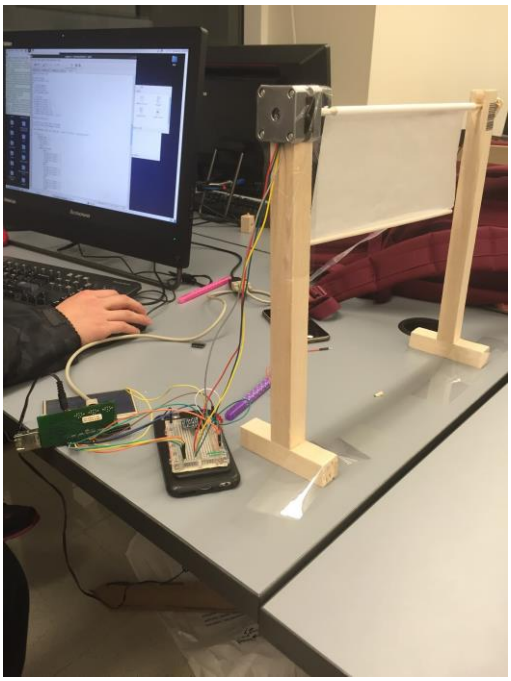

Boston University
College of Engineering
EC535 Spring 2016 Project Final Report
Team Smartain
Yida Zhang (yida), Jooyoun Hong (hongjooy), Sparsh Kumar (sparshk)

Abstract

In this project, we built a smart home application, “Smartain”. With our project, user can conveniently control the curtain under different modes. Under “Smart Mode”, The curtain and LED light could be automatically controlled base on the light condition, where the light condition is captured via lux sensor. Under “Manual Mode,” the curtain could be controlled manually by using a sliding bar in the LCD touch screen to control how much the curtain would be opened. Also, we can control LED light by clicking a button on touchscreen, and setup a scheduled time to open curtain with UI in touchscreen. The curtain moving speed could be changed as a user preference too.

Introduction

Our project can be placed in the smart home/building field. In this day and age more things are becoming automated because it will allow society to function more efficiently. We wanted to make a project that could impact society in a positive way, and be a little more unique compared to the other projects in the course.

We chose to make a smart curtain because it has the potential to be applied in the real world, and solve a variety of problems with managing curtains. Our curtain can also help people wake up in the morning by using natural sunlight. Even though most people use an alarm to wake up, it can still be uncomfortable hearing the same loud noise every morning. With our project, one can set an alarm to open up the curtain at a certain time so that the user can wake up with the sunlight. This will allow people to wake up more comfortably and with no loud noises. In addition, the user can control the speed of the curtain. So if one person wants the sunlight to come in slowly, they can adjust the settings for their alarm. If a person wants the sunlight come in quicker, then they can make the curtain rise faster.

Another way our Smart Curtain can be useful in the real world is making the whole process simpler. When pulling down a curtain, some people don’t know how to use it properly or have difficulty controlling it. Our project can control how much you want the curtain up or down on a touch screen. People will have an easier time using a touch screen with a nice user interface than pulling a string and getting it tangled. People with disabilities and senior citizens will more likely prefer using a touch screen than getting up and trying to pull down the curtain.

There is also an automated mode that determines when the curtain should be up or down. If the sunlight is too high then the curtain will be down, and the opposite applies when it’s too dark. Having automated curtains will allow people to not worry about adjusting their curtains in the first place. There is a mode where they can select this, and when this is on they have one less thing to worry about in their home. On the other hand, if they want to adjust the curtain themselves, then they can turn off “Smart Mode,” and control it in the “Manual Mode.”

This project has the potential to go very far and contribute to the overall smart home concept. It can be used to control multiple curtains in a home or a large building, save energy by allowing a certain amount of sunlight in a room, interact with other features of a room such as the lamps and AC, and analyze the outside weather to make better decisions regarding the position of the curtain. Initially it would be difficult to install our project in homes because you would have to customize the settings for each unique curtain, but eventually it could end up being a simple installation so that any home or business can use it. Updates to the software can be provided; this can consist of bug fixes, more efficient code, and new features. Given all the weather and sunlight data out there, we could use that to determine the optimal conditions for curtain positions.

