the “while”-loop and terminate the program.

Save your changes and exit the text editor. Return to the terminal and run your new program by entering the command:

```
Terminal  
python show_video.py
```

You should see a window popup showing a “LIVE” feed from your webcam. Smile! You’re on camera now!

Try to see if you can exit your program using the  key, as we now check for a specific key code in the program. If you are unable to exit the webcam feed using the  key, switch back to your terminal window using the  +  keys and terminating the process by using the  +  keys.

Convert the video into grayscale

We will once again show a live webcam feed, but this time, our goal is to convert the frame into a grayscale image. This is a lossy operation; that means, we lose the color information in the frame by converting it to grayscale. If you convert the video frame back into color/red-green-blue (RGB) space, the result will still be a grayscale frame.

However, we would still like to convert BACK into the RGB-space because we would like the onscreen “[LIVE]” text to still be red.

Copy your previous file to a new one, and open the new file up in your text editor.

Terminal

```
cp show_video.py convert_gray.py
gedit convert_gray.py &
```

You have now opened a fifth Python source file, whose contents are exactly that of your fourth program. Edit this file in **gedit** to include some new lines of code.

gedit convert_gray.py

```
#!/usr/bin/env python
import cv2, sys

# Define constants
DEVICE_NUMBER = 0
FONT_FACES = [
    cv2.FONT_HERSHEY_SIMPLEX,
    cv2.FONT_HERSHEY_PLAIN,
    cv2.FONT_HERSHEY_DUPLEX,
    cv2.FONT_HERSHEY_COMPLEX,
    cv2.FONT_HERSHEY_TRIPLEX,
    cv2.FONT_HERSHEY_COMPLEX_SMALL,
    cv2.FONT_HERSHEY_SCRIPT_SIMPLEX,
    cv2.FONT_HERSHEY_SCRIPT_COMPLEX
]

# Initialize webcam
vc = cv2.VideoCapture(DEVICE_NUMBER)

# Check if the webcam works
if vc.isOpened():
    # Try to get the first frame
    retval, frame = vc.read()
else:
```

```

# Exit the program
sys.exit(1)

# If the webcam read is successful, loop indefinitely
while retval:

    # Convert frame to grayscale to wipe out color data
    frame = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    # Convert back to color (to use color text)
    frame = cv2.cvtColor(frame, cv2.COLOR_GRAY2RGB)

    # Write text onto the frame using a single font
    font_typeface = FONT_FACES[5]
    font_scale = 2
    font_color = (0,0,255)
    font_weight = 5
    x = 0
    y = 50
    cv2.putText(frame, "[LIVE]", (x,y),
                font_typeface, font_scale, font_color, font_weight)

    # Show the frame on the screen
    cv2.imshow("DragonBoard 410c Workshop", frame)

    # Read in the next frame
    retval, frame = vc.read()

    # Exit program if the ESCAPE key is pressed
    if cv2.waitKey(1) == 27:
        break

```

The OpenCV `cvtColor` function converts the frame object from one color-space to another in a lossy fashion. We switch to-and-from the grayscale space to make the frame grayscale. This operation is done in the two new lines of code shown above.

Save your changes and exit the text editor. Return to the terminal and run your new program by entering the command:

Terminal

```
python convert_gray.py
```

You should see a window popup showing a “LIVE” feed from your webcam. However, now the image should be in grayscale. The text on the top-left should still be red.

Identify faces in the webcam video

We would now like to identify faces as seen on the live webcam feed. If a face is detected, we want a rectangle to show outlining the boundaries of the face. We will extend upon the code in our grayscale program. Copy your previous file to a new one, and open the new file up in your text editor.