Our project fixes many common issues regarding controlling a curtain, and it can be taken farther with the right tools, data, and software. Our project that we presented has two modes: Manual mode and Smart mode. In manual mode you can control the position of the curtain, turn on or off the LED, control the speed of the motor. In smart mode the curtain will automatically roll up or down depending on the value the lux sensor returns. We also have an alarm feature in which the curtain will roll up at a time set by the user.

Design Flow

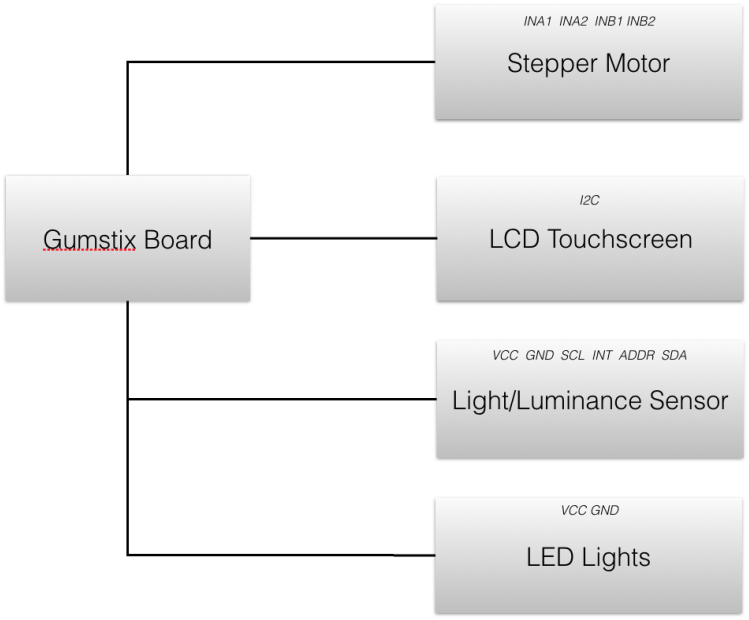

Figure 1. Design Flow Block Diagram

The Stepper motor is implemented for controlling the curtain. The light sensor is used to measure the amount of light in the room. The LCD panel provided from the class is used for user interface. Manual mode is merely controlled by user clicking button on the screen. When users set how much they want to open up the curtain through sliding bar, the percentage value is stored in variable and sent to the kernel when user presses the “Go” button. Other functionalities including “Smart Mode” enable, curtain speed, and alarm curtain are implemented similar way which whenever user clicks associated button, it sends the signal to the kernel module. Light sensor also has its own user level program that reads the value from the sensor and send the value to the kernel module. Then the kernel process the input data depends on the type of data by conditional statements.

Yida worked on the kernel module. He controlled the stepper motor based on input value from light sensor and Qt user input. Jooyoun designed the user interface on the Qt, managed the user input signal to the kernel module, and created the digital clock and alarm curtain functionality using Qt library. Sparsh worked on the code that interacted with the lux sensor and managed the LED light via GPIO.

Project Details

a. Stepper Motor

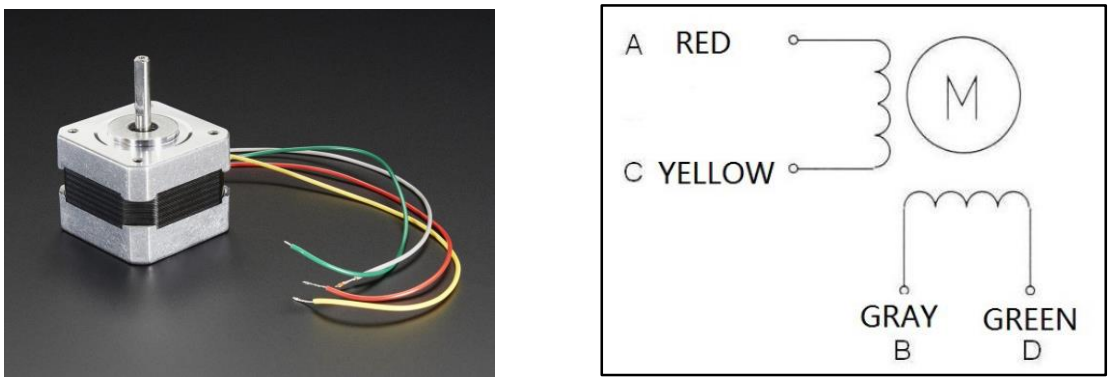

Figure 2. NEMA-17 size - 200 steps/rev - 12V 350mA [1][2]

AC97			
7.VCC	5.SDATA_OUT > GPIO30	3.SDATA_IN0 > GPIO29	1.GND
8.SYNC > GPIO31	6.NACRESET > GPIO113	4.CLK32 > GPIO9	2. X_BIT_CLK > GPIO28

Table 1. GPIO usage on gumstix board and the AC97 inputs

We used one stepper motor to control the curtain in this project. Since we used Gumstix as the main board, we could not use the compatible breakout board for this. We have to control this by purely changing the ends of the two H bridges. By using four GPIO inputs on the Gumstix board (GPIO 28, 29, 30 and 31), we can control the stepper motor and move it step by step with a sequential order of the four ends of two H bridges. We connect GPIO 28 to RED, 29 to YELLOW, 30 to GRAY and 31 to GREEN as shown above.

The next part is a connection between the GPIO control and the curtain control. If we want to open curtain by 50%, we will move the stepper motor 50*step_size = 600 steps (total steps), where the step_size is parameterized based on the model we build. To continuously enable stepper motor moving to the target we set, we used a loop to keep sending a sequential ordered GPIO signal. The stepper motor will keep moving until it moves to the total step we set.

To prevent motor swinging back and forth if it kept receiving different target percentages, we set another running flag to make sure the curtain or the motor would only execute next target signal only if it has finished the previous task. We initially had issues with this, but then we thought of this flag strategy to keep track of the curtain position. In initial testing, we noticed that the stepper motor would go to different positions in Smart mode if you kept activating the lux, but having this flag prevents this.

In the kernel module we programmed, it will read the information from the Lux sensor and LCD screen. What we do in kernel module is reading the node file contain signal string, which would start with a flag character together with an integer number. By reading the flag value, we can do something based on the signal such as switch to smart mode, open light, and move the curtain. By reading the

value after the flag, we know the detail in one specific action under one flag such as control the curtain to a specific percentage. We got a lot of our inspiration from the lab assignments in the class in regards to GPIO and reading node files.

b. LCD

The LCD Panel given in class is implemented for the user interface. The control tab contains *Light, Set Speed, Go*, and *Smart Mode* button. The *Light* functions is a simple switch to turn the LED light on or off. The *Set Speed* button is used to set up the speed of the curtain. The *Go* button is used to open/close the curtain after setting the "Open percentage" of the curtain with sliding bar. Finally, the *Smart Mode* button enables/disables the auto control of the curtain based on the lux value coming from light sensor. In the clock tab, it displays current time just like digital clock and contains alarm clock function. This function opens up the curtain at the time user set. The UI is designed to be intuitive. Instead of printing an on/off label sign on small screen, we change the color as blue for the "ready-state" and red for "active" state using Qt's QPalette library.

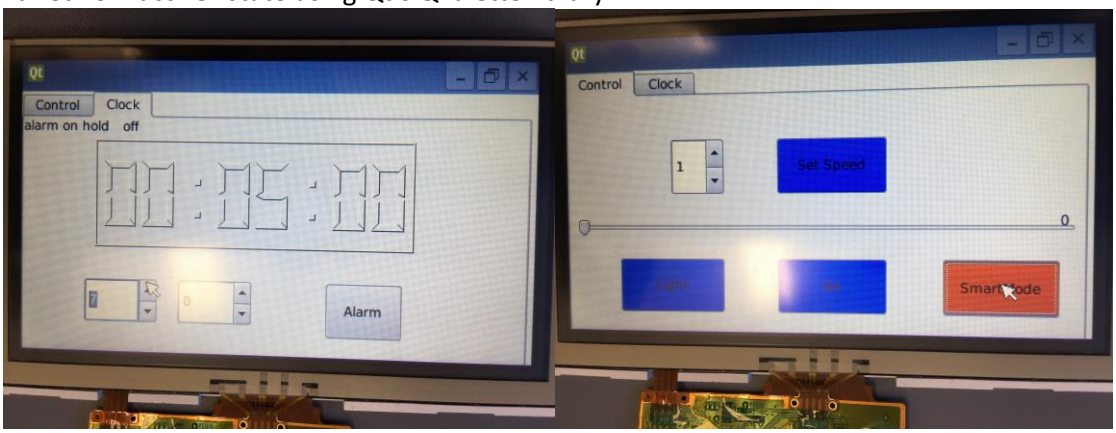

Figure3. User Interface

Most of the functionality is done by sending various signals to the kernel module. Using Qt's built in functionality "Signals & Slots", whenever the user interacts with LCD panel it calls associated callback function. The callback functions are writing values attached with letter for type identification (ex. "v10" means open curtain 10% and "a2" means set curtain moving speed to 2) to "/dev/mygpio" via "fputs" function. Then the kernel reads from the user level and differentiate each commands through the first character.