Terminal

```
cp convert_gray.py detect_face.py
gedit detect_face.py &
```

You have now opened a sixth Python source file, whose contents are exactly that of your fifth program. Edit this file in **gedit** to include some new lines of code.

gedit detect_face.py

```
#!/usr/bin/env python
import cv2, sys

# Define constants
DEVICE_NUMBER = 0
FONT_FACES = [
    cv2.FONT_HERSHEY_SIMPLEX,
    cv2.FONT_HERSHEY_PLAIN,
    cv2.FONT_HERSHEY_DUPLEX,
    cv2.FONT_HERSHEY_COMPLEX,
    cv2.FONT_HERSHEY_TRIPLEX,
    cv2.FONT_HERSHEY_COMPLEX_SMALL,
    cv2.FONT_HERSHEY_SCRIPT_SIMPLEX,
    cv2.FONT_HERSHEY_SCRIPT_COMPLEX
]

# Get XML file input
if len(sys.argv) > 1:
    XML_PATH = sys.argv[1]
else:
    print "Error: XML path not defined"
    sys.exit(1)

# Initialize the cascade classifier
faceCascade = cv2.CascadeClassifier(XML_PATH)

# Initialize webcam
vc = cv2.VideoCapture(DEVICE_NUMBER)

# Check if the webcam works
```

```

if vc.isOpened():
    # Try to get the first frame
    retval, frame = vc.read()
else:
    # Exit the program
    sys.exit(1)

i = 0
faces = []

# If the webcam read is successful, loop indefinitely
while retval:

    # Convert back to color (to use color text)
    frame = cv2.cvtColor(frame, cv2.COLOR_GRAY2RGB)

    # Write text onto the frame using a single font
    font_typeface = FONT_FACES[5]
    font_scale = 2
    font_color = (0,0,255)
    font_weight = 5
    x = 0
    y = 50
    cv2.putText(frame, "[LIVE]", (x,y),
    font_typeface, font_scale, font_color, font_weight)

    # Define the frame which the program will show
    frame_show = frame

    if i % 5 == 0:
        # Convert frame to grayscale to perform facial detection
        frame = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

        # Detect objects and return an array of faces
        faces = faceCascade.detectMultiScale(
            frame,
            scaleFactor=1.2,
            minNeighbors=2,
            minSize=(50, 50),
            flags=cv2.cv.CV_HAAR_SCALE_IMAGE
        )

        # Draw a rectangle around the faces
        for (x, y, w, h) in faces:
            cv2.rectangle(frame_show, (x, y), (x+w, y+h), (0, 0, 255), 2)

        # Show the frame on the screen
        cv2.imshow("DragonBoard 410c Workshop", frame_show)

```

```
# Read in the next frame
retval, frame = vc.read()

# Exit program if the ESCAPE key is pressed
if cv2.waitKey(1) == 27:
    break

i += 1
```

Be sure to check your indentation! Once again, there are significant changes to our program, which we will explore in detail:

Our program will now require an additional input in the command-line. This will be the path to the cascade classifier XML file (we downloaded this in Phase 1). If our program does not receive enough arguments, it will terminate. Once the XML file is loaded, we initialize the cascade classifier OpenCV object.

We have added two new variables: “i” and “faces”. Variable “faces” will contain an array of detected faces. Variable “i” is simply a counting number; every time we step through the “while”-loop, we increment “i”. If “i” is a multiple of 5, we perform the facial recognition calculation. By only doing this computation every 5 frames, we speed up our program and smooth out our results.

The detectMultiScale function for cascade classifier objects is what does the facial recognition. We have provided some working values, but you are welcome to change these to see how it affects the facial recognition performance!

The portion of the code which writes the text “[LIVE]” has been removed.

Also, note that we’ve introduced a new frame variable called “frame_show”. While we convert variable “frame” into the grayscale color-space and perform the facial recognition on that, we draw the rectangles inside of “frame_show”, and then put “frame_show” on the screen. This explains why we are looking at a color video, and not a grayscale one.

Save your changes and exit the text editor. Return to the terminal and run your new program by entering the command. Remember to provide the XML file downloaded earlier:

Terminal

```
python detect_face.py frontalface.xml
```

You should see a window popup showing a live webcam feed, as well as a red square around any detected face in the video. Ta-da!

Phase 3: Emails and Text Alerts

In this phase you'll learn how to send email and text messages in Python. This will allow your home automation project to send alerts and notifications via the internet.

Install sendmail

To send emails and text messages from the DragonBoard 410c, you'll first need to install an email-sending utility, such as sendmail. Sendmail is an easy-to-use email-sending utility that implements various email internet protocols, such as SMTP (Simple Mail Transfer Protocol). The sendmail executable can be used to send emails from the command-line, but in our case we'll be using a Python module that uses SMTP to send email/text via Python programs.

In your terminal window, run the command below to install sendmail from the software repository:

Terminal

```
sudo apt-get install -y sendmail
```

On the current version of linux for the DragonBoard 410c, an additional step is required:

Terminal

```
sudo gedit /etc/hosts &
```

Update the first line of the file, as shown below:

gedit hosts

```
127.0.0.1 localhost
127.0.0.1 localhost.localdomain localhost linaro-alip
```

Run the command below to send yourself an email. Replace <Email Address> with your email address.

Terminal

```
/usr/lib/sendmail "<Email Address>" <<< "Test email!"
```

You should now have an email with the message 'Test email!' in your email inbox. It's very likely that this email will end up in your spam folder since it is coming from an unknown domain (linaro@linaroai-alip). Note that some company and university networks block the ports used by SMTP (25, 2525, 587, and 465), which will prevent emails from going out on those networks (we recommend trying this part on a home network).

Send an email

There are several ways to send an email in Python, but for this lab we'll use the 'email' package. It provides several capabilities that are well-suited to home-automation applications, and is easy-to-use.

In your terminal window, create a new file named 'send_email.py':

Terminal

```
gedit send_email.py &
```

Add the following lines of code and save your changes. Follow the comments and read below to see what each part of the code does. Replace <Insert Email Address> with your email address (preferably one you can access right now).

gedit send_email.py

```
#!/usr/bin/env python

# Import smtplib for the actual sending function
import smtplib

# Import the email modules we'll need
from email.mime.text import MIMEText

# Create a text/plain message
msg = MIMEText("Face detected!")
msg['Subject'] = "Your DragonBoard 410c has detected someone"
msg['From'] = "linaro@localhost"
msg['To'] = "<Insert Email Address>"

# Send the message via our own SMTP server (sendmail)
s = smtplib.SMTP('localhost')
s.set_debuglevel(1)
s.sendmail(msg['From'], [msg['To']], msg.as_string())
s.quit()
print "Email sent!"
```

Here's what each line of code above does:

- 'import smtplib' adds the library containing the 'smtplib.SMTP' function
- 'from email.mime.text import MIMEText' adds the library containing the 'MIMEText' function

- `'msg = MIMEText(...)` creates a new `MIMEText` object, which is an object representing a simple text email (with the message contained in quotes)
- The next 3 lines add a Subject, From, and To to the email object
- `'s = smtplib.SMTP('localhost')` creates a new `SMTP` object, which is an object able to send email via the SMTP protocol (using sendmail behind the scenes)
- `'s.sendmail(...)` sends the email content using the 'From' and 'To' strings entered above
- `'s.quit'` waits for the email to be sent then cleans up the `SMTP` object

Save your changes and exit the text editor. Return to the terminal and run your new program by entering the command:

Terminal

```
python send_email.py
```

Your program will print “Email sent” and should complete in a few seconds. If you see an error printed to the screen, check your program for errors. Remember that in Python you cannot mix tabs and spaces, so make sure all indentation are either all tabs or all spaces. Consult your workshop administrators for additional questions.

Now check your email inbox (or spam folder if you don't see a new message in your inbox). You should see an email from 'linaro@linaroaiip-alip' that says “Face detected”. Note that you could have put any email address on the 'From' line and it would appear as though it was sent from that person. The IP address of the sender is sent along with the email, though, which can be used to help identify the actual sender. Also, any replies to the email would go to the person indicated in the 'From' line (although localhost is a generic domain that is not listed in DNS servers, so replies to this test email won't be possible without some additional setup on the DragonBoard 410c).

Send a text + image email in Python

Now that you have sent a text email, you can extend this program by embedding an image to the email. For this part of the workshop, you'll need an image file that you've captured with the webcam. We will be using the "output.jpg" file created in Phase 2 of this workshop.

Copy your previous Python program to a new one, and open the new file up in your text editor.