The Digital clock and alarm functionality are achieved through Qt's built-in library QTime and QTimer. QTimer is setup to and expires every 1 seconds, and timer expiration callback function calls QTime's currentTime() to reads the current time of the system to display. Also, if user sets the alarm, the value of hour and minute is stored, and get compared every seconds, and if the hour and minutes are the same as current time, it sends the "v99" (meaning open the curtain 99%) signal to the kernel module.

c. TSL2561 Lux Sensor

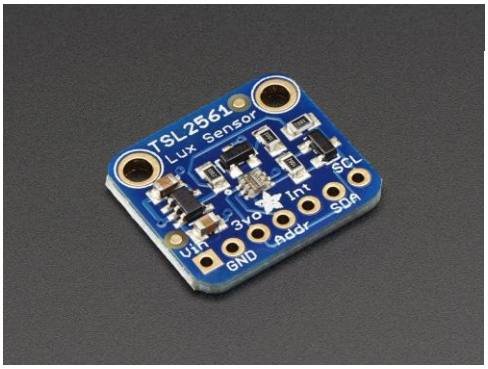

Figure 4. TSL2561 Lux Sensor

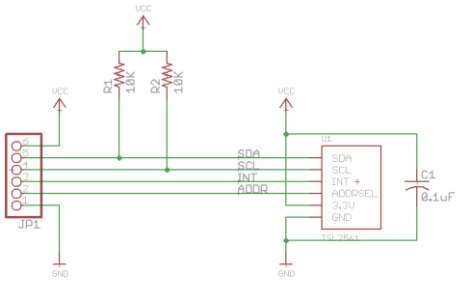

Figure 5. TSL2561 Lux Sensor Schematic

GPIO 101		Vin	GND
	GPIO 17	SCL	SDA

Table 2. The inputs of the I2C section of the Gumstix Board

We ordered a TSL2561 luminosity sensor on Adafruit. This is an advanced digital light sensor which can be used in a wide range of light situations. This sensor is more precise which allows it for exact lux calculations and can be configured for different gain/timing ranges to detect light ranges from up to 0.1 - 40,000+ Lux. The best part of this sensor is that it contains both infrared and full spectrum diodes. However, for our project we are only interested in the Lux values pertaining to sunlight and darkness. In figure 4 you can see which lux sensor we used and its inputs.

The TSL2561 communicates to the gumstix via Initial Inter Integrated Circuit (I2C). The retailer's website provided a simple example on how to use this senor on an Arduino. So after soldering the pins on the senor we tested it out with the Arduino board and we got it working. Now in order to program this for the gumstix, we had to look at how to program I2C in Linux. There were a few useful resources that helped me write up script in Linux. First there was a site that is dedicated to writing script in regards to i2c. It had a listing off all the functions in the i2c-dev.h file and how to use them. I found the addresses for the Lex sensor in the documentation and after some trial and error I was able to get values from the lux sensor. I wrote a user level file to get the commands, but it is possible that a kernel module could suffice as well. I also found an open source file that calculated the lux, and I incorporated that in my code by having it calculate the lux values after it received the data from the i2c functions. You can find the sources of my code in the appendix and in the scripts.

As mentioned earlier, this sensor communicates using an Inter-Integrated Circuit (I^2C) Protocol. The general concept is that the protocol allows "slave" devices to communicate with a "master" chip. In our case, the Gumstix is the master and the Lux sensor is the slave. When building this project, I required a few pins on the I2C section of the Gumstix, as shown in table 2. The Vin and ground pins were directly connected on the lux sensors. Sometimes the Vin pin on the Gumstix wouldn't work properly so I sometimes connected it to the Vcc on the AC97 section of the board. This sometimes shows that the software might not be the issue when your code isn't working. I also had to connect the SDA and SCL pins, as well as ground the ADDR. The gumstix will send the clock signal, and sometimes the lux sensor

will delay the clock so that it can get the data. The data then gets sent through the SDA and is recorded in an address.

d. LED Lights

This part is one of the easier components of our project. We mainly incorporated a few LED diodes to show the current state of the lux. In our manual mode, we have an option where you can turn on or off the LED. This can allow the user to choose which state they want it in based on their own needs. When we are in smart mode, the LED will automatically turn on when its dark and turn off when its bright. This is how the LED interacts with the lux sensor. We connected a few LED's in parallel through GPIO 101 on the I2C board. Anytime we were required to change the LED state, we simply had script in our kernel module that would set GPIO 101 to either high or low. We used examples from past lab assignments to write the script that would interact with the GPIO of the Gumstix board.

e. Kernel module

Inside kernel module, we made a node file: /dev/mygpio using command: mknod "/dev/mygpio c 61 0". We set up five GPIO settings inside kernel module: GPIO 28 - 31 for motor and GPIO 101 for LED. To keep rotating the motor, we defined a timer and modified the timer when it was complete. As a result, we can achieve the function of looping some code inside kernel module. In the timer done function, we set up GPIO 28 - 31 with sequential ordered signals to move the motor step by step. Also, a running flag is created so that the motor would only process the next task only if it has finished the previous one. Because we are not reading anything in user level, we did not define a lot of functions in buffer_read part. However, we need to read a lot of signals from user. So we have several if statements in buffer_write part. In buffer_write, we check the first character of input date to determine the type of task we are going to process. For example, user send 's1' to kernel module, we see 's' as a task type, which mean set up smart mode or not. The following '1' means we are going to turn on smart mode. We also set up four other flags, which stands for motor speed, lux meter read and percentage of curtain we want. To disable all manual functions in smart mode, we checker the smart mode flag whenever we are doing some changes to motor and LED. If smart mode is on, everything manual is not allowed. If smart mode is off, LED and motor are controlled by the Lux meter. We set up a threshold for the lux meter. In our project, curtain and LED would open if the Lux meter in under 30000. For lux data, we added a 'l' flag before the double value to indicate the value of the lux. This parts plays a major role in the smart mode as we have to continuously have to analyze the data for smart mode.

Summary

Our project is a Smart Curtain; also known as “Smartain.” Our project has two modes: manual mode and smart mode. In manual mode, one can control how much they want the curtain up or down, the speed of the curtain rising or falling, set up an alarm where the curtain will rise at a time the user has provided, and turn on or off the LED light. In smart mode, most of the features in manual mode are shut off and the curtain will rise and fall based on the input from the lux sensor. We were able to demo all of these features successfully on the demonstration day, and we appreciated the positive reception from not only the staff but also our peers.

Some of the issues that could arise is error handling for the stepper position, because it could get off and then the curtain might overshoot the distance. We did not think off all the potential errors so there could be times that the curtain position might seem inaccurate. The alarm feature might not even check the state of the curtain prior to opening, as we assume that the curtain will be all the way down prior to the alarm time. There could be times that the lux sensor might be inaccurate, as we do have to consider several other weather factors such as cloudy days. Given that, sometimes the lux sensor may not work due to clouds and precipitation blocking the sunlight. Therefore, we might have to potentially add more sensors for all the potential cases out there.

We believe that this is a solid start to a prototype. Even though we would have to consider several other factors such as curtain size, motor configurations and user interface, our project does have the potential to contribute to the smart home image. We would like to thank the professors, teaching assistants, family, and friends for giving us the advice and support for completing this project. We hope to use the skills in future courses and in industry. Thank You very much for taking the time to read this!

References

[1] Stepper motor image: https://www.adafruit.com/products/324
[2] Stepper motor datasheet: https://cdn-shop.adafruit.com/product-files/324/C140-A+datasheet.jpg
[3] How to control a stepper motor using GPIO: http://www.intorobotics.com/control-stepper-motors-raspberry-pi-tutorials-resources/
[4] TSL2561 Lux Sensor Image: https://cdn-shop.adafruit.com/970x728/439-00.jpg
[5] TSL2561 Lux Sensor Schematic: http://i43.tinypic.com/1z4haue.png
[6] TSL2561 Lux Sensor Product Information: https://www.adafruit.com/products/439
[7] TSL2561 Lux Sensor Datasheet: https://cdn-shop.adafruit.com/datasheets/TSL256x.pdf
[8] How to interact with I2C in Linux: https://www.kernel.org/doc/Documentation/i2c/dev-interface