Terminal

```
cp send_email.py send_email_pic.py
gedit send_email_pic.py &
```

Edit this file in **gedit** to include some new lines of code.

gedit send_email_pic.py

```
#!/usr/bin/env python

# Import smtplib for the actual sending function
import smtplib

# Import the email modules we'll need
from email.mime.text import MIMEText
from email.mime.image import MIMEImage
from email.mime.multipart import MIMEMultipart

# Create a multipart message
msg = MIMEMultipart()
msg.attach( MIMEText("Face detected!") )
msg['Subject'] = "Your DragonBoard 410c has detected someone"
msg['From'] = "linaro@localhost"
msg['To'] = "<Insert Email Address>"

# Try opening and attaching the image file
try:
    f = open("output.jpg", "rb")
    img = MIMEImage( f.read() )
    f.close()
    msg.attach(img)
except IOError:
    print "Error: Cannot find 'output.jpg'!"

# Send the message via our own SMTP server (sendmail)
s = smtplib.SMTP('localhost')
s.set_debuglevel(1)
```

```
s.sendmail(msg['From'], [msg['To']], msg.as_string())  
s.quit()  
print "Email sent!"
```

This program adds a few more lines of code:

- The new import lines add the 'MIMEImage' and 'MIMEMultipart' packages, used for sending image and multi-part emails (text and images)
- This time the email is created as a 'MIMEMultipart' object, with a 'MIMEText' object containing the text message attached as the first part of the message
- We attempt to open an image file "output.jpg" (which was generated in Phase 2). Then, the image is loaded into a MIMEImage object. We then safely close the image file, then attach the image to the email message object.
- If there is an error opening the file, we print out an error statement "Cannot find 'output.jpg!'". This is how a try-and-except statement works in Python.

Return to the command line and run this new program:

Terminal

```
python send_email_pic.py
```

Same as before, the program should output "Email sent" when it is finished, so check the syntax and spacing if you see any errors. Notes that in the example above, the path to the image is 'output.jpg', which means it expects the images to be in the same directory that the program was run in. Check your email inbox and see if the message has arrived with the image attached!

Send a text + image + date email in Python

Next we'll add a date/time string to the email, so you'll know exactly when the email was sent.

Copy your previous Python program to a new one, and open the new file up in your text editor.

Terminal

```
cp send_email_pic.py send_email_pic_date.py
gedit send_email_pic_date.py &
```

Edit this file in **gedit** to include some new lines of code.

gedit send_email_pic_date.py

```
#!/usr/bin/env python

# Import smtplib for the actual sending function
import smtplib

# Import time module to create a timestamp
import time

# Import the email modules we'll need
from email.mime.text import MIMEText
from email.mime.image import MIMEImage
from email.mime.multipart import MIMEMultipart

# Create a multipart message
msg = MIMEMultipart()
msg.attach( MIMEText("Face detected!") )
msg['Subject'] = "Your DragonBoard 410c has detected someone (%s)" \
    % time.strftime("%m/%d/%Y %H:%M")
msg['From'] = "linaro@localhost"
msg['To'] = "<Insert Email Address>"

# Try opening and attaching the image file
try:
    f = open("output.jpg", "rb")
    img = MIMEImage( f.read() )
    f.close()
    msg.attach(img)
except IOError:
    print "Error: Cannot find 'output.jpg' !"

# Send the message via our own SMTP server (sendmail)
```

```
s = smtplib.SMTP('localhost')
s.set_debuglevel(1)
s.sendmail(msg['From'], [msg['To']], msg.as_string())
s.quit()
print "Email sent!"
```

In this program we made the following changes:

- We have added 'import time' to include the common time functions.
- Now, the email's subject will include '(%s)' in the string, which Python will swap out with the text returned by 'time.strftime(...)'. This function is used to return the current date in a user-configurable format (for example, in our program: 12/13/2016 23:59).
- Note that the code uses a backslash '\ ' to escape the newline character, allowing the '%time.strftime(...)' to be placed on the next line. For this to work, the backslash must be the last character on the line (or you can remove the backslash and put both lines on the same line).

Run the new program just as you did the old programs before:

Terminal

```
python send_email_pic_date.py
```

You should now see an email that contains the date that the email was sent in the subject. Note that this uses the time on the DragonBoard 410c, so make sure that is configured correctly to ensure the time sent in emails is correct (refer to the APPENDIX for details on changing the DragonBoard 410c's time zone).

Send text messages in Python

Now that you know how to send emails from the DragonBoard 410c, sending text messages is a piece of cake (thanks to the phone service providers). All of the major cell carriers have email addresses that map to the phone numbers on their network (called the SMS Gateway Domain). The email address for a phone number is simply the 10-digit phone number @ the domain specified by the carrier. The table below shows the SMS Gateway Domains for the major carriers.

Carrier	Email Domain
AT&T	txt.att.net
Sprint	messaging.sprintpcs.com
Verizon	vtext.com
T-Mobile	tmomail.net
Cricket Wireless	mms.cricketwireless.net
Republic Wireless	text.republicwireless.com
U.S. Cellular	email.uscc.net
Virgin Mobile	vmobl.com

Determine the email address that maps to your phone number using the table above (ex. 8005551234@txt.att.net). Test that you have the correct email address by sending an email from your email address to your SMS email address. Now we'll try it on the DragonBoard 410c!

Because we will only be sending a simple text message, we can go back to the code use for just plaintext emails. Copy your previous Python program to a new one, and open the new file up in your text editor. Be sure to replace '<Insert Email Address>' and '<Insert Gateway Address>' with your email address, and the gateway address determined above, respectively.

Terminal

```
cp send_email.py send_text_msg.py
gedit send_text_msg.py &
```

Edit this file in **gedit** to include some new lines of code.

gedit send_text_msg.py

```
#!/usr/bin/env python

# Import smtplib for the actual sending function
import smtplib
```



```
# Import the email modules we'll need
from email.mime.text import MIMEText

# Create a simple email to be converted into a text message
msg = MIMEText("Face detected!")
msg['Subject'] = ""
msg['From'] = "<Insert Email Address>"
msg['To'] = "<Insert Gateway Address>"

# Send the message via our own SMTP server (sendmail)
s = smtplib.SMTP('localhost')
s.set_debuglevel(1)
s.sendmail(msg['From'], [msg['To']], msg.as_string())
s.quit()
print "Text message sent!"
```

The program above is almost identical to the 'send_email.py' program, only the subject is left blank (as text messages do not have a subject line). Additionally, the 'To' and 'From' addresses are different. Run the new program just as you did the old programs before:

Terminal

```
python send_text_msg.py
```

If you see errors printed to the screen, check your syntax and spacing. At this point, one of two things might occur:

- 1) Either the text message will be sent and will make its way to your phone, showing your email address as the sender,
- 2) Or the text message will be rejected by your carrier's spam filter and won't be delivered.

Unlike your email inbox, there is no spam folder for your text messages, so if the message is flagged by your carrier's spam filter, it won't be delivered. If this happens, the carrier will typically reply to the sender (your email address) with a message indicating that the email was flagged as spam. There are ways to get around this, including using a service like Twilio (www.twilio.com), or additional configurations to your DragonBoard 410c. Twilio is a paid service but is free initially when developing, and isn't too expensive.

Phase 4: Putting It All Together

In this last phase of the workshop, we will stitch together our facial recognition program with our image-attached email delivery program. Together, this will form the basic operation of a smart surveillance camera which may see growing use in the modern home.

You can copy your earlier programs, or manually type out the resulting program. Let's name our final script "smart_cam.py". Open it now in **gedit**:

Terminal

```
gedit smart_cam.py &
```

The combined code from your previous files may look like the following:

gedit smart_cam.py

```
#!/usr/bin/env python
import cv2, sys, smtplib

# Import the email modules we'll need
from email.mime.text import MIMEText
from email.mime.image import MIMEImage
from email.mime.multipart import MIMEMultipart

# Define constants
DEVICE_NUMBER = 0
IMAGE_FILE = "detected_face.jpg",

# Get XML file input
if len(sys.argv) > 1:
    XML_PATH = sys.argv[1]
else:
    print "Error: XML path not defined"
    sys.exit(1)

# Initialize the cascade classifier
faceCascade = cv2.CascadeClassifier(XML_PATH)

# Initialize webcam
vc = cv2.VideoCapture(DEVICE_NUMBER)

# Check if the webcam works
if vc.isOpened():
    # Try to get the first frame
    retval, frame = vc.read()
```

```

else:
    # Exit the program
    sys.exit(1)

i = 0
faces = []

# If the webcam read is successful, loop indefinitely
while retval:

    # Define the frame which the program will show
    frame_show = frame

    if i % 5 == 0:
        # Convert frame to grayscale to perform facial detection
        frame = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

        # Detect objects and return an array of faces
        faces = faceCascade.detectMultiScale(
            frame,
            scaleFactor=1.2,
            minNeighbors=2,
            minSize=(50, 50),
            flags=cv2.cv.CV_HAAR_SCALE_IMAGE
        )

        # If at least one face has been detected
        if len(faces) > 0:

            # Draw a rectangle around the faces
            for (x, y, w, h) in faces:
                cv2.rectangle(frame_show, (x, y), (x+w, y+h), (0, 0, 255), 2)

            # Save the frame as an image file
            cv2.imwrite(IMAGE_FILE, frame_show)

            # Exit the webcam while-loop
            break

        # Show the frame on the screen
        cv2.imshow("DragonBoard 410c Workshop", frame_show)

        # Read in the next frame
        retval, frame = vc.read()

        # Exit the webcam while-loop if the ESCAPE key is pressed
        if cv2.waitKey(1) == 27:

```

```

        break

    i += 1

# Once you have exited the webcam while-loop,
# create an email and try to attach the detected face image
msg = MIMEMultipart()
msg.attach( MIMEText("Face detected!") )
msg['Subject'] = "Your DragonBoard 410c has detected someone"
msg['From'] = "linaro@localhost"
msg['To'] = "<Insert Email Address>"

try:
    f = open(IMAGE_FILE, "rb")
    img = MIMEIMAGE( f.read() )
    f.close()
    msg.attach(img)
except IOError:
    print "Error: Cannot find", IMAGE_FILE

# Send the message via our own SMTP server (sendmail)
s = smtplib.SMTP('localhost')
s.set_debuglevel(1)
s.sendmail(msg['From'], [msg['To']], msg.as_string())
s.quit()
print "Email sent!"

```

Save your changes and exit the text editor. Now try running the program:

Terminal

```
python smart_cam.py frontalface.xml
```

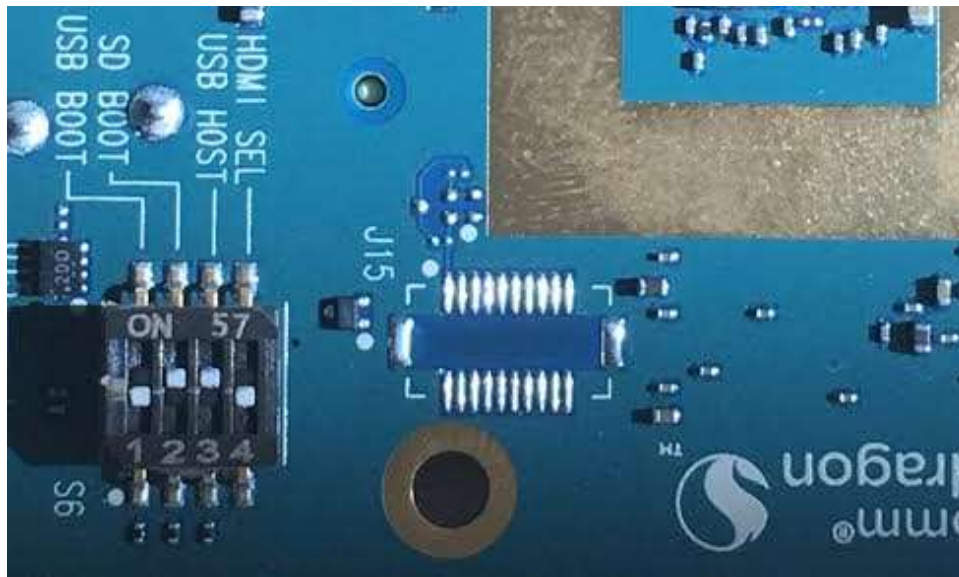
Then check if you received an email from your DragonBoard 410c with the facial recognition output image attached. If you have reached this point, congratulations! You have completed the “Home Automation with the DragonBoard 410c” workshop. By this point, you should have increased familiarity and comfort with using the Python scripting language as well as gained a basic understanding of the OpenCV library. The above program can be improved and extended in many ways to serve your needs. Some possible extensions you can explore on your own include:

- Optimizing the cascade classifier parameters to detect faces faraway and nearby
- Sending a text message alert if a face is recognized
- Training a classifier to treat certain faces as known and other faces as unknown
- Writing time-stamp information directly onto the image before emailing

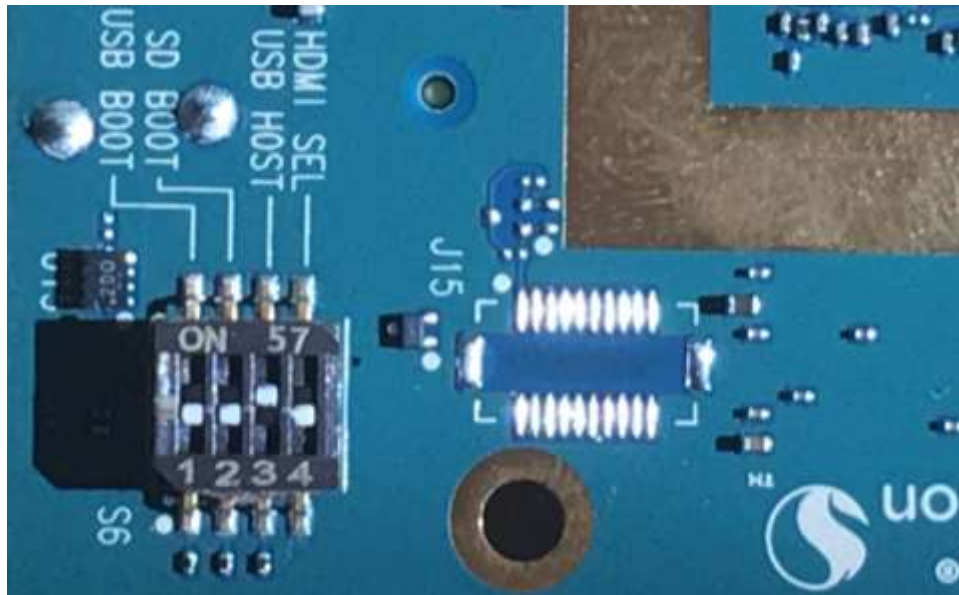
APPENDIX

Flashing Linux onto the DragonBoard 410c

1. Unplug every adapter and cord from the DragonBoard 410c, then insert the microSD card that contains the version of Linux that you desire (currently [Ubuntu](#) and [Debian](#) available).
2. Set the S6 switch on the DragonBoard 410c to 0-1-1-0 (SD Boot and USB Host switches set to “ON”)
*** See **Setting S6 Switches** *** below in the Appendix for more info.



3. Plug a USB mouse into either of the two USB connectors on the DragonBoard 410c.
4. Connect an HDMI monitor to the DragonBoard 410c with an HDMI cable and turn the monitor on.
5. Connect the power cable to the DragonBoard 410c.
6. When a menu appears on the screen, click on the install icon located on the top left of the window. Click YES to the prompt to confirm that you want to flash Ubuntu.
7. When successful, a window should say “OS Installed”.
8. Remove the SD card.
9. When you are ready, press OK to reboot.
10. Disconnect the power cable from the DragonBoard 410c.
11. Set the S6 switches so that ONLY “USB Host” is enabled (0-0-1-0).



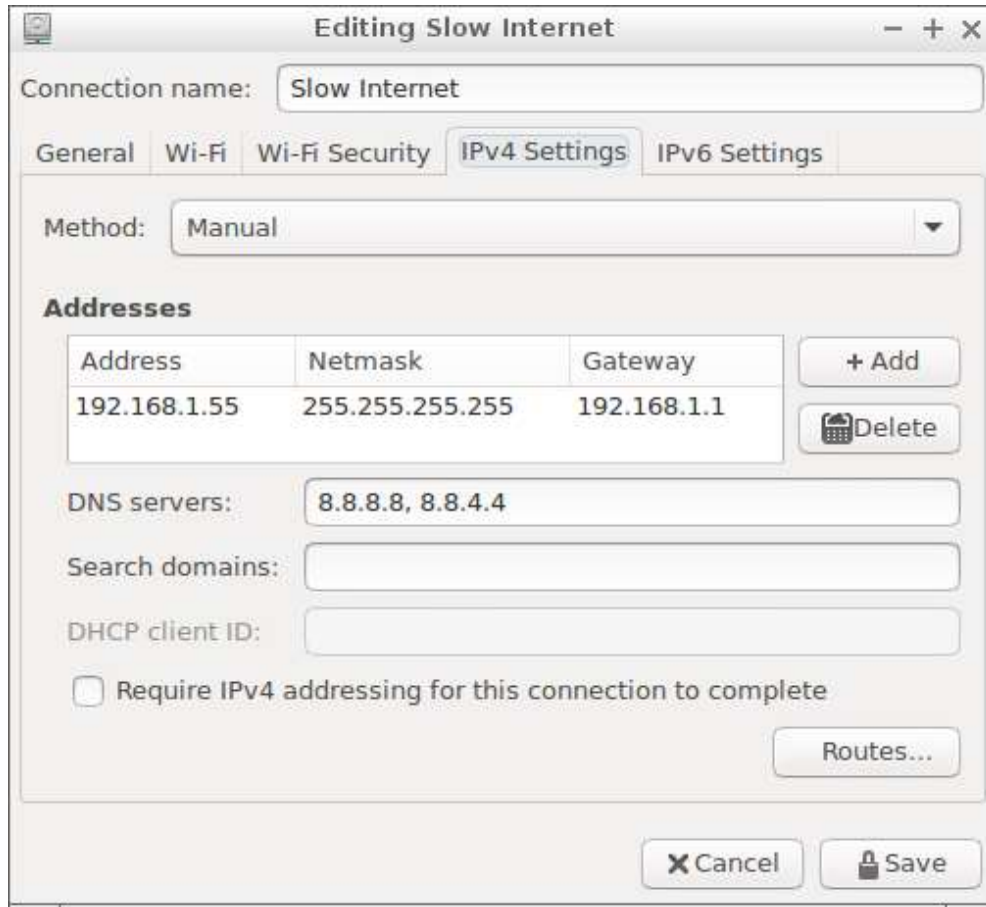
12. Reconnect the power cable to the DragonBoard 410c which will reboot into Ubuntu.

Setting S6 Switches

1. Ensure that the DragonBoard 410c is OFF and not plugged into power.
2. If you flip the DragonBoard 410c upside down, you will notice a small block of 4 switches near the HDMI port. Right next to this block you should see the letters “S6” printed on the board. Additionally, you should see a description of each switch printed on the board on the opposite side of the block.
3. Using a pen or other sharp object, carefully push the switches you desire to their correct positions. Note that the block has the word ON on the side that is considered ON. Flipping the switches to the other side would indicate that the switch is OFF.

Troubleshooting Wi-Fi Connection Issues

1. Make sure the DragonBoard 410c’s mac address matches the sticker printed on the bottom of the board
Open a terminal window and run: `echo /lib/firmware/wlan/macaddr0`
2. If it doesn’t match, follow the steps here:
https://developer.qualcomm.com/qfile/29540/lm80-p0436-44_set_up_wi-fi_mac_addr_dragonboard410c.pdf
3. If the network credentials work but the router never allocates the board an IP address, try disabling DHCP (example settings shown below), disable the adapter, then re-enable. Select a unique IP address, determine the gateway from a board/computer that is able to connect, and use Google’s DNS servers temporarily (8.8.8.8, 8.8.4.4).



4. If the network connection is successful but shell commands can't access the internet, open a web browser and try visiting a web page. There may be a login or agreement on the network that must be completed before access will be allowed.

Setting the DragonBoard 410c's time zone

One simple step. Follow the prompts after you run this command:

Terminal

```
sudo dpkg-reconfigure tzdata
